Université de Mons
Faculté Polytechnique
Mathématique et Recherche Opérationnelle

Binary and Boolean Matrix Factorizations

Christos Kolomvakis

A thesis presented in partial fulfillment of the requirements for the degree of
Docteur en Sciences de l'Ingénieur et Technologies

Dissertation committee:

Professor Nicolas Gillis	University of Mons	Supervisor
Professor Arnaud Vandaele	University of Mons	Co-supervisor
Professor Fabian Lecron	University of Mons	Chair
Professor François Glineur	Université catholique de Louvain	
Professor Lieven De Lathauwer	KU Leuven	
Professor Pauli Miettinen	University of Eastern Finland	

Abstract

The processing of data has become an important part of many applications during the last years. Such applications include hyperspectral unmixing, recommender systems (for example in services like Spotify and Netflix), computer vision, text mining and audio source separation, to name a few. Many models and algorithms have been proposed, including linear dimensionality reduction (LDR), the perceptron, a precursor to modern neural networks, expectation-maximization, and the more recent algorithms for neural networks.

Our focus in this thesis is on a subclass of ML models called matrix factorization models. These are unsupervised algorithms whose goal is to decompose a given data matrix into a product of two smaller matrices, referred to as factors. Both factors are typically significantly smaller than the initial data matrix. There are multiple applications for matrix factorizations. Compression is one of them, since the factors are smaller than the original matrix. In text mining, matrix factorization retrieves topics from a collection of documents. In recommender systems, it creates groupings (clusters) of data points with common characteristics. As an example, when the entries of the input matrix represent a measure of the interaction between a user (rows of the matrix) and a product (columns of the matrix), these clusters represent groups with similar tastes. In addition, depending on the application, constraints can be imposed to the factors. An example of a constrained model is nonnegative matrix factorization (NMF) where the elements of the factors need to be nonnegative.

In this thesis, we focus on three models: (1) semi-binary Matrix Factorization (semi-bMF), where one factor has no constraints and the other can only have elements that are either 0 or 1, (2) Boolean matrix factorization (BMF) where both factors must only have elements that are either 0 or 1 and (3) Boolean matrix tri-factorization (BMTF), where we are factorizing into three factors. Furthermore, the matrix product used in the Boolean factorizations is the Boolean matrix product, which is different than the standard matrix product operation. This makes the computation harder but allows for better approximations and additional interpretability properties. For both models, we present new algorithms that are competitive with the state of the art and show that they can provide meaningful results in several applications, including topic modeling, image analysis, and clustering.

Acknowledgements

At the end of my PhD studies I want to reflect on the people that have supported me on this lengthy and intense journey.

I want to extend my thanks to my PhD advisor, Nicolas Gillis, who one day suddenly saw an email from a person with a weird name and surname, and was open to communication and accepted me in his team. His guidance and his advice were instrumental for the completion of the thesis. I also want to thank Arnaud Vandaele, my co-advisor for his help, his advice and his general down-to-earth attitude.

I am grateful to have been part of a group full of cheerful and interesting people. I want to thank them, by chronological order of when I met each one: Nicolas Nadisic, Olivier, Maryam, Pierre, Atharva, Hien, Giovanni Seraghiti, Giovanni Barbarino, Subhayan, Timothy, Florian, Stefano, Amjad and Alexandra.

I want to give special thanks to my parents, my brothers, and many others for their support and the good times that we have been having together.

I would also like to thank Emerance Delacenserie and Laurence Theunis for their help in navigating the post-PhD non-academic world.

I want to thank the jury members for agreeing to evaluate this thesis and for their useful comments.

Finally, I want to give my utmost thanks and gratitude to Hajaar for all the beautiful moments we have shared together these last years, for always supporting me and for always being there.

I acknowledge the support by the F.R.S.-FNRS and the FWO (EOS, O005318F-RG47), and the European Research Council (ERC, consolidator grant no 101085607).

Contents

Abstract **3**

Contents **9**

Notation **11**

List of Figures **15**

List of Tables **17**

1 Introduction **19**
1.1 Contributions and thesis outline 21

2 Robust Binary Component Decompositions **23**
2.1 Introduction 23
2.2 Models and Algorithms from [70] 24
2.3 Robust versions of the algorithms 30
2.3.1 Robust SSCD 30
2.3.2 Robust ASCD and robust ABCD 33
2.4 Gradient methods for the SDP problems 35
2.4.1 First-order method for SSCD 35
2.4.2 First order method for ASCD 36
2.5 Burer-Monteiro (B-M) approach 38
2.5.1 B-M on the SSCD problem 38
2.5.2 B-M on the ASCD problem 39
2.6 Numerical Experiments 40
2.6.1 Original experiments from [62] 41
2.6.2 Comparison of the B-M approach versus the PG approach . . . 44
2.7 Conclusion 45

3 Novel Algorithms for Boolean Matrix Factorization **47**
3.1 Boolean matrix factorization (BMF) 48
3.1.1 Illustrative example 49
3.1.2 Previous works 50

3.2 Alternating optimization (AO) for BMF 54
3.2.1 IP formulation for BMF subproblems 54
3.2.2 AO for BMF 55
3.2.3 Initialization of AO 55
3.3 Combining multiple BMF solutions 57
3.3.1 MS-Comb-AO 57
3.3.2 Tree-BMF 58
3.4 Greedy BMF heuristics 58
3.4.1 Greedy Boolean LS 59
3.4.2 Greedy combining methods 60
3.4.3 Custom data structure for Boolean vectors and matrices in C++ 62
3.5 Numerical Experiments 64
3.5.1 Comparison of our `bitmatrix` data structure 64
3.5.2 Experiments from the datasets in [52] 65
3.5.3 Application to topic modeling 71
3.5.4 Application on facial images 84
3.6 Conclusion 88

4 Boolean Matrix Tri-Factorization **89**
4.1 Introduction 89
4.2 Boolean matrix tri-factorization 90
4.3 Identifiability via Orthogonality 91
4.4 Block-coordinate descent for BMTF 92
4.4.1 Introduction 92
4.4.2 Imposing sparsity 93
4.4.3 Generating sparser and more expressive solutions 93
4.4.4 Initialization of the algorithm 94
4.5 Numerical Experiments 95
4.5.1 Experiments on synthetic data 95
4.5.2 Clustering a real dataset with BMTF 96
4.6 Conclusion 97

5 Conclusions **99**
5.1 Summary of the contributions 99
5.2 Future Work 100

Notation

$\mathbb{R}$	Set of real numbers
$\mathbb{R}^m$	Set of real column vectors with m elements
$\mathbb{R}^{m \times n}$	Set of real matrices with dimensions $m \times n$
$\mathbf{A}$	Matrix $\mathbf{A}$
$\mathbf{A}(:, k)$	k–th column of matrix $\mathbf{A}$
$\mathbf{A}(k, :)$	k–th row of matrix $\mathbf{A}$
$\mathbf{A}(:, \mathcal{I})$	Subset of columns of matrix $\mathbf{A}$ specified by the indices in the set $\mathcal{I}$
$\mathbf{a}$	Vector $\mathbf{a}$
$\mathbf{a}^T$	Transpose of vector $\mathbf{a}$
a_i	Element i from vector $\mathbf{a}$
$a_{i,j}$	Element i,j from matrix $\mathbf{A}$
$\mathbf{A} \circ \mathbf{B}$	Boolean matrix product of $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product of $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} \odot \mathbf{B}$	Elementwise (or Hadamard) product of $\mathbf{A}$ and $\mathbf{B}$
$\{\mathbf{a}_i\}_{i=1}^r$	Collection of vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_r$
Δ^r	Probability simplex, the set $\{\tau \mid \tau_i \geq 0 \text{ for all } i, \sum_{i=1}^r \tau_i = 1\}$
$\mathbf{I}_n$	Identity matrix of dimensions $n \times n$
$\text{sign}(\mathbf{x})$	Elementwise sign operator that sets an element to -1 if it is negative, to $+1$ if it is nonnegative
$\mathcal{S}^n$	The set of symmetric matrices in $\mathbb{R}^{n \times n}$
$\mathbf{X} \succeq \mathbf{0}$	$\mathbf{X}$ is positive semi definite
$\mathbf{X} > 0$	$\mathbf{X}$ has positive elements
$\text{diag}(\mathbf{X})$	A column vector that contains the diagonal elements of $\mathbf{X}$
$\mathcal{E}_n$	The elliptope of dimension n, that is $\{\mathbf{X} \in \mathcal{S}^n : \text{diag}(\mathbf{X}) = \mathbf{e} \text{ and } \mathbf{X} \succeq \mathbf{0}\}$
$\text{orth}(\mathbf{A})$	Orthogonal basis column space of $\mathbf{A}$
$\text{tr}(\mathbf{A})$	Trace of square matrix $\mathbf{A}$, the sum of its diagonal elements
$\text{vec}(\mathbf{A})$	Vectorization of matrix $\mathbf{A}$ (stacking of the columns of $\mathbf{A}$, from left to right, into a vector)
$\|\mathbf{A}\|_2$	Spectral norm of the matrix $\mathbf{A}$ (its greatest singular value)
$\|\mathbf{A}\|_F$	Frobenius norm of matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$: $\sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{i,j}^2}$
$\|\mathbf{a}\|_2$	l_2 norm of vector $\mathbf{a} \in \mathbb{R}^n$: $\sqrt{\sum_{i=1}^n a_i^2}$
$P_S(\mathbf{X})$	Projection of $\mathbf{X}$ to the set S: $P_S(\mathbf{X}) = \arg\min_{\mathbf{Y} \in S} \|\mathbf{X} - \mathbf{Y}\|$
$\sigma_i(\mathbf{A})$	The i–th singular value of matrix $\mathbf{A}$

$\min(1, \mathbf{A})$	Entrywise application of the minimum between 1 and $a_{i,j}$
$\max(1, \mathbf{A})$	Entrywise application of the maximum between 1 and $a_{i,j}$
$\mathbf{y} \leq \mathbf{x}$	For $\mathbf{y}, \mathbf{x} \in \mathbb{R}^n$, it holds that $y_i \leq x_i$ for all $i \in \{1, \ldots, n\}$
$\mathbf{Y} \leq \mathbf{X}$	For $\mathbf{Y}, \mathbf{X} \in \mathbb{R}^{m \times n}$, it holds that $y_{i,j} \leq x_{i,j}$, for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$

Acronyms

ABCD	Asymmetric binary component decomposition	p. 29
ALS	Alternating least squares	p. 42
AO	Alternating optimization	p. 40
ASCD	Asymmetric sign component decomposition	p. 27
B-M	Burer - Monteiro	p. 24
BCD	Block-coordinate descent	p. 92
bMF	Binary matrix factorization	p. 22
BMF	Boolean matrix factorization	p. 21
BMTF	Boolean matrix tri-factorization	p. 22
BTF	Boolean tensor factorization	p. 53
BoolLS	Boolean least squares	p. 54
CMF	Coupled matrix factorization	p. 42
ELBMF	Elastic boolean matrix factorization	p. 71
IP	Integer program	p. 47
IPM	Interior point method	p. 42
MF	Matrix factorization	p. 19
MTF	Matrix tri-factorization	p. 20
MIP	Mixed integer programming	p. 63
MS-AO	Multistart alternating optimization	p. 57
MS-Comb-AO	Multistart, combination and alternating optimization	p. 58
NMF	Nonnegative matrix factorization	p. 20
NMTF	Nonnegative matrix tri-factorization	p. 89
PCA	Principal component analysis	p. 20
PG	Projected gradient	p. 36
SBCD	Symmetric binary component decomposition	p. 27
Semi-bMF	Semi-binary matrix factorization	p. 21
SDP	Semi definite programming	p. 23
SNR	Signal to noise ratio	p. 42
SSCD	Symmetric sign component decomposition	p. 24
SVD	Singular value decomposition	p. 20

List of Figures

2.1 Illustration of the approach of Kueng and Tropp [70] where the convex hull of $\mathbf{W}(:,1)\mathbf{W}(:,1)^\top$ and $\mathbf{W}(:,2)\mathbf{W}(:,2)^\top$ is a polyhedral face of the elliptope. This is an example with $r = 2$, and $m = 3$. Image adapted from [70]. 25
2.2 Picture showing an iteration of the algorithm for SSCD. This example is for $r = 3$. Image adapted from [70]. 26
2.3 Mean Hamming Distance over 20 trials. The legend includes the average running time at SNR 20dB. 43

3.1 Venn diagram showing the overlapping communities that animals are assigned to. 50
3.2 Venn diagram showing the overlapping communities that animal characteristics are assigned to. 50
3.3 Sample Tree BMF with depth = 2. The order of the function calls for this tree is depth-first postorder. Initially, the leaf nodes solve instances of MS-Comb-AO and then propagate their solutions to the parent node, for it to combine. This procedure continues until we reach the root node, when the final factors are returned. 59
3.4 Facial features extracted by AO-BMF on the CBCL data set for $r = 20$. 85
3.5 Facial features extracted by Greedy-Comb on the binarized CBCL data set $\hat{\mathbf{X}}$. 86
3.6 Facial features extracted by Greedy-TreeBMF on the binarized CBCL data set $\hat{\mathbf{X}}$. 87

List of Tables

3.1 Average execution time in seconds over 30 trials of Gurobi to solve noisy BoolLS problems (3.4) for various values of m and r. 55
3.2 Run times to combine N rank-one factors for a 105-by-105 matrix with $r = 5$ (namely, the apb data set, see Section 3.5). 58
3.3 Results of the Boolean matrix multiplications for the multiple sizes considered and all the data structures considered. Average run times for 10 such multiplications. 64
3.4 Results of the Boolean matrix multiplications for the multiple sizes considered and all the data structures considered. Average run times for 10 such multiplications. 65
3.5 Four binary real-world data sets. 65
3.6 Four binary real-world incomplete data sets. 65
3.7 Objective function $\|\mathbf{M} \odot (\mathbf{X} - \min(1, \mathbf{WH}))\|_F^2$ of the best solution found by various algorithms in [52, Table 4, page 20]. 66
3.8 Best result obtained with the method in [33] (left), and result of this best solution improved by AO (right). The numbers indicate the difference compared with the best values in Table 3.7. A negative value means an improvement, a positive value means a worse solution. . . . 66
3.9 Best result obtained with the method in [21] (left), and result of this best solution improved by AO (right). The numbers indicate the difference compared with the best values in Table 3.7. 67
3.10 Results for all our proposed methods for both timelimits considered. This Table is for the datasets with no missing elements. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined. 68
3.11 Part of our results for all our proposed methods for both timelimits considered. This Table is for the datasets with missing elements. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined. 69
3.12 The remaining part of our results for all our proposed methods for both timelimits considered. This Table is for the datasets with missing elements. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined. . 69

3.13 The three subsets of **X** we will consider in this section and the parameters that we used to create them. 71
3.14 Best results on the relative errors for all the document subsets and all the different algorithms considered. In parenthesis, we note the mean time over all Monte Carlo trials (if multiple trials are performed for a method). In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined. . 72
3.15 Best results on the relative errors for all the document subsets and all the different algorithms considered. In parenthesis, we note the mean time over all Monte Carlo trials (if multiple trials are performed for a method). In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined. . 73
3.16 Best results on the number of 1s covered, for all the document subsets and all the different algorithms considered. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined. 73
3.17 List of topics retrieved and number of times each topic appears among all the 10 runs. 75
3.18 Table for topics 1 - 5 after using AO-BMF. 75
3.19 Table for topics 6 - 10 after using AO-BMF. 76
3.20 Table for topics 1 - 5 after using MS-Comb-AO. 76
3.21 Table for topics 6 - 10 after using MS-Comb-AO. 77
3.22 Table for topics 1 - 7 after using Greedy-Comb. 79
3.23 Table for topics 8 - 14 after using Greedy-Comb. 80
3.24 Table for topics 15 - 20 after using Greedy-Comb. 81
3.25 Table that compares the best relative error of most of the methods tested for the binarized CBCL dataset. In parenthesis, we note the mean time over all Monte Carlo trials. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined. 84

4.1 Percentage of time the ground-truth factors, or zero errors, are found among the 50 Monte Carlo runs on synthetic data sets. In brackets, we report the percentage of entries found wrong on average for the factors **W**, **S** and **H**, and report the average relative error. 96
4.2 Clusters of animals. 97
4.3 Clusters of characteristics. 97

Chapter 1

Introduction

Artificial intelligence (AI) is a very influential discipline that originally started in the 1950s. It was Alan Turing, in his work in 1950 [112], that started the discipline and first posed the question "Can machines think?". In this seminal work, he presents the concept of the "Turing test". One of its definitions is: Given a text transcript between a machine and a human, can someone reliably distinguish the machine from that conversation? The machine passes the test if the human cannot distinguish the human from the machine in the transcript.

Since then, there have been many works on AI, with distinct methodologies. One approach that was once popular is *Expert Systems* that tried to emulate the decision making process of humans. This was done by having a *knowledge base*, through which it derived facts and rules, and an *inference engine*, which made it deduce new facts [100]. Periods of intense research on AI were often followed by many periods referred to as "AI winters", where, due to discouraging results, there were severe funding cuts to AI projects.

Machine Learning (ML) [12] is a subfield of AI, where statistical algorithms are used to learn parameters from data. ML is also often used erroneously as a synonym to the whole discipline of AI. The current AI boom that started in the 2010s has made immense changes in the directions of research and in the applications and services created by using AI. Deep learning, a subfield of machine learning that uses neural networks for ML tasks, grew in popularity during the 2010s. An important work is in [69] where a neural network (NN) is shown to outperform the state of the art in the task of visual recognition. Although works in NNs did not start with this paper (consider for example the seminal papers [84, 96, 99] which were released many years before), the hardware needed for the neural networks to perform all their necessary operations became available around the 2010s, which allowed the growth of deep learning. Since then, the AI boom has continued, with the recent development and scaling of large language models (LLMs) being considered a very important milestone [19, 61, 115].

This thesis focuses on a subclass of unsupervised models called Matrix Factorization (MF) models where the data are stored in a matrix. Given a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$,

where each column represents a data point, we want to factorize it as the product of two matrices, $\mathbf{W} \in \mathbb{R}^{m \times r}$ and $\mathbf{H} \in \mathbb{R}^{r \times n}$ where $r \ll \min(m, n)$, such that $\mathbf{X} \approx \mathbf{WH}$. We call r the rank of the factorization. Matrix factorizations are unsupervised learning algorithms (although semi-supervised and supervised variants exist). In contrast, supervised learning algorithms need labels attached to the data points for prediction tasks. An example of a matrix factorizations include the truncated Singular Value Decomposition (SVD). These are examples of unconstrained matrix factorizations and they can be computed in polynomial time. Given the application at hand, one may want to impose constraints on the factors. For example, in hyperspectral imaging, we want the factors to have nonnegative elements, because of physical interpretations of the factors, and we can use Nonnegative Matrix Factorization (NMF) [46]. In facial image reconstruction, sparse constraints on the factors (that is, restricting them to have as few non-zero elements as possible) can also be introduced together with nonnegativity. This gives rise to Sparse NMF [59].

Typically, noise is present, and the MF problem is cast as follows:

$$\min_{\mathbf{W} \in D_{\mathbf{W}}^{m \times r}, \mathbf{H} \in D_{\mathbf{H}}^{r \times n}} d(\mathbf{X}, \mathbf{WH}), \tag{1.1}$$

where $D_{\mathbf{W}}$ is the domain of the elements of the matrix $\mathbf{W}$, $D_{\mathbf{H}}$ is the domain of the elements of the matrix $\mathbf{H}$, and $d(\mathbf{X}, \mathbf{WH})$ is a cost function measuring how close the product of the factors are to the data matrix. The cost function is chosen according to the application. Functions that can be chosen include the Frobenius norm or the Kullback-Leibler distance [57]. In many cases, constrained matrix factorization problems are NP-hard to solve [88, 116]. Hence we restrict ourselves to finding two factors that approximate the data matrix $\mathbf{X}$ as well as possible, typically looking for locally optimal solutions. Examples can be found in [1, 48, 60, 74, 125].

Constrained MF has successfully been applied in a wide variety of applications such as document classification, community detection, hyperspectral unmixing, and recommender systems, to cite a few; see, e.g., [46, 83, 113]. Other constrained MF models that can be considered depending on the problem include variants of Principal component analysis (PCA), such as sparse [126] and robust [25] PCA, semi-NMF where only one factor is non-negative [48, 109] and orthogonal NMF, where one of the non-negative factors is also constrained to have orthonormal columns [4, 120].

A closely related problem to matrix factorization is matrix completion (MC) [24, 95]. In MC, elements of the data matrix are missing. This is a realistic assumption, and one such example is in a user-movie dataset as most users have not watched most movies in the data base. The goal of MC is to estimate the missing elements of the matrix, given the already observed elements, under the low-rank assumption of the input matrix. As is the case with MF, in MC one can also add constraints to the factors, depending on the application. Nonnegative matrix completion (NMC) is an example of a constrained matrix completion task [37, 108, 122].

Finally, let us mention matrix tri-factorization (MTF), where the input matrix is factorized into three factors instead of two [36, 124]. MTF will be the focus of Chapter 4. An MTF is defined by two ranks r_1 and r_2, trying to decompose $\mathbf{X}$ as follows

$$\mathbf{X} = \mathbf{WSH}. \tag{1.2}$$

Here, $\mathbf{W} \in D_{\mathbf{W}}^{n \times r_1}$, $\mathbf{S} \in D_{\mathbf{S}}^{r_1 \times r_2}$ and $\mathbf{H} \in D_{\mathbf{H}}^{r_2 \times m}$, and $D_{\mathbf{S}}$ is the domain of the elements of the matrix $\mathbf{S}$. Note that the number of columns of $\mathbf{W}$ $(= r_1)$ and rows of $\mathbf{H}$ $(= r_2)$ can be different, as opposed to standard MF. The factors $\mathbf{W}$ and $\mathbf{H}$ are linked through a middle factor $\mathbf{S}$. For MTF to make sense, constraints need to be imposed on the factors $\mathbf{W}$ and $\mathbf{H}$; see Chapter 4 for more details. The computation of three factors makes MTF more complex than MF to compute, but it has the advantage to be more flexible in interpreting data. For example, in community detection, MF clusters the column data points and the row data points of the input matrix into r clusters (that is, r clusters for column data points and r clusters for row data points), corresponding to the r rank-one factors, $\mathbf{W}(:, k)\mathbf{H}(k, :)$ for $k = 1, 2, \ldots, r$. On the other hand, MTF

- allows for a distinct number of clusters for the column and the row data points. More specifically, the column data points are grouped into r_1 clusters in $\mathbf{W}$, while the row data points are grouped into r_2 clusters in $\mathbf{H}$. For example, in a movie-user data set, there could be a different number of communities of movies and users.
- links these clusters together via the factor $\mathbf{S}$: since $\mathbf{X} = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{S}(i, j)\mathbf{W}(:, i)\mathbf{H}(j, :)$, the i-th cluster of the column data points interacts with the j-th cluster of the row data points by the value $\mathbf{S}(i, j)$. In MTF, the two groups of clusters can be linked in multiple ways, according to the elements of $\mathbf{S}$. MF is a special case of MTF where $\mathbf{S}$ is the identity matrix.

1.1 Contributions and thesis outline

This thesis focuses mainly on algorithms for matrix factorizations with binary factors.

In chapter 2, we focus on semi-binary matrix factorization (semi-bMF). Semi-bMF factorizes a dataset into a factor with real elements and a factor with binary elements. Our focus in this chapter is an extension of the works in [70, 71], where the authors present four matrix factorizations, which, under normal circumstances, are difficult to compute. For all four factorizations, the authors present algorithms and conditions under which the proposed algorithms can compute the ground truth in polynomial time. However, these can only be applied in the ideal case where the input dataset satisfies all these conditions, which are very restrictive. The algorithms fail in the slightest presence of noise. Furthermore, these methods cannot be scaled to handle medium and large-sized datasets. In this chapter, we propose extensions that allow the handling of noisy data and address scalability [62]. We will also present an unpublished work that improves the scalability of [62] with the help of the Burer-Monteiro approach [15, 29].

The second problem we focus on in this thesis is the design of algorithms for Boolean Matrix Factorization (BMF), in Chapter 3. BMF is used to factorize matrices with elements that are 0 or 1. This factorization uses a special version of the matrix product called *Boolean matrix product* which uses logical operations from Boolean algebra instead of the standard addition and multiplication (it uses the Boolean-OR and Boolean-AND, respectively.). BMF can be used in a plethora of scenarios.

For example, in recent years, it has often been the case that websites no longer use rating systems such as zero to five stars, or a rating system from 0 to 10. Spotify, Youtube and Instagram are only but a few websites that now invite users to use just "likes" to indicate preferences. Likes and dislikes can be modeled with 1s and 0s, respectively. Other than recommender systems, BMF can also be applied in bioinformatics [54, 78], role mining [79, 114], and computer vision [73]. A matrix factorization model that is closely related to BMF is binary Matrix Factorization (bMF). bMF is also used for datasets with elements in $\{0, 1\}$, but uses the standard addition and multiplication. However, bMF has drawbacks in comparison to BMF. This allows for greater interpretability than bMF and smaller approximation errors. Both bMF and BMF problems are NP-hard to solve [88]. We show in more detail how BMF allows for greater expressivity than bMF. Then, we present our algorithms based on approaches that use commercial solvers like Gurobi [53], as well as greedy approaches. We will be comparing with the state of the art, and we will also be showing how our algorithms perform in various applications. This chapter is based on our previous work in [63] as well as a journal paper that is in preparation (add the journal citation here).

Finally, in Chapter 4, we propose Boolean Matrix Tri-Factorization (BMTF). BMTF factorizes a binary matrix into the Boolean matrix product of three factors. We provide uniqueness conditions for BMTF, propose an algorithm and present an application in clustering for an animal dataset [39]. Our motivation in using BMTF instead of BMF for a clustering task is twofold: First, to be able to cluster the row data points and the column data points of the input matrix into different numbers of clusters. Second, the middle factor $\mathbf{S}$ provides the relations between the two types of clusters (row data points and column data points) and gives more insight into the data. The results in this chapter are based on our work in [64].

We conclude the thesis with Chapter 5, where we present our conclusions from this thesis and give suggestions for possible future work.

For all the experiments that are presented in this thesis, we will be providing links, where the interested reader may download the codes and replicate our experiments.

Chapter 2

Robust Binary Component Decompositions

Semi-binary matrix factorization (semi-bMF) is a matrix decomposition model where the elements of one factor are binary. Semi-BMF can be interpreted as a generalization of k-means, and can be employed in clustering problems such as community detection. In the absence of noise, Kueng and Tropp [70, 71] have recently proposed a provably correct algorithm for semi-BMF that requires to solve semidefinite programs (SDPs). In this chapter, we extend their approach in the presence of noise. Moreover, since standard solvers for SDP rely on interior-point methods and do not scale well, we also propose first-order methods to reduce the computational costs, to the expense of finding a suboptimal solution (in our case, not finding all of the ground truth). We test our new algorithms on synthetic data, and show that they compare favorably with the state of the art. The content of this chapter is mainly from [62]:

- C. Kolomvakis and N. Gillis, "Robust Binary Component Decompositions," ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

Due to the space limitations for this paper, some parts that were necessary in the analysis of our methods were skipped. These will be explained in more detail in this chapter. We will also discuss a new method that allows for greater scalability, which improves on our results in [62] that we have not published over the course of this PhD.

2.1 Introduction

In [70] and [71], the authors describe exact algorithms that can compute semi-BMF, when the binary factor has elements in either $\{0, 1\}$ or $\{\pm 1\}$, as long as the input matrix satisfies specific properties. These algorithms are shown not only to be polynomial in the dimensions of the input matrix, but they can also recover the unique

ground truth factors. The drawback of their algorithms, however, is that they can only be applied in the noiseless case, rendering them useless in practice. In addition, these algorithms require solving SDPs, for which standard solvers rely on interior-point methods (IPMs) [91], and hence are not scalable. Correlation matrices play an important role in these factorizations. These matrices describe the correlation between multiple variables, they are symmetric and psd, have elements in $(-1, 1)$ and the diagonal is always full of 1s.
Our goal is twofold:

1. Extend the models and algorithms proposed in [70, 71] to handle noisy input matrices.

2. Design algorithms that are less demanding in terms of memory needs and time.

The chapter is organized as follows. In Section 2.2, we recall the models and algorithms from [70, 71]. In Section 2.3, we show how the model can be adapted in the presence of noise and describe the new algorithms. In Section 2.4, we explain our first-order methods that solve the SDPs of the new proposed algorithms. In section 2.5 we show a new approach to gain greater scalability gains. This is achieved through the use of the Burer - Monteiro (B-M) method [20] that deals with the computational bottlenecks of the method of section 2.4. Finally, in Section 2.6, we compare with several state-of-the-art algorithms and we also compare all of our proposed approaches to solve the SDPs. For the latter, we test the computational speed of each algorithm, as well as their success rate in computing the ground truth. We also examine the scalability gains achieved by using the B-M method to solve the SDPs.

2.2 Models and Algorithms from [70]

In [70], the following matrix decomposition is initially considered.

Definition 2.1 (Symmetric Sign Component Decomposition) *The matrix $\mathbf{A} \in \mathcal{S}^m$ is a correlation matrix and admits a rank-r symmetric sign component decomposition (SSCD) if it can be decomposed as follows*

$$\mathbf{A} = \sum_{i=1}^{r} \tau_i \mathbf{s}_i \mathbf{s}_i^\top, \tag{2.1}$$

where $\tau \in \Delta_+^r = \{\tau \mid \tau_i > 0 \text{ for all } i, \sum_{i=1}^r \tau_i = 1\}$, and the sign components $\mathbf{s}_i \in \{\pm 1\}^m$ for all i.

Without any additional properties regarding either $\mathbf{s}_i$ or τ_i, computing the SSCD is intractable. However, if the sign components satisfy the following property, the computation of the SSCD can be performed in polynomial time [70].

Definition 2.2 (Schur independence for sign vectors) *A collection of sign vectors, $\{\mathbf{s}_1, \ldots, \mathbf{s}_r\} \subset \{\pm 1\}^m$, is Schur independent if the set $\{\mathbf{e}\} \cup \{\mathbf{s}_i \odot \mathbf{s}_j : 1 \leq i < j \leq r\} \subset \mathbb{R}^m$ is linearly independent, where $\mathbf{e}$ is the vector of all ones, and $\odot$ is the Hadamard (elementwise) product.*

If a matrix $\mathbf{S}$ with elements in $\{\pm 1\}$ has columns that are Schur independent, we say that $\mathbf{S}$ is Schur independent.

Under Schur independence, [70] proposed an algorithm to compute an SSCD of a given correlation matrix $\mathbf{A}$ as in (2.1) using SDPs; see Alg. 1. The rationale behind their algorithm is the geometric properties of the elliptope,

$$\mathcal{E}_m = \{\mathbf{X} \in \mathcal{S}^m : \operatorname{diag}(\mathbf{X}) = \mathbf{e} \text{ and } \mathbf{X} \succeq \mathbf{0}\},$$

where $\mathcal{S}^m$ is the set of real symmetric $m \times m$ matrices. The elliptope contains all $m \times m$ correlation matrices. In fact, given an orthogonal basis column space of $\mathbf{A}$, $\mathbf{U} = \operatorname{orth}(\mathbf{A})$, the set $\{\mathbf{X} \in \mathcal{E}_m \mid \operatorname{tr}(\mathbf{U}^\top \mathbf{X}\mathbf{U}) = m\}$ is a polyhedral face of $\mathcal{E}_m$ [72]. Hence maximizing a linear functional over this set will almost surely provide an extreme point of that face, which will be a rank-one sign matrix $\mathbf{s}_i\mathbf{s}_i^\top$, which are the vertices of the elliptope $\mathcal{E}_m$ [70]. This is shown in more detail in Figure 2.1.

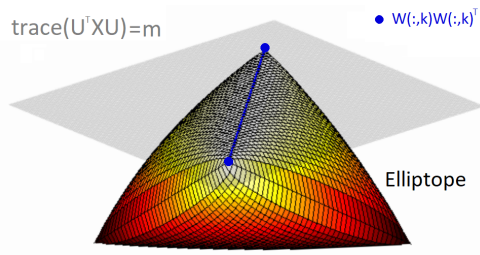

Figure 2.1: Illustration of the approach of Kueng and Tropp [70] where the convex hull of $\mathbf{W}(:,1)\mathbf{W}(:,1)^\top$ and $\mathbf{W}(:,2)\mathbf{W}(:,2)^\top$ is a polyhedral face of the elliptope. This is an example with $r = 2$, and $m = 3$. Image adapted from [70].

Once a component is identified, it can be deflated from $\mathbf{A}$, and the same trick can be applied iteratively. We next show the algorithm for SSCD from [70] and we also show the procedure of locating a vertex and then deflating the input matrix in Figure 2.2.

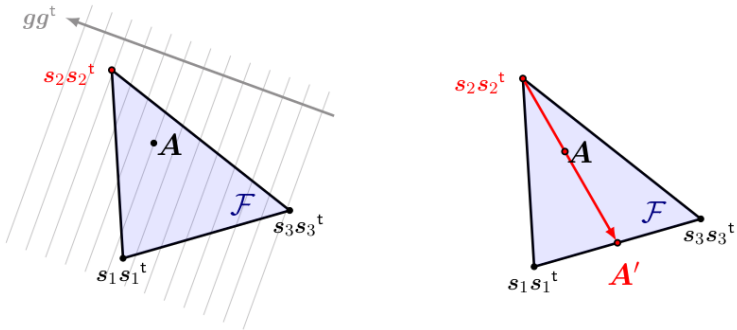

Figure 2.2: Picture showing an iteration of the algorithm for SSCD. This example is for $r = 3$. Image adapted from [70].

Algorithm 1 Noiseless SSCD [70]

Require: $\mathbf{A} = \sum_{i=1}^{r} \tau_i \mathbf{s}_i \mathbf{s}_i^\top \in \mathcal{E}_m$, where $\tau \in \Delta^r$ and $\mathbf{s}_i \in \{\pm 1\}^m$.
Ensure: Recover τ and $\{\mathbf{s}_i\}_{i=1}^r$, up to permutation.
1: **for** $k = 1, \ldots, r-1$ **do**
2: Let $\mathbf{U} = \text{orth}(\mathbf{A}) \in \mathbb{R}^{m \times r}$, then generate $\mathbf{g} \in \mathbb{R}^m$ randomly and solve

$$\max_{\mathbf{X} \in \mathcal{E}_m} \mathbf{g}^\top \mathbf{X} \mathbf{g} \quad \text{such that} \quad \text{tr}(\mathbf{U}^\top \mathbf{X} \mathbf{U}) = m, \tag{2.2}$$

to obtain $\mathbf{X}_\star = \mathbf{s}_k \mathbf{s}_k^\top$ for $\mathbf{s}_k \in \{\pm 1\}^n$
3: Calculate ζ^*, an optimal solution of

$$\max_{\zeta \in \mathbb{R}} \zeta \quad \text{such that} \quad \zeta \mathbf{A} + (1-\zeta)\mathbf{X}^* \succeq \mathbf{0}.$$

4: Update $\mathbf{A} \leftarrow \zeta^* \mathbf{A} + (1-\zeta^*)\mathbf{X}^* \succeq \mathbf{0}$ and $k \leftarrow k+1$.
5: **end for**
6: Compute $\tau = \text{argmin}_{\mathbf{y} \in \Delta^r} ||\mathbf{A} - \sum_{i=1}^{r} \mathbf{y}_i \mathbf{s}_i \mathbf{s}_i^\top||_2$.

We need to mention some additional steps that we are doing to Algorithm 1 to get the desired results and which were not described in [70] due to space limitations. In line 2 of the algorithm, when we solve problem (2.2) and factorize $\mathbf{X}_\star$. to obtain the rank-one component $\mathbf{s}_k \mathbf{s}_k^\top$, we do not immediately get a sign vector from $\mathbf{X}_\star$. From our observations, we have noticed that initially, the values of the candidate vector are in $\{\pm \frac{1}{\sqrt{m}}\}$ and not in $\{\pm 1\}$. As a result, in order to get the sign component $\mathbf{s}_k$, we need to apply the $sign(\cdot)$ operator elementwise. We now summarize all the steps to extract the rank-1 sign component in line 2 of Algorithm 1:

1. After solving the SDP in (2.2) and obtain $\mathbf{X}_\star$, we factorize it and we obtain a vector that contains elements in $\{\pm \frac{1}{\sqrt{m}}\}$,

2. Since this vector is not a sign vector yet, we apply elementwise the $sign(\cdot)$ operator to finally obtain $\mathbf{s}_k$.

The following is the second MF presented in [70].

Definition 2.3 (Symmetric Binary Component Decomposition) *The matrix $\mathbf{H} \in \mathbb{R}^{m\times m}$ is a positive semi definite (psd) matrix and admits a rank-r symmetric binary component decomposition (SBCD) if it can be decomposed as follows*

$$\mathbf{H} = \sum_{i=1}^{r} \tau_i \mathbf{z}_i \mathbf{z}_i^\top, \tag{2.3}$$

where $\tau \in \Delta_+^r = \{\tau \mid \tau_i > 0 \text{ for all } i, \sum_{i=1}^r \tau_i = 1\}$, and the binary components $\mathbf{z}_i \in \{0,1\}^m$ for all i.

If the binary components are Schur independent, then (2.3) is uniquely defined, up to trivial symmetries. The definition of Schur independence is different for binary vectors:

Definition 2.4 (Schur independence for binary vectors) *A set of binary vectors, $\{\mathbf{z}_1, \ldots, \mathbf{z}_r\} \subseteq \{0,1\}^m$, is Schur independent if the set $\{\mathbf{e}\} \cup \{\mathbf{z}_i : 1 \leq i \leq r\} \cup \{\mathbf{z}_i \odot \mathbf{z}_j : 1 \leq i < j \leq r\} \subset \{0,1\}^m$ is linearly independent.*

Just like in the sign vector definition of Schur independence, if a binary matrix $\mathbf{Z}$ has Schur independent columns, then we say that $\mathbf{Z}$ that is Schur independent. SBCD is not used in the computation of the next factorizations that we will present, and as such, we do not focus on it in this thesis.

Remark 2.1 *SBCD is discussed only in [70] and not in [71], where it is referred to as "binary component decomposition". We added the "symmetric" to the name of the decomposition to distinguish it from a model that we will present later. SSCD is also only referred to as "sign component decomposition" in [70], with the "symmetric" part of the name being added in [71].*

We now continue our discussion with the other factorization models from the paper [71]. These are two factorizations that use SSCD as a step to compute them. We present the first one, *Asymmetric Sign Component Decomposition* (ASCD):

Definition 2.5 (Asymmetric Sign Component Decomposition [71]) *The matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$ admits an asymmetric sign component decomposition (ASCD) $\mathbf{B} = \mathbf{S}\mathbf{W}^\top$ (where $\mathbf{S} \in \{\pm 1\}^{m\times r}$ and $\mathbf{W} \in \mathbb{R}^{n\times r}$) if*

- *The sign matrix $\mathbf{S} \in \{\pm 1\}^{m\times r}$ is Schur independent.*
- *The matrix $\mathbf{W} \in \mathbb{R}^{n\times r}$ has full column rank.*

Under these assumptions, the authors prove that the decomposition is unique and can be computed in polynomial time. Their algorithm for noiseless ASCD is shown in Algorithm 2. We do not need to add any additional steps that are not presented

Algorithm 2 Noiseless ASCD [71]

Require: Dimensions m and n, rank-r matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$ that satisfies the conditions of Definition 2.5, rank r.

Ensure: Sign matrix $\mathbf{S} \in \{\pm 1\}^{m \times r}$ and weight matrix $\mathbf{W} \in \mathbb{R}^{n \times r}$ with $\mathbf{B} = \mathbf{S}\mathbf{W}^\top$

1: $\mathbf{U} \leftarrow orth(\mathbf{B})$

2: Find the solution $(\mathbf{X}_\star, \mathbf{Y}_\star)$ to the semidefinite program

$$\begin{aligned} \min_{\mathbf{X} \in \mathcal{S}^m, \mathbf{Y} \in \mathcal{S}^n} \quad & \operatorname{trace}(\mathbf{Y}) \\ \text{s.t.} \quad & \operatorname{trace}(\mathbf{U}^\top \mathbf{X} \mathbf{U}) = m \text{ and } \operatorname{diag}(\mathbf{X}) = \mathbf{e} \\ & \begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0} \end{aligned} \tag{2.4}$$

3: Apply algorithm 1 to $\mathbf{X}_\star$ to obtain an SSCD

$$\mathbf{X}_\star = \mathbf{S} \operatorname{diag}(\boldsymbol{\tau}) \mathbf{S}^\top$$

4: Find $\mathbf{W}$ by solving the linear system $\mathbf{B} = \mathbf{S}\mathbf{W}^\top$

in Algorithm 2, like in the case of SSCD.

Before discussing the next factorization, let's discuss problem (2.4). The authors intially present the following lemma:

Lemma 2.1 *Let $\mathbf{B} \in \mathbb{R}^{m \times n}$. The semidefinite relation*

$$\begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0} \tag{2.5}$$

enforces a factorization of $\mathbf{B}$ in the following sense:

1. *If $\mathbf{B} = \mathbf{U}\mathbf{V}^\top$, then (2.5) holds when $\mathbf{X} = \mathbf{U}\mathbf{U}^\top$ and $\mathbf{Y} = \mathbf{V}\mathbf{V}^\top$.*
2. *If (2.5) holds, then we can decompose $\mathbf{B} = \mathbf{U}\mathbf{V}^\top$ into factors $\mathbf{U} \in \mathbb{R}^{m \times r}$ and $\mathbf{V} \in \mathbb{R}^{n \times r}$ that satisfy $\mathbf{X} = \mathbf{U}\mathbf{U}^\top$ and $\mathbf{Y} = \mathbf{V}\mathbf{V}^\top$.*

This lemma, as well as the rest of the constraints of (2.4), force $\mathbf{X}_\star$ to be a correlation matrix whose range equals the range of B. The authors further prove the following proposition.

Proposition 2.1 *Let $\mathbf{B} \in \mathbb{R}^{m \times n}$ that admits an ASCD. Let $(\mathbf{X}_\star, \mathbf{Y}_\star)$ be the unique minimizer of (2.4). Then $\mathbf{X}_\star = \mathbf{S} \operatorname{Diag}(\boldsymbol{\tau}) \mathbf{S}^\top$, where $\mathbf{S} \in \{\pm 1\}^{m \times r}$ is the sign component of $\mathbf{B}$ and $\boldsymbol{\tau} \in \Delta_r^+$.*

We can then find the sign component of $\mathbf{B}$ by applying our SSCD algorithm on $\mathbf{X}_\star$. We now discuss the final factorization that is presented in [71].

Definition 2.6 (Asymmetric Binary Component Decomposition) *The matrix* $\mathbf{C} \in \mathbb{R}^{m\times n}$ *admits an asymmetric binary component decomposition (ABCD)* $\mathbf{C} = \mathbf{Z}\mathbf{W}^\top$*, where* $\mathbf{Z} \in \{0,1\}^{m\times r}$ *and* $\mathbf{W} \in \mathbb{R}^{n\times r}$*, if*

- $\mathbf{Z} \in \{0,1\}^{n\times r}$ *is Schur independent.*
- $\mathbf{W} \in \mathbb{R}^{m\times r}$ *has full column rank.*

Remark 2.2 *ABCD described in [71] is the same as semi-bMF. For the remainder of the paper, when referring to ABCD, we are referring to the model and algorithm described in [71].*

The authors show that if the conditions in Definition 2.6 are satisfied, then the ABCD is unique and can be computed in polynomial time.

In [71], it is explained that the two Schur independence definitions can be linked as follows. First, let us define the following one-to-one affine map:

$$\mathbf{F} : \{0,1\}^m \to \{\pm 1\}^m \quad \text{where} \quad \mathbf{F} : \mathbf{z} \to 2\mathbf{z} - \mathbf{e} \quad \text{and} \quad \mathbf{F}^{-1} : \mathbf{s} \to \tfrac{1}{2}(\mathbf{s} + \mathbf{e}), \tag{2.6}$$

where $\mathbf{e}$ is the all ones vector. Based on equation (2.6) we can give this additional definition on Schur independence for binary matrices.

Lemma 2.2 *A binary matrix* $\mathbf{Z}$ *is Schur independent if and only if the sign matrix* $\begin{bmatrix}\mathbf{F}(\mathbf{Z}) & \mathbf{e}\end{bmatrix} \in \{\pm 1\}^{m\times r+1}$ *is Schur independent.*

This gives us an easier way to find if a binary matrix is Schur independent. Furthermore, the authors show in [71] that given a matrix $\mathbf{C}$ that admits an ABCD of rank r, the matrix $\mathbf{B} = 2\mathbf{C} - \mathbf{E}$ admits an ASCD of rank $r+1$. By finding this ASCD, and performing some additional operations, the ABCD of $\mathbf{C}$ can be retrieved. Algorithm 3 shows the original algorithm from [71] for ABCD.

Algorithm 3 Noiseless ABCD [71]

Require: Dimensions m and n, rank-r matrix $\mathbf{C} \in \mathbb{R}^{m \times n}$ that satisfies the conditions of Definition 2.6, rank r.

Ensure: Binary matrix $\mathbf{Z} \in \{0,1\}^{m \times r}$ and weight matrix $\mathbf{W} \in \mathbb{R}^{n \times r}$ with $\mathbf{C} = \mathbf{Z}\mathbf{W}^\top$

1: Form the matrix $\mathbf{B} = 2\mathbf{C} - \mathbf{E}$ // $\mathbf{E}$ is the all ones matrix.

2: Apply Algorithm 2 to obtain an asymmetric sign component decomposition

$$\mathbf{B} = \mathbf{S}\mathbf{W}^\top = \begin{bmatrix} \mathbf{s_1} & \dots & \mathbf{s_{r+1}} \end{bmatrix} \begin{bmatrix} \mathbf{w_1} & \dots & \mathbf{w_{r+1}} \end{bmatrix}^\top$$

3: Find the index i and sign $\phi \in \{\pm 1\}$ where $\mathbf{s}_i = \phi\mathbf{e}$. Permute the sequences to interchange

$$\mathbf{s}_i \leftrightarrow \mathbf{s}_{r+1} \quad \text{and} \quad \mathbf{w}_i \leftrightarrow \mathbf{w}_{r+1}$$

4: Find the solution $\boldsymbol{\xi} \in \mathbb{R}^r$ to the linear system

$$\begin{bmatrix} \mathbf{w}_1 & \dots & \mathbf{w}_r \end{bmatrix} \boldsymbol{\xi} = \phi\mathbf{w}_{r+1} + \mathbf{e}$$

5: Set $\mathbf{z}_i = \frac{1}{2}(\xi_i \mathbf{s}_i + \mathbf{e})$ for each index $i = 1, \dots, r$

6: Define the binary matrix $\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \dots & \mathbf{z}_r \end{bmatrix}$ and the weight matrix $\mathbf{W} = \begin{bmatrix} \xi_1\mathbf{w}_1 & \dots & \xi_r\mathbf{w}_r \end{bmatrix}$

It is worth noting, that while the algorithms that were proposed are polynomial, the memory demands are still quite high and do not allow the solution of problems with high n and m (for our machine with a 16 GB RAM, the limit is around a value of 100 for both n and m.). For this reason, we want to also develop scalable algorithms that may not solve the SDPs as well as an SDP solver but will allow us to tackle larger problems.

In the next section, we will show our new algorithms for robust ABCD. We have to modify all three algorithms that we previously presented in this section, so that they can handle noisy inputs. As we mentioned, even in the slightest presence of noise in the input matrices, all of these algorithms fail, because the SDP for SSCD has an empty feasible set.

2.3 Robust versions of the algorithms

In order to solve ABCD in the presence of noise relying on the approach of Kueng and Tropp, we need to adapt SSCD in the presence of noise.

2.3.1 Robust SSCD

Let us consider $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{N}$, where $\mathbf{A} \in \mathcal{S}^m$ admits an SSCD with Schur independent sign components, and $\mathbf{N} \in \mathbb{R}^{m \times m}$ is additive noise. Let us see how we can adapt the different steps of Alg. 1:

Step 3. The basis for the column space of $\mathbf{A}$, $\mathbf{U} \in \mathbb{R}^{m\times r}$, is unknown but can be replaced by the first r left singular vectors of $\tilde{\mathbf{A}}$, denoted $\tilde{\mathbf{U}} \in \mathbb{R}^{m\times r}$. If the noise is sufficiently small, the column spaces of $\mathbf{U}$ and $\tilde{\mathbf{U}}$ will be close to one another [107]. It can be proven that $\mathrm{tr}(\mathbf{U}^\top \mathbf{X}\mathbf{U}) \leq m$ holds for all $\mathbf{X} \in \mathcal{E}_m$, and hence the constraint $\mathrm{tr}(\mathbf{U}^\top \mathbf{X}\mathbf{U}) = m$ can be replaced with $\mathrm{tr}(\mathbf{U}^\top \mathbf{X}\mathbf{U}) \geq m$. Second, the SDP (2.2) is infeasible in general when $\mathbf{U}$ is replaced by $\tilde{\mathbf{U}}$, because the hyperplane $\{\mathbf{X} \mid \mathrm{tr}(\tilde{\mathbf{U}}^\top \mathbf{X}\tilde{\mathbf{U}}) = m\}$ intersects $\mathcal{E}_m$ only when the basis $\tilde{\mathbf{U}}$ coincide with the column space of a matrix within $\mathcal{E}_m$. This will happen with probability zero when $r < m$ and $\mathbf{N}$ is randomly generated. We therefore compute an optimal solution, $\mathbf{X}^*$, of

$$\max_{\mathbf{X}\in\mathcal{E}_m} \mathbf{g}^\top \mathbf{X}\mathbf{g} \quad \text{such that} \quad \mathrm{tr}(\tilde{\mathbf{U}}^\top \mathbf{X}\tilde{\mathbf{U}}) \geq m(1-\epsilon), \tag{2.7}$$

where ϵ is a hyperparameter. We can prove by duality that the trace constraint is active, and hence replace it with equality for simplicity.

Activity of the trace constraint. Assume that there exists $\epsilon > 0$ such that

$$\mathrm{tr}(\tilde{\mathbf{U}}^\top \mathbf{X}_0\tilde{\mathbf{U}}) < m(1-\epsilon) < \max_{\mathbf{X}\in\mathcal{E}_m} \mathrm{tr}(\tilde{\mathbf{U}}^\top \mathbf{X}\tilde{\mathbf{U}}),$$

where $\mathbf{X}_0$ is an optimal solution of the unconstrained problem $\max_{\mathbf{X}\in\mathcal{E}_m} \mathbf{g}^\top \mathbf{X}\mathbf{g}$. The second inequality ensures strict feasibility, which means Slater's condition holds, hence strong duality holds. Introducing a multiplier $\mu \geq 0$ for $\mathrm{tr}(\tilde{\mathbf{U}}^\top \mathbf{X}\tilde{\mathbf{U}}) \geq m(1-\epsilon)$, the KKT conditions give

$$\mu^\star \big(\mathrm{tr}(\tilde{\mathbf{U}}^\top \mathbf{X}^\star\tilde{\mathbf{U}}) - m(1-\epsilon)\big) = 0.$$

If the trace constraint is inactive, then $\mu^\star = 0$ and $\mathbf{X}^\star$ would also solve the unconstrained problem, hence $\mathbf{X}^\star = \mathbf{X}_0$. This contradicts $\mathrm{tr}(\tilde{\mathbf{U}}^\top \mathbf{X}_0\tilde{\mathbf{U}}) < m(1-\epsilon)$. Therefore $\mu^\star > 0$ and the trace constraint is active.

Steps 4-5. Steps 4-5 of Alg. 1 perform a deflation. This is useful when $\mathbf{A}$ belongs to a simplicial face of the elliptope $\mathcal{E}_m$. This is however not the case for $\tilde{\mathbf{A}}$, and this deflation step cannot produce a meaningful result: in fact, because of the noise and low-rankness of $\mathbf{A}$, $\tilde{\mathbf{A}}$ is typically not positive semidefinite and does not belong to the elliptope. This is interesting to notice, and we actually preprocess the input matrix, $\tilde{\mathbf{A}}$, by projecting it onto the elliptope. We observe that this preprocessing improves the performance of our algorithm. Moreover, in the noisy case, (2.7) will not produce a rank-one sign matrix. Hence after computing $\mathbf{X}^*$, we let $\mathbf{u}_1$ be the first left singular vector of $\mathbf{X}^*$, and set $\mathbf{s}_1 = \mathrm{sign}(\mathbf{u}_1)$.

Once $\mathbf{s}_1$ is identified, how can we extract more components? Of course, we could generate another $\mathbf{g}$ randomly, hoping to obtain another rank-one sign matrix. However, there is no guarantee that this will work, and could require many attempts to find all sign components. To alleviate this, we use an orthogonalization procedure for the random $\mathbf{g}$ that we generate: the next $\mathbf{g}$ is orthogonal to the computed $\mathbf{s}_k$'s so that $\mathbf{s}_k\mathbf{s}_k^\top$ cannot maximize the (nonnegative) objective, $\mathbf{g}^\top \mathbf{X}\mathbf{g} \geq 0$, since $\mathbf{g}^\top \mathbf{s}_k\mathbf{s}_k^\top \mathbf{g} = 0$. Note that this idea could also be used in Alg. 1 to replace its deflation step which is computationally (slightly) more demanding than simply generating $\mathbf{g}$'s orthogonal to the previously extracted $\mathbf{s}_k$'s. Still, in practice, because of the noise, the computed

rank-one factor $\mathbf{s}_k\mathbf{s}_k^\top$ at step k might not be satisfactory. This can be verified by checking that $\mathrm{tr}(\tilde{\mathbf{U}}^\top\mathbf{s}_k\mathbf{s}_k^\top\tilde{\mathbf{U}})$ is large enough, otherwise we start again by generating another $\mathbf{g}$. Alg. 4 summarizes our noisy variant of Alg. 1.

Algorithm 4 Noisy SSCD

Require: A matrix $\tilde{\mathbf{A}} = \sum_{i=1}^r \tau_i\mathbf{s}_i\mathbf{s}_i^\top + \mathbf{N}$, where $\tau \in \Delta^r$ and $\mathbf{s}_i \in \{\pm 1\}^m$, $\epsilon > 0$.
Ensure: Recover τ and $\{\mathbf{s}_i\}_{i=1}^r$, up to permutation, given that $\|\mathbf{N}\|_2$ is sufficiently small.

1: $\mathbf{P} = \mathbf{I}_m$; Project $\tilde{\mathbf{A}}$ to $\mathcal{E}_m$.
2: **for** $k = 1 : r$ **do**
3: Let $\tilde{\mathbf{U}}$ contain the first r left singular vectors of $\tilde{\mathbf{A}}$.
4: **repeat**
5: Generate $\mathbf{g} \in \mathbb{R}^m$ randomly, set $\mathbf{g} \leftarrow \mathbf{P}\mathbf{g}$, and solve

$$\max_{\mathbf{X}\in\mathcal{E}_m} \mathbf{g}^\top\mathbf{X}\mathbf{g} \text{ such that } \mathrm{tr}(\tilde{\mathbf{U}}^\top\mathbf{X}\tilde{\mathbf{U}}) = m(1-\epsilon),$$

to obtain $\mathbf{X}^*$.
6: Compute $\mathbf{u_1}$, the first left singular vector of $\mathbf{X}^*$, and set $\mathbf{s}_k = \mathrm{sign}(\mathbf{u_1})$.
7: **until** a suitable sign vector $\mathbf{s}_k$ is computed
8: $\mathbf{P} = (\mathbf{I}_m - \frac{(\mathbf{P}\tilde{\mathbf{s}}_k)(\mathbf{P}\tilde{\mathbf{s}}_k)^\top}{(\|\mathbf{P}\tilde{\mathbf{s}}_k)\|_2^2})\mathbf{P}$
9: **end for**
10: Compute $\tau = \mathrm{argmin}_{\mathbf{y}\in\Delta^r} ||\tilde{\mathbf{A}} - \sum_{i=1}^r \mathbf{y}_i\mathbf{s}_i\mathbf{s}_i^\top\|_2$.

After retrieving an $\mathbf{X}$ from step 5 of Algorithm 2, we perform step 6 to retrieve $\mathbf{s}_k$. The fact that $\mathbf{X}$ satisfies the trace equality constraint of the feasible set does not imply that the retrieved $\mathbf{s}_k\mathbf{s}_k^T$ will also satisfy it. We call $\mathbf{s}_k$ a suitable vector if $\mathbf{s}_k\mathbf{s}_k^\top$ satisfies the trace equality constraint $\mathrm{tr}(\tilde{\mathbf{U}}^\top\mathbf{X}\tilde{\mathbf{U}}) = m(1-\epsilon)$. In addition, the $\tilde{\mathbf{A}}$ used in line 10 is the one retrieved after the projection onto the elliptope.
Why does Alg. 4 work? Let us provide a recovery guarantee for Alg. 4.

Theorem 2.1 *Let* $\mathbf{A} = \sum_{i=1}^r \tau_i\mathbf{s}_i\mathbf{s}_i^\top \in \mathcal{S}^m$, $\mathbf{N} \in \mathbb{R}^{m\times m}$ *be the noise such that* $\|\mathbf{N}\|_2 \leq \delta$, *and* $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{N}$. *For* δ *sufficiently small, there exists some* $\epsilon \geq 0$ *such that Alg. 4 recovers* $\{\mathbf{s}_i\}_{i=1}^r$ *exactly, with probability one.*

Proof 2.1 (Sketch of the proof) *First, let us denote* $\tilde{\mathbf{U}}(\delta)$ *the column space of the first* r *singular vectors of* $\tilde{\mathbf{A}}$. *This subspace changes smoothly with* δ, *and coincide with that of* $\mathbf{U}$ *for* $\delta = 0$ *[107]. In the noiseless SDP* (2.2), *we can prove that if the* $\mathbf{s}_i$*'s generating* $\mathbf{A}$ *are Schur independent, then* $\mathrm{tr}(\mathbf{U}^\top\mathbf{s}\mathbf{s}^\top\mathbf{U}) \leq m-1$ *for any* $\mathbf{s} \in \{\pm 1\}^m$ *with* $\mathbf{s} \neq \mathbf{s}_i$ *for all* i. *The function* $f(\mathbf{X}) = \mathrm{tr}(\tilde{\mathbf{U}}(\delta)^\top\mathbf{X}\tilde{\mathbf{U}}(\delta))$ *changes continuously with* δ, *while we know* $f(\mathbf{X}) \leq m$ *for any* $\mathbf{X} \in \mathcal{E}_m$. *Therefore, for a well chosen* ϵ, *given that* δ *is sufficiently small, the only rank-one sign matrices feasible for* (2.7) *will be the rank-one components of* $\mathbf{A}$.

Recall that, with probability one, the optimal solution of (2.7) *is unique (it identifies a vertex of the face of* $\mathcal{E}_m$ *that contains A in its interior). The optimal solution*

of (2.7) *changes smoothly with* δ *and* ϵ *[14], and coincides with* (2.2) *when* $\delta = \epsilon = 0$. *Hence for* δ *sufficiently small, there exists* ϵ *such that an optimal solution of* (2.7), $\mathbf{X}^*$, *will be close to that of* (2.7) *(a rank-one sign matrix of* $\mathbf{A}$*). After the post-processing of* $\mathbf{X}^*$ *(step 6 of Alg. 4), we recover a ground truth factor* $\mathbf{s}_i$.

Of course, in practice, the noise level, δ, might not be small enough. However, as we will see, Alg. 4 is able to recover accurately the sign vectors that generated $\mathbf{A}$ even for relatively large noise levels.

What is a suitable sign vector $\mathbf{s}_k$? In noisy experiments we have noticed that even though a retrieved rank-one matrix $\mathbf{X}^*$ satisfies the trace constraint $\text{tr}(\tilde{\mathbf{U}}^\top \mathbf{X} \tilde{\mathbf{U}}) = m(1-\epsilon)$, the post-processed sign vector we retrieve from it may not. A suitable sign vector still satisfies it after post-processing. For higher noise levels, there is a higher probability of obtaining a $\mathbf{X}_\star$ that does not give us a sign vector that is part of the ground truth. In that case, there is a probability of failure by the algorithm to retrieve the groundtruth.

2.3.2 Robust ASCD and robust ABCD

The robust versions of ASCD and ABCD do not need considerable changes (like it was the case with SSCD), in comparison to the original versions of the algorithms. We will mention the changes that we make to Algorithms 2 and 3 to adapt them for the noisy scenario.

For ASCD:

- In problem (2.4), the constraint $\text{trace}(\mathbf{U}^\top \mathbf{X} \mathbf{U}) = m$ is replaced by the constraint $\text{trace}(\mathbf{U}^\top \mathbf{X} \mathbf{U}) = m(1-\epsilon)$ where ϵ is a hyperparameter and not necessarily equal to the ϵ of robust SSCD.
- In line 3, we obtain $\mathbf{X}_\star$ through Algorithm 4 instead of Algorithm 1.

For ABCD:

- In line 2, we use robust ASCD instead of Algorithm 2.
- In line 5, when we apply elementwise the operation $\mathbf{z}_i = \frac{1}{2}(\xi_i \mathbf{s}_i + \mathbf{e})$, we do not immediately get a vector with elements in $\{0, 1\}$, but we get elements that are close to 0 and 1. As such, we have to apply an elementwise rounding operator to get $\mathbf{Z}$.

We now show the algorithms for robust ASCD and robust ABCD.

Algorithm 5 Robust ASCD

Require: Dimensions m and n, rank-r matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$ that satisfies the conditions of Definition 2.5, rank r, tolerance hyperparameter ϵ.

Ensure: Sign matrix $\mathbf{S} \in \{\pm 1\}^{m \times r}$ and weight matrix $\mathbf{W} \in \mathbb{R}^{n \times r}$ with $\mathbf{B} = \mathbf{S}\mathbf{W}^\top$

1: $\mathbf{U} \leftarrow orth(\mathbf{B})$

2: Find the solution $(\mathbf{X}_\star, \mathbf{Y}_\star)$ to the semidefinite program

$$\begin{aligned} \min_{\mathbf{X} \in \mathcal{S}^m, \mathbf{Y} \in \mathcal{S}^n} \quad & \operatorname{trace}(\mathbf{Y}) \\ \text{s.t.} \quad & \operatorname{trace}(\mathbf{U}^\top \mathbf{X} \mathbf{U}) = m(1-\epsilon) \text{ and } \operatorname{diag}(\mathbf{X}) = \mathbf{e} \\ & \begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0} \end{aligned} \tag{2.8}$$

3: Apply Algorithm 4 to $\mathbf{X}_\star$ to obtain an SSCD

$$\mathbf{X}_\star = \mathbf{S} \operatorname{diag}(\boldsymbol{\tau}) \mathbf{S}^\top$$

4: Find $\mathbf{W}$ by solving the linear system $\mathbf{B} = \mathbf{S}\mathbf{W}^\top$

Algorithm 6 Robust ABCD

Require: Dimensions m and n, rank-r matrix $\mathbf{C} \in \mathbb{R}^{m \times n}$ that satisfies the conditions of Definition 2.6, rank r.

Ensure: Binary matrix $\mathbf{Z} \in \{0, 1\}^{m \times r}$ and weight matrix $\mathbf{W} \in \mathbb{R}^{n \times r}$ with $\mathbf{C} = \mathbf{Z}\mathbf{W}^\top$

1: Form the matrix $\mathbf{B} = 2\mathbf{C} - \mathbf{E}$ // $\mathbf{E}$ is the all ones matrix.

2: Apply Algorithm 5 to obtain an asymmetric sign component decomposition

$$\mathbf{B} = \mathbf{S}\mathbf{W}^\top = \begin{bmatrix} \mathbf{s_1} & \dots & \mathbf{s_{r+1}} \end{bmatrix} \begin{bmatrix} \mathbf{w_1} & \dots & \mathbf{w_{r+1}} \end{bmatrix}^\top$$

3: Find the index i and sign $\phi \in \{\pm 1\}$ where $\mathbf{s}_i = \phi \mathbf{e}$. Permute the columns so that $\phi \mathbf{e}$ is the last column

$$\mathbf{s}_i \leftrightarrow \mathbf{s}_{r+1} \quad \text{and} \quad \mathbf{w}_i \leftrightarrow \mathbf{w}_{r+1}$$

4: Find the solution $\boldsymbol{\xi} \in \mathbb{R}^r$ to the linear system

$$\begin{bmatrix} \mathbf{w}_1 & \dots & \mathbf{w}_r \end{bmatrix} \boldsymbol{\xi} = \phi \mathbf{w}_{r+1} + \mathbf{e}$$

5: Set $\mathbf{z}_i = \frac{1}{2}(\xi_i \mathbf{s}_i + \mathbf{e})$ for each index $i = 1, \dots, r$ and round each element to the closest integer.

6: Define the binary matrix $\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \dots & \mathbf{z}_r \end{bmatrix}$ and the weight matrix $\mathbf{W} = \begin{bmatrix} \xi_1 \mathbf{w}_1 & \dots & \xi_r \mathbf{w}_r \end{bmatrix}$

2.4 Gradient methods for the SDP problems

2.4.1 First-order method for SSCD

One way to solve (2.7) with high accuracy is through an interior point method [91]. For an $m \times m$ input correlation matrix, this requires $O(m^6)$ operations per iteration, which makes it impractical for large data sets. To reduce the computational cost, we propose a first-order method to solve (2.7); see Alg. 8. Note, and this is crucial, that we do not need a high accuracy solution of (2.7), since we are going to postprocess the solution to obtain a rank-one sign matrix.

For the projection step onto the feasible set, $\mathbb{E}_m \cap \mathbb{T}_\epsilon$, where $\mathbb{T}_\epsilon = \{\mathbf{X} \mid \operatorname{tr}(\tilde{\mathbf{U}}^\top \mathbf{X} \tilde{\mathbf{U}}) = m(1-\epsilon)\}$, we use an alternating projection strategy. We choose to first project onto the set of correlation matrices, and then onto the set $\mathbb{T}_\epsilon$. The projection onto $\mathbb{T}_\epsilon$ is simple, and given by $P_{\mathbb{T}_\epsilon}(\mathbf{Y}) = \mathbf{Y} - \frac{\operatorname{tr}(\mathbf{U}\mathbf{U}^\top \mathbf{Y}) - m(1-\epsilon)}{r} \mathbf{U}\mathbf{U}^\top$. A method for the projection onto the elliptope, inspired by Dykstra's algorithm for projecting into an intersection of convex sets [17], is described in [58]. This algorithm is not to be confused with Dijkstra's algorithm for finding the shortest paths between nodes in a weighted graph. The author in [58] states that as $k \to \infty$, then both $\mathbf{X}_k$ and $\mathbf{Y}_k$

Algorithm 7 Projection to the nearest correlation matrix [58]

Require: A symmetric matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, max_iter, maximum number of iterations of the for loop.
Ensure: A correlation matrix $\mathbf{D}_\star \in \mathbb{R}^{n \times n}$.

1: Set $\mathbf{Y}_0 = \mathbf{D}$, $\mathbf{\Delta S}_0 = \mathbf{0}_{n \times n}$
2: **for** $k = 1, 2, \ldots max\_iter$ **do**
3: $\quad \mathbf{R}_k = \mathbf{Y}_{k-1} - \mathbf{\Delta S}_{k-1}$
4: $\quad \mathbf{X}_k = P_{(\cdot \succeq \mathbf{0})}(\mathbf{R}_k)$
5: $\quad \mathbf{\Delta S}_k = \mathbf{X}_k - \mathbf{R}_k$
6: $\quad \mathbf{Y}_k = P_{(\operatorname{Diag}(\cdot) = \mathbf{e})}(\mathbf{X}_k)$.
7: **end for**
8: Set $\mathbf{D}_\star$ as either $\mathbf{X}_k$ or $\mathbf{Y}_k$.

converge to the desired correlation matrix. In our implementation, we choose our output to be $\mathbf{Y}_k$. We perform 10 iterations of this alternating strategy (onto the set $\mathbb{E}_m \cap \mathbb{T}_\epsilon$), while the number of iterations for the algorithm of [58] is set to 15.

The projections on lines 4 and 6 of Algorithm 7 are not straightforward, and we will now give their formulas. The projection of a symmetric matrix to the set of matrices where the diagonal is full of ones $P_{(\operatorname{Diag}(\cdot)=\mathbf{e})}(\mathbf{A})$ is given by [58]

$$P_{(\operatorname{Diag}(\cdot)=\mathbf{e})}(\mathbf{A}) = \mathbf{A} - \operatorname{Diag}(\theta_i), \tag{2.9}$$

where $\theta = \begin{bmatrix}\theta_1, \ldots, \theta_n\end{bmatrix}^\top$ is the solution of

$$\theta = \operatorname{Diag}(\mathbf{A} - \mathbf{I}). \tag{2.10}$$

The projection onto the set of positive semi-definite matrices $P_{(\cdot \succeq \mathbf{0})}(\mathbf{A})$ is more complex. Let us consider the spectral decomposition $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, where $\mathbf{D} = \mathrm{Diag}(\lambda_i)$ and $\mathbf{Q}$ is orthogonal and let

$$\mathbf{A}_+ = \mathbf{Q}\,\mathrm{Diag}(\max(\lambda_i, 0))\mathbf{Q}^\top. \tag{2.11}$$

$P_{(\cdot \succeq \mathbf{0})}(\mathbf{A})$ is then given by

$$P_{(\cdot \succeq \mathbf{0})}(\mathbf{A}) = \mathbf{I}_m^{-1/2}(\mathbf{I}_m^{1/2}\mathbf{A}_+\mathbf{I}_m^{1/2})\mathbf{I}_m^{1/2}, \tag{2.12}$$

where $\mathbf{I}_m$ is the identity matrix of size $m \times m$.

We resume the discussion on the projected gradient (PG) method used for SSCD. The cost function $f(\mathbf{X}) = \mathbf{g}^\top\mathbf{X}\mathbf{g}$ is a linear function of $\mathbf{X}$ and its gradient is $\nabla f(\mathbf{X}) = \mathbf{g}\mathbf{g}^\top$. In our implementation, we use a naive step-size, namely $1/\|\mathbf{g}\|_2^2$, which appears to perform well in practice. Surprisingly, we observe that performing only 5 iterations of the projected gradient is enough to obtain competitive results on synthetic data sets.

Algorithm 8 Projected gradient method for the SSCD to solve (2.7)

Require: Initial matrix $\mathbf{X} \in \mathbb{R}^{m\times m}$, matrix $\mathbf{U} \in \mathbb{R}^{m\times r}$, a vector $\mathbf{g} \in \mathbb{R}^m$, a scalar $\epsilon > 0$.

Ensure: Solves (2.7), approximately.

$k = 1$, $\mathbf{X}_k = \mathbf{X}$, $L = \sigma_1(\mathbf{g}\mathbf{g}^\top) = \|\mathbf{g}\|_2^2$
while maximum gradient iterations have not been reached **do**
  $\mathbf{X}_{k+1} = \mathbf{X}_k - \frac{1}{L}(-\mathbf{g}\mathbf{g}^\top)$
  while maximum projecting iterations have not been reached **do**
    Project $\mathbf{X}_{k+1}$ to $\mathcal{E}_m$ according to [58].
    Set $\mathbf{X}_{k+1} = \mathbf{X}_{k+1} - \frac{\mathrm{tr}(\mathbf{U}\mathbf{U}^\top\mathbf{X}_{k+1}) - m(1-\epsilon)}{r}\mathbf{U}\mathbf{U}^\top$
  end while
  $k = k + 1$.
end while

2.4.2 First order method for ASCD

We want to use a projected gradient method to solve (2.8) as well, so we are able to tackle large-scale problems. In order to make the projection step easier, we note that the constraint

$$\begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0}$$

also enforces $\mathbf{X} \succeq \mathbf{0}$. As a result, we restate problem 2.8 as

$$\begin{aligned} \min_{\mathbf{X}\in\mathcal{S}^m, \mathbf{Y}\in\mathcal{S}^n} \quad & \text{trace}(\mathbf{Y}) \\ \text{s.t.} \quad & \text{trace}(\mathbf{U}^\top \mathbf{X}\mathbf{U}) = m(1-\epsilon) \\ & \text{diag}(\mathbf{X}) = \mathbf{e} \\ & \mathbf{X} \succeq \mathbf{0} \\ & \begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0}. \end{aligned} \tag{2.13}$$

Now we can use the projection step from Section 2.4.1, followed by a projection of $\begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix}$ onto the set of positive semidefinite matrices. We finally apply SSCD on the retrieved $\mathbf{X}$ from the block matrix to get the sign components.
The gradient of $\text{tr}(\mathbf{Y})$ is the identity matrix and as was the case in the previous section, it is a linear function that is 0-Lipschitz. A naive decision for the step–size would be to set it to a fixed size. The projected gradient algorithm for Robust ASCD is shown in Algorithm 9.

Algorithm 9 Projected gradient method for the ASCD to solve (2.8)

Require: Initial matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$, matrix $\mathbf{U} \in \mathbb{R}^{m\times r}$, a scalar $\epsilon > 0$, fixed gradient step size $\eta > 0$.
Ensure: Solves (2.8), approximately.

1: $k = 1$
2: Initialize $\mathbf{X}_k$ and $\mathbf{Y}_k$ as random symmetric matrices.
3: **while** maximum gradient iterations have not been reached **do**
4: $\quad \mathbf{Y}_{k+1} = \mathbf{Y}_k - \eta(\mathbf{I}_n)$
5: $\quad$ **while** maximum projecting iterations have not been reached **do**
6: $\quad\quad$ Project $\mathbf{X}_{k+1}$ to $\mathcal{E}_m$ according to [58].
7: $\quad\quad$ Set $\mathbf{X}_{k+1} = \mathbf{X}_{k+1} - \frac{\text{tr}(\mathbf{U}\mathbf{U}^\top \mathbf{X}_{k+1}) - m(1-\epsilon)}{r}\mathbf{U}\mathbf{U}^\top$
8: $\quad$ **end while**
9: $\quad$ Project the matrix $\begin{bmatrix} \mathbf{X}_{k+1} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y}_{k+1} \end{bmatrix}$ onto the set of psd matrices.
10: $\quad$ Extract the matrices $\mathbf{X}_{k+1}$ and $\mathbf{Y}_{k+1}$.
11: $\quad k = k + 1$.
12: **end while**

If we wish to solve problems where n and m are big, then we have to resort to solving (2.13) with a projected gradient method, as the memory requirements for an interior point method (through cvx for example) make this problem very difficult to solve. For a laptop with 16GB RAM memory, the memory requirements for a problem where $n = 220$ and $m = 250$ where too high, rendering it unable to solve the problem.

2.5 Burer-Monteiro (B-M) approach

2.5.1 B-M on the SSCD problem

So far, we have tried a first order method to solve (2.7). This would give us an initial approach to handle larger instances of this problem [62]. However, the projection to the set of positive semidefinite (PSD) matrices requires the computation of the eigenvalue decomposition of $\mathbf{X}$, which has a cost of $O(m^3)$. As part of a projected gradient algorithm, projection onto the feasible set of (2.7) will be performed multiple times. This renders it a computational bottleneck; in order to have an algorithm that can handle larger instances, we need to find a way to address this issue.

Let $p \in \{1, \dots, m\}$ and $\mathbf{Y} \in \mathbb{R}^{m \times p}$. If we consider $\mathbf{X} = \mathbf{Y}\mathbf{Y}^\top$, then problem (2.7) becomes [20] [15]

$$\begin{aligned} \max_{\mathbf{Y} \in \mathbb{R}^{m \times p}} \quad & \mathbf{g}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{g} \\ \text{s.t.} \quad & \operatorname{tr}(\tilde{\mathbf{U}}\mathbf{Y}\mathbf{Y}^\top\tilde{\mathbf{U}}) = m(1-\epsilon), \\ & \operatorname{Diag}(\mathbf{Y}\mathbf{Y}^\top) = \mathbf{e}. \end{aligned} \tag{2.14}$$

The matrix $\mathbf{Y}\mathbf{Y}^\top$ is always PSD, so we can drop the corresponding constraint. The other two constraints after some algebraic operations are equivalently written as

- $\operatorname{tr}(\tilde{\mathbf{U}}\mathbf{Y}\mathbf{Y}^\top\tilde{\mathbf{U}}) = m(1-\epsilon) \Leftrightarrow \|\tilde{\mathbf{U}}^\top\mathbf{Y}\|_F^2 = m(1-\epsilon)$
- $\operatorname{Diag}(\mathbf{Y}\mathbf{Y}^\top) = \mathbf{e} \Leftrightarrow \|\mathbf{y}_i\|_2^2 = 1$, where $\mathbf{y}_i$ is the i–th row of $\mathbf{Y}$, with $i \in \{1, \dots, m\}$.

As such, the problem is recast as

$$\begin{aligned} \min_{\mathbf{Y} \in \mathbb{R}^{m \times p}} \quad & f(\mathbf{Y}) = -\mathbf{g}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{g} \\ \text{s.t.} \quad & \|\tilde{\mathbf{U}}^\top\mathbf{Y}\|_F^2 = m(1-\epsilon), \\ & \|\mathbf{y}_i\|_2^2 = 1, \text{ for all } i \in \{1, \dots, m\} \end{aligned} \tag{2.15}$$

Projection onto the latter constraint set involves just dividing each row of $\mathbf{Y}$ by its norm. To the best of our knowledge, the projection onto $\|\tilde{\mathbf{U}}^\top\mathbf{Y}\|_F^2 = m(1-\epsilon)$ does not have a closed form solution. We thus propose solving the following problem instead via a penalty method.

$$\begin{aligned} \min_{\mathbf{Y} \in \mathbb{R}^{m \times p}} \quad & f(\mathbf{Y}) = -\mathbf{g}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{g} + \lambda\big(\|\tilde{\mathbf{U}}^\top\mathbf{Y}\|_F^2 - m(1-\epsilon)\big)^2 \\ \text{s.t.} \quad & \|\mathbf{y}_i\|_2^2 = 1, \text{ for all } i \in \{1, \dots, m\}. \end{aligned} \tag{2.16}$$

This would alleviate the bottleneck of the eigenvalue decomposition. For this new cost function, the gradient is given by

$$\nabla f(\mathbf{Y}) = -2\mathbf{g}\mathbf{g}^\top\mathbf{Y} + 4\lambda(\|\tilde{\mathbf{U}}^\top\mathbf{Y}\|_F^2 - m(1-\epsilon)))\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top\mathbf{Y}. \tag{2.17}$$

The most expensive operation is now the multiplication $\mathbf{g}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{g}$ with a cost of $O(m^2 p)$, down from $O(m^3)$ (due to the eigenvalue decompositions in Algorithm 7).

We now present the SSCD PG algorithm that uses the Burer-Monteiro scheme in more detail in Algorithm 10. We note that Algorithm 10 introduces additional parameters compared to the PG algorithm for SSCD that need to be tuned. However, as we will see in section 2.6.2, this new algorithm is much faster and can also tackle larger datasets.

Algorithm 10 Projected gradient method for SSCD with the Burer-Monteiro scheme and extraction of a sign component.

Require: Parameter p to form $\mathbf{Y} \in \mathbb{R}^{m\times p}$, matrix $\mathbf{U} \in \mathbb{R}^{m\times r}$, a vector $\mathbf{g} \in \mathbb{R}^m$, a scalar $\epsilon > 0$, *outer_lim* upper limit of penalty method iterations (outer iterations), parameter $\lambda > 0$ for the penalty method, parameter $c > 1$ that will be used to increase λ after every outer iteration.

Ensure: Solves (2.16), approximately and returns a sign component $\mathbf{s}_k$.

1: **for** $l = 1, \ldots,$ *outer_lim* **do**
2: Form $\mathbf{Y} \in \mathbb{R}^{m\times p}$ with each element being drawn from $\mathcal{U}[0, 1]$.
3: $k = 1$, $\mathbf{Y}_k = \mathbf{Y}$
4: **while** maximum gradient iterations have not been reached **do**
5: Compute the gradient via (2.17).
6: Compute step η via Armijo backtracking line search.
7: $\mathbf{Y}_{k+1} = \mathbf{Y}_k - \eta \nabla f(\mathbf{Y})$
8: Divide each row of $\mathbf{Y}$ by its norm to fulfill the constraint in (2.14).
9: $k = k + 1$.
10: **end while**
11: $\lambda = c\lambda$
12: $\mathbf{X} = \mathbf{Y}\mathbf{Y}^\top$
13: Compute $\mathbf{u_1}$, the first left singular vector of $\mathbf{X}$, and set $\mathbf{s} = \text{sign}(\mathbf{u_1})$.
14: If $\mathbf{s}$ is a desired sign component, return $\mathbf{s}$ and continue Algorithm 4 from line 8.
15: **end for**

2.5.2 B-M on the ASCD problem

We remind that for the computation of an Asymmetric Sign Component Decomposition (ASCD) [71], we recast the SDP problem as

$$\begin{aligned} \min_{\mathbf{X}\in\mathcal{S}^m, \mathbf{Y}\in\mathcal{S}^n} \quad & \text{trace}(\mathbf{Y}) \\ \text{s.t.} \quad & \text{trace}(\mathbf{U}^\top \mathbf{X}\mathbf{U}) = m(1-\epsilon), \\ & \text{diag}(\mathbf{X}) = \mathbf{e}, \\ & \mathbf{X} \succeq \mathbf{0}, \\ & \begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0}. \end{aligned} \tag{2.18}$$

As we already mentioned, we have to perform multiple eigenvalue decompositions, which, for growing n, raise considerably the time required to solve the SDP. In fact, if we solve the SSCD using the Burer-Monteiro approach, then solving (2.18) becomes the computational bottleneck and it needs a considerable amount of time to be solved. A straightforward approach to circumvent this problem would be to solve it with the Burer-Monteiro approach as well. If we consider $\mathbf{X} = \mathbf{Z}\mathbf{Z}^\top$, where $\mathbf{Z} \in \mathbb{R}^{m\times p}$ (where $p \in \mathbb{N}$ and is a parameter that we arbitrarily set), (2.18) is transformed into

$$\begin{aligned} \min_{\mathbf{Z}\in\mathbb{R}^{m\times p},\mathbf{Y}\in\mathcal{S}^n} \quad & \operatorname{trace}(\mathbf{Y}) \\ \text{s.t.} \quad & \|\mathbf{U}^\top\mathbf{Z}\|_F^2 = m(1-\epsilon), \\ & \|\mathbf{z}_i\|_2 = 1, \text{ for all } i \in \{1,\dots,m\}, \\ & \begin{bmatrix} \mathbf{Z}\mathbf{Z}^\top & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0}. \end{aligned} \tag{2.19}$$

As we have already mentioned, the projection of $\mathbf{Z}$ onto the set $\{\mathbf{Z} \in \mathbf{R}^{m\times p} : \|\mathbf{U}^\top\mathbf{Z}\|_F^2 = m(1-\epsilon)\}$ does not have a closed form solution. By introducing the constraint as a penalty term, we get the following new problem

$$\begin{aligned} \min_{\mathbf{Z}\in\mathbb{R}^{m\times p},\mathbf{Y}\in\mathcal{S}^n} \quad & f(\mathbf{Z},\mathbf{Y}) = \operatorname{trace}(\mathbf{Y}) + \lambda(\|\mathbf{U}^\top\mathbf{Z}\|_F^2 - m(1-\epsilon))^2 \\ \text{s.t.} \quad & \|\mathbf{z}_i\|_2 = 1, \text{ for all } i \in \{1,\dots,m\}, \\ & \begin{bmatrix} \mathbf{Z}\mathbf{Z}^\top & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0}. \end{aligned} \tag{2.20}$$

We use alternating optimization (AO) to solve this problem. The gradient of the cost function with respect to $\mathbf{Z}$ is

$$\nabla_{\mathbf{Z}} f(\mathbf{Z},\mathbf{Y}) = (4\lambda(\|\tilde{\mathbf{U}}^\top\mathbf{Z}\|_F^2 - m(1-\epsilon)))\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top\mathbf{Z}, \tag{2.21}$$

while the gradient with respect to $\mathbf{Y}$ is

$$\nabla_{\mathbf{Y}} f(\mathbf{Z},\mathbf{Y}) = \mathbf{I}_n. \tag{2.22}$$

Our B-M algorithm for ASCD is presented in Algorithm 11.

For this algorithm, we see that computing an eigenvalue decomposition is reintroduced, increasing the total cost of the algorithm to $O((n+m)^3)$. However, we observed that the number of EVDs that are computed are considerably lower than our other PG algorithms that do not use B-M.

2.6 Numerical Experiments

All experiments in this section are run on MATLAB R2018a on a laptop with AMD Ryzen 7 5800H @ 3.2 GHz and 16GB RAM [1].

[1]The code is available at https://gitlab.com/ckolomvakis/robust-binary-component-decompositions

Algorithm 11 Projected gradient method for ASCD with the Burer-Monteiro scheme.

Require: Input matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$, parameter p to form $\mathbf{Z} \in \mathbb{R}^{m \times p}$, matrix $\mathbf{U} \in \mathbb{R}^{m \times r}$, a scalar $\epsilon > 0$, *outer_lim* upper limit of penalty method iterations (outer iterations), parameter $\lambda > 0$ for the penalty method, parameter $c > 1$ that will be used to increase λ after every outer iteration.

Ensure: Solves (2.20) approximately.

Form an initial symmetric $\mathbf{Y}_0$ initially drawing elements from a standard normal distribution and form an initial $\mathbf{Z}_0 \in \mathbb{R}^{m \times p}$, with each element being drawn from $\mathcal{U}[0, 1]$.
$k = 1$, $\mathbf{Z}_k = \mathbf{Z}_0$
$j = 1$, $\mathbf{Y}_j = \mathbf{Y}_0$
while maximum alternating optimization iterations have not been reached **do**
  for $l = 1, \ldots,$ *outer_lim* **do**
    $k = 1$
    while maximum gradient iterations have not been reached **do**
      Compute the gradient via (2.21).
      Compute step η_1 via Armijo backtracking line search.
      $\mathbf{Z}_{k+1} = \mathbf{Z}_k - \eta_1 \nabla_{\mathbf{Z}} f(\mathbf{Z}_k, \mathbf{Y}_j)$
      Divide each row of $\mathbf{Z}_{k+1}$ by its norm to fulfill the constraint in (2.20).
      $k \leftarrow k + 1$.
    end while
    $\lambda = c\lambda$
  end for
  Project the matrix $\begin{bmatrix} \mathbf{Z}_k \mathbf{Z}_k^\top & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y}_j \end{bmatrix}$ onto the set of psd matrices.
  Extract the matrices $\mathbf{Z}_k \mathbf{Z}_k^\top$ and $\mathbf{Y}_j$ and then extract $\mathbf{Z}_k$.
  $j \leftarrow 1$
  while maximum gradient iterations have not been reached **do**
    Compute the gradient via (2.22).
    Compute step η_2 via Armijo backtracking line search.
    $\mathbf{Y}_{j+1} = \mathbf{Y}_j - \eta_2 \nabla_{\mathbf{Y}} f(\mathbf{Z}_k, \mathbf{Y}_j)$
    $j = j + 1$.
  end while
  Project the matrix $\begin{bmatrix} \mathbf{Z}_k \mathbf{Z}_k^\top & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y}_j \end{bmatrix}$ onto the set of psd matrices.
  Extract the matrices $\mathbf{Z}_k \mathbf{Z}_k^\top$ and $\mathbf{Y}_j$ and then extract $\mathbf{Z}_k$..
end while
Perform any SSCD algorithm on $\mathbf{Z}_k \mathbf{Z}_k^\top$ to extract the sign components of $\mathbf{B}$.

2.6.1 Original experiments from [62]

As far as we know, there do not exist heuristic algorithms for SSCD itself, only the algorithm of [70] in the noiseless case. Hence, as far as we know, *Alg. 2.2 is the first*

to handle the noisy SSCD problem. On the contrary, Semi-bMF is rather popular, and several heuristics have been introduced recently. We will compare our approach to a tensor-based approach [104] and the matrix decomposition-based approaches in [105].

2.6.1.1 Synthetic experiments under various levels of noise

For our first experiment, we consider $r = 4$, $m = 45$, $n = 65$. The input matrix $\mathbf{C} \in \mathbb{R}^{m\times n}$ is generated via a binary factor $\mathbf{Z} \in \{0,1\}^{m\times r}$, whose elements have a probability of $1/2$ to be either 0 or 1, and a random matrix $\mathbf{W} \in \mathbb{R}^{r\times n}$ with standard normal elements. Random Gaussian noise $\mathbf{N} \in \mathbb{R}^{m\times n}$ is then added, as well as an additional term $(1/15)(\mathbf{c}\mathbf{d}^\top)$ (as it was done in [105]), where $\mathbf{c} \in \mathbb{R}^m$ and $\mathbf{d} \in \mathbb{R}^n$ are uniformly distributed in $[0,1]$. We define the Signal to Noise Ratio (SNR) as $SNR = 10\log_{10}\left(\|\mathbf{ZW}\|_F^2/(\sigma_N^2\|\mathbf{N}\|_F^2)\right)$, with σ_N^2 varying according to the SNR value considered. The number of trials for each SNR value is set to 20.

For ABCD, we solve the SDPs through CVX [49] (denoted as 'ABCD algorithm using IPM'), and through our first-order approach (denoted as 'ABCD algorithm using PG', where PG stands for 'Projected Gradient'). We set $\epsilon = 0.05$. The approaches presented in [105] are a coupled matrix factorization (CMF) method (denoted as 'CMF alg.') and an Alternating Least Squares (ALS) method for Semi-bMF (denoted as 'ALS'). The tensor-based approach from [104] is referred to as 'Algebraic'.

Our comparing metric, noted *Hamming distance* here, is calculated as follows:

- We convert the retrieved binary factor into a factor in $\{\pm 1\}^{m\times r}$.
- Since the columns of the retrieved factor are only identifiable up to a permutation, we align them with the groundtruth by solving a minimum-cost assignment problem: we build the cost matrix $\mathbf{M} \in \mathbb{R}^{r\times r}$ with entries

$$M_{ij} \;=\; \min\big(d(\mathbf{s}_i, \mathbf{s}_j^\star),\, m - d(\mathbf{s}_i, \mathbf{s}_j^\star)\big),$$

  where $\mathbf{s}_i$ denotes the i-th retrieved column and $\mathbf{s}_j^\star$ the j-th groundtruth column. We then compute the optimal permutation π that minimizes $\sum_{i=1}^r M_{i,\pi(i)}$ (e.g., using the Munkres/Hungarian algorithm).
- The column-wise discrepancy is measured with

$$d(\mathbf{s}_i, \mathbf{s}_j^\star) = \|(\mathbf{e} - \mathbf{s}_i \odot \mathbf{s}_j^\star)\|_2^2/4,$$

  which counts the number of mismatched entries between $\mathbf{s}_i$ and $\mathbf{s}_j^\star$.

Figure 2.3 displays the results. We observe that for all SNR values, the best performance is achieved by 'Algebraic'. For $SNR > 10$, our approach becomes competitive and performs better than all other methods except 'Algebraic'. As with 'Algebraic', we can provide a proof of robustness [104]. Another advantage of our method is that it computes the rank-one factors in a greedy fashion: in noisy settings, we could leverage this property to estimate the rank depending on the reconstruction error after each rank-one factor is added.

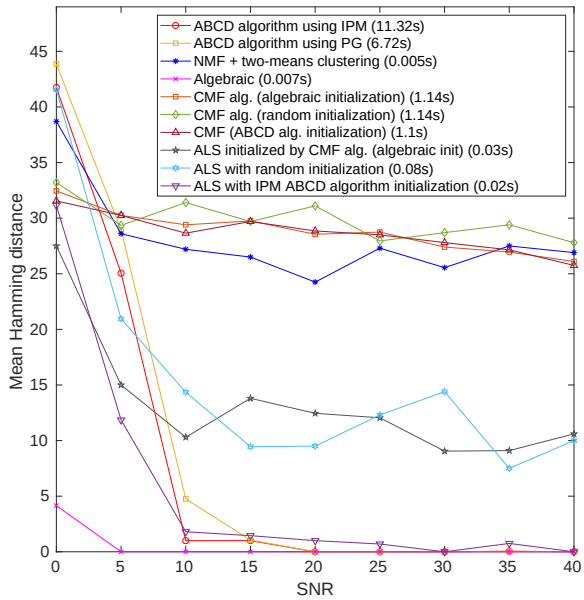

Figure 2.3: Mean Hamming Distance over 20 trials. The legend includes the average running time at SNR 20dB.

2.6.1.2 Scalability tests

Let us compare the execution time for the two methodologies presented to tackle ABCD. In this experiment, $\mathbf{C}$ is generated via a binary factor $\mathbf{Z}$, whose elements are drawn from a uniform distribution in $[0, 1]$ and then rounded to the nearest integer, while $\mathbf{W}$ has elements generated from a standard normal distribution. Standard normal noise $\mathbf{N}$ is added and the SNR for this experiment is at 13 dB. We set $\epsilon = 0.05$ for both methods. The number of trials for each instance considered is 20. The results are presented in the table below which provides the mean execution time.

(m, n, r)	$t_{IPM}(s.)$	$t_{PG}(s.)$
$(90, 90, 6)$	247.1626	82
$(140, 120, 5)$	1623.3	188.7
$(200, 200, 5)$	–	521.9
$(260, 290, 4)$	–	915.7

CVX could not solve the last two cases due to high memory requirements. We denote the result of these experiments as '-'. In one of the trials for $(m, n, r) = (140, 120, 5)$, the first-order method retrieved one element of the binary factor wrong. In the rest of the trials, all methods retrieved the binary factor exactly.

2.6.2 Comparison of the B-M approach versus the PG approach

In this section, we will be comparing the scalability of our previous PG method and the new B-M approaches. We initially test the B-M approach in both SSCD and ASCD and compare with our original results. We also consider additional larger datasets. We also run tests where we use the B-M method only for SSCD and not for ASCD, compare the performance gains and notice the distinct gains from using B-M in each SDP. For all experiments for the B-M method, we performed 1 alternating optimization iteration.

2.6.2.1 Extending the scalability experiments

We repeat the scalability experiments with new results from our B-M methods. We also report new results from our original PG method, where we perform more gradient steps for the ASCD decomposition. While in the original experiments we only performed 1 gradient step, here we perform 5, in order to improve the performance of the PG algorithm. For B-M we set $p = 3r$. We perform 20 trials and report on the average time.

(m, n, r)	$t_{IPM}(s.)$	$t_{PG}(s.)$	$t_{BM}(s.)$
$(90, 90, 6)$	247.1626	106.88	2.32
$(140, 120, 5)$	1623.3	240.18	4.64
$(200, 200, 5)$	–	677.8	6.59
$(260, 290, 4)$	–	704.71	12.35
$(500, 550, 5)$	–	7038	28.61
$(1000, 1100, 6)$	–	$> (24h)$	300.02

For the PG method we also report the following: For $(m, n, r) = (90, 90, 6)$, 3 trials did not find the ground truth, while for next three instances, one trial did not find the ground truth. For $(m, n, r) = (500, 550, 5)$ the average time increases significantly. Furthermore, out of the 20 trials, 3 could not find the ground truth. The B-M method was able to find the ground truth in all trials but one for $(m, n, r) = (1000, 1100, 6)$. Furthermore, for the latter parameters, the PG method exceeded a time limit of 24 hours, after which we terminated the algorithm.

2.6.2.2 Testing the two PG ASCD alternatives

Finally, we perform experiments where we use the B-M method for the SSCD, and compare between using just PG and using the B-M method for ASCD. We denote the time for using simple PG for ASCD as t_{PG_2}.

(m, n, r)	$t_{PG}(s.)$	$t_{PG_2}(s.)$	$t_{BM}(s.)$
$(90, 90, 6)$	106.88	17.85	2.32
$(200, 200, 5)$	677.8	103.77	6.59
$(260, 290, 4)$	704.71	166.35	12.35

We observe that the bulk of the performance gains are obtained by solving the SSCD using the B-M method. We also observe that not using B-M for the ASCD still keeps a considerable computational overhead, as there are still a lot of eigenvalue decompositions performed. Using B-M for ASCD as well allows for greater scalability: almost 50x faster for $(90, 90, 6)$, more than 100x faster for $(200, 200, 5)$ and 57x faster for $(260, 290, 4)$. Furthermore, the number of Eigenvalue decompositions dramatically decrease by using the B-M method for ASCD as well, and this is seen on the results reported above. More specifically, by using B-M for all SDPs, the only point where an eigenvalue decomposition is performed is in the enforcement of the $\begin{bmatrix} \mathbf{Z}\mathbf{Z}^\top & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix} \succeq \mathbf{0}$ constraint.

In these last two experiments, we showed that using B-M to solve the SDPs not only gives a considerable speedup in comparison to our simpler PG algorithms, but it also still performs well in finding the ground truth.

2.7 Conclusion

In this chapter, we extended the algorithms for SSCD, ASCD and ABCD proposed in [70, 71] to handle noisy data. We also proposed gradient methods to deal with the various SDP problems that arise in the decompositions. First, we proposed a PG algorithm to deal with the SDP of the SSCD decomposition. We also proposed the use of the Burer-Monteiro scheme to further scale the SDPs of the SSCD and the ASCD. Although this introduces more hyperparameters that need to be chosen well by the user, it provides a considerable improvement, while not compromising on its performance, in comparison to our work in [62].

Finally, we discuss possible future work directions for the work of this chapter.

- While we have provided many experiments comparing our methods, we have restricted ourselves to synthetic data. Future work may include testing our methods on real datasets and comparing them with other similar methods. We can also test other decompositions, like NMF, and see whether our methods can provide better approximations, in terms of error or in terms of more interpretable results.
- Another thing to note is that in general we experimented with lower ranks. If we were to repeat some of the scalability experiments with the same level of noise, for fixed n, m and with increasing rank, the probability of not finding the ground truth increases. It would be interesting to examine why this happens and how this can be improved.
- Another interesting work would be to understand why our methods were suffering less from noise, based on the figure 2.3.
- Finally, further improvement on the B-M method for ASCD could be done by factorizing entirely the block matrix

$$\begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{Y} \end{bmatrix},$$

and not just $\mathbf{X}$, to remove all psd constraints from the problem and scale the algorithm further.

Chapter 3

Novel Algorithms for Boolean Matrix Factorization

In this chapter we consider Boolean matrix factorization. We first propose an algorithm for BMF relying on alternating optimization (AO) using integer programming (IP). Building on this algorithm, we then use combining schemes to further enhance this algorithm and find better solutions.

We then propose new greedy methods, that lead to significantly more scalable algorithms. Through our experiments we show that they are competitive with the state of the art, even though they are more simple. We showcase the performance of all our proposed methods and compare with the state of the art on various real datasets, including applications in topic modeling and imaging.

The content of this chapter is adapted from the following publications:

- C. Kolomvakis, A. Vandaele and N. Gillis, "Algorithms for Boolean Matrix Factorization using Integer Programming," 2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP) [63],
- C. Kolomvakis, T. Bobille, A. Vandaele and N. Gillis, "Algorithms for Boolean Matrix Factorization using Integer Programming and Heuristics", submitted for journal publication.

This chapter is organized as follows: In Section 3.1, we formally define BMF, showcase BMF's benefits compared to bMF and the property of BMF of mining overlapping communities. We conclude the section by referencing related work. In Section 3.2, we describe our proposed alternating optimization (AO) algorithm for BMF where the subproblems are quadratic integer programs (IPs). We also provide two initialization strategies for the AO algorithm. In Section 3.3, we provide an IP formulation to optimally combine several BMF solutions and extend on what we previously proposed in [63]. In Section 3.4 we introduce our greedy algorithms to tackle the BMF problem. We start by showing how we solve the Boolean least squares problem and we finish the section by presenting greedy combining algorithms, similar

to Section 3.3. Section 3.5, is our experimental section where we test on a variety of real-world datasets, including topic modelling and imaging.

3.1 Boolean matrix factorization (BMF)

Let us first define the matrix Boolean product.

Definition 3.1 (Boolean matrix product) *Given two Boolean matrices,* $\mathbf{W} \in \{0,1\}^{m\times r}$ *and* $\mathbf{H} \in \{0,1\}^{r\times n}$*, their Boolean matrix product is denoted* $\mathbf{W} \circ \mathbf{H} \in \{0,1\}^{m\times n}$ *and is defined for all* i, j *as*

$$(\mathbf{W} \circ \mathbf{H})_{ij} = \bigvee_{k=1}^{r} \mathbf{W}_{ik} \wedge \mathbf{H}_{kj} = \bigvee_{k=1}^{r} \mathbf{W}_{ik}\mathbf{H}_{kj}, \tag{3.1}$$

where $\vee$ *is the logical OR operation (that is,* $0\vee 0 = 0$*,* $1\vee 0 = 0\vee 1 = 1$*, and* $1\vee 1 = 1$*) and* $\wedge$ *is the logical AND operation (that is,* $0\wedge 0 = 0$*,* $1\wedge 0 = 0\wedge 1 = 0$*, and* $1\wedge 1 = 1$*). For scalars that are in* $\{0,1\}$*, the Boolean AND and the standard multiplication are equivalent opeations. Interestingly, for matrices with binary elements, we can link the Boolean matrix product with the standard matrix product via the relation* $\mathbf{W} \circ \mathbf{H} = \min(1, \mathbf{WH})$ *where* $\mathbf{WH}$ *is the standard matrix product of* $\mathbf{W}$ *and* $\mathbf{H}$*.*

We now show an example of a Boolean matrix product. Let us consider

$$\mathbf{W} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \text{ and } \mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix},$$

then the Boolean matrix product is

$$\mathbf{W} \circ \mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}.$$

The Boolean operations restrict the elements of the product to be in $\{0,1\}$. If we performed standard matrix multiplication, the product instead would be

$$\mathbf{WH} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix},$$

giving us an element that is not in $\{0,1\}$.

Let us now define the BMF problem. To make it more general, we allow the input matrix, $\mathbf{X}$, to have missing elements. The matrix $\mathbf{M}$ is used to model incomplete datasets, that is, datasets for which some values are unavailable. The matrix $\mathbf{M}$ always has the same dimensions as the input matrix $\mathbf{X}$. If an element $\mathbf{X}(i,j)$ is observed, then $\mathbf{M}(i,j) = 1$, otherwise it is equal to 0. For complete datasets (where all of the elements are observed), $\mathbf{M}$ is the all-one matrix.

Definition 3.2 (BMF) *Given a Boolean matrix* $\mathbf{X} \in \{0,1\}^{m\times n}$*, a mask* $\mathbf{M} \in \{0,1\}^{m\times n}$*, and a factorization rank* r*, BMF aims to find matrices* $\mathbf{W} \in \{0,1\}^{m\times r}$ *and* $\mathbf{H} \in \{0,1\}^{r\times n}$ *that solve*

$$\min_{\mathbf{W}\in\{0,1\}^{m\times r},\mathbf{H}\in\{0,1\}^{r\times n}} \|\mathbf{M} \odot (\mathbf{X} - \mathbf{W} \circ \mathbf{H})\|_F^2, \tag{3.2}$$

where $\odot$ *is the Hadamard or elementwise product between two matrices of the same dimensions, and* $\|\cdot\|_F^2$ *is the squared Frobenius norm.*

BMF allows one to find subset of columns and rows of $\mathbf{X}$ that are highly correlated, since the entries equal to one in each binary rank-one factor $\mathbf{W}(:,k)\mathbf{H}(k,:)$ correspond to a rectangular submatrix of $\mathbf{X}$ that should contain many entries equal to one. The Boolean OR operation among the rank-1 factors gives more flexibility to the $\mathbf{W}$ and $\mathbf{H}$ factors, since any $1's$ of the input matrix can be approximated by multiple rank-1 factors. As an example, in community detection applications, this allows BMF to detect overlapping communities. In contrast, in bMF, the rank-1 factors should have disjoint support sets. Applications of BMF include role mining [79, 114], bioinformatics [54, 78] and computer vision [73].

3.1.1 Illustrative example

Let us provide an illustrative example of BMF in community detection and how it is able to detect overlapping communities. We consider a real binary dataset of 101 animals with 17 characteristics (for example, 'hairy', 'can be airborne', 'aquatic') [39, 52]. We consider $r = 3$. The factors for a submatrix of the input are as follows:

	hair	feathers	eggs	aquatic	milk
bass	0	0	1	1	0
bear	1	0	0	0	1
chicken	0	1	1	0	0
gorilla	1	0	0	0	1
ostrich	0	1	1	0	0
seahorse	0	0	1	1	0

$=$

1	0	0
0	0	1
0	1	0
0	0	1
0	1	0
1	0	0

$\circ$

0	0	1	1	0
0	1	1	0	0
1	0	0	0	1

Each column of $\mathbf{W}$ and each row of $\mathbf{H}$ correspond to communities; in other words, each rank one-factor $\mathbf{W}(:,k)\mathbf{H}(k,:)$ correspond to a community. The columns of $\mathbf{W}$ assign animals to the communities, while the rows of $\mathbf{H}$ assign the characteristics to these communities. In this example, we see observe that the first community contains the bass and the seahorse (first column of $\mathbf{W}$), and the characteristics "eggs" and "aquatic" (first row of $\mathbf{H}$). Hence the first community represents the class of aquatic animals. Similarly, the second community that includes ostrich and chicken and the characteristics "feathers" and "eggs" corresponds to birds. The third community includes the gorilla and the bear and its characteristics include "has hair" and "produces milk", and hence corresponds to mammals.

In Figure 1 represents a Venn diagram of the animals that are assigned to the communities. Note that some animals are assigned to multiple communities. Figure

2 represents a Venn diagram of the characteristics that are assigned to the communities. These two figures show some additional features not present in the submatrix above. Note that not all characteristics need to be present in an animal assigned to a community (this corresponds to a 0 entry in $\mathbf{X}$ approximated by a 1). For example, the penguin cannot be airborne, but is still correctly classified as a bird.

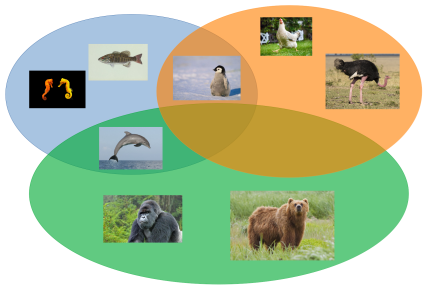

Figure 3.1: Venn diagram showing the overlapping communities that animals are assigned to.

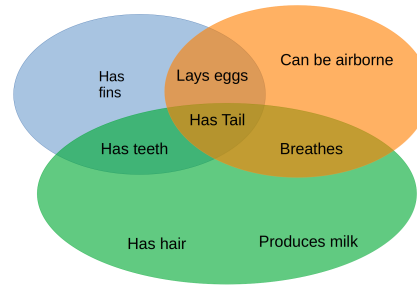

Figure 3.2: Venn diagram showing the overlapping communities that animal characteristics are assigned to.

3.1.2 Previous works

We finish this section by referencing previous important works on BMF. Our focus is on algorithms that have been proposed, identifiability results, and we also briefly mention models that are closely related to BMF.

3.1.2.1 Algorithms

In a similar manner to [89], our brief review of past BMF works is split into combinatorial approaches, continuous approaches where the BMF problem is relaxed into a continuous domain and other miscellaneous approaches.

Continuous approaches Common traits among continuous approaches (but not the only possible ones) are to either introduce penalty functions in order to force the factors to have Boolean elements, use a function on the product $\mathbf{WH}$ (a thresholding function for example) to give an approximation that is a Boolean matrix, or a combination of both.

In [90] the authors consider a non-linear function for the Boolean matrix product and add penalty terms to enforce the factors to have elements either 0 or 1. The authors then optimize on the nonnegative orthant. A similar work that optimizes in the nonnegative orthant and enforces the factors to be binary through a penalty term is in [111]. Works using proximal gradient algorithms can be found in [33, 55, 56]. More specifically, in [56] the proximal function is chosen to push the retrieved factors close to 0 or 1 and to ensure that the factors end up being binary, a post-processing procedure is performed that relies on grid search. The presented algorithm

is referred to as *Primp*. In [55], an extension is presented, which is called *C-SALT*. C-SALT is an algorithm that is suited to be used for supervised learning, in contrast to Primp (and usually most MF algorithms) for unsupervised learning. In [33], the authors propose different proximal functions. They add an l_1 and an l_2 regularization term to the factors (also known as an elastic-net regularizer) and each regularization term has a parameter assigned to it. To ensure that the factors are Boolean, the parameter of the l_2 term is gradually increased after each alternating optimization (AO) iteration. A projection step is only performed if the method ends and no convergence has happened (for example, the maximum number of iterations have been used). The first algorithm for federated BMF is presented in [34], where the authors use the elastic-net regularizing term, as well as a term that makes sure that all local processing units have the same factor locally stored.

Generalized versions of the Boolean matrix product are considered in [21, 41], where instead of the Boolean OR, other logical operations are used, such as the NAND and XOR. They then use a projected gradient to optimize the factors. *FastStep* [3], is a scalable algorithm that computes non-negative factors and then produces a Boolean approximation by applying a thresholding operator to the product $\mathbf{WH}$. The novelties of this algorithm are in the computation of the cost function and the gradient, to reduce their costs and make it scalable.

Probabilistic approaches to BMF also exist. In [98], a Bayesian approach to BMF is shown. The authors also fit the model through a metropolized Gibbs sampler. In [94], the BMF problem is recast as a maximum a posteriori (MAP) problem. The authors use probabilistic graphical models to implement a message passing algorithm that can also provide scalability. Finally, the work in [77] proposes an expectation minimization (EM) algorithm that does not make any prior assumptions for the factors. The work also shows an application in breast cancer subtype classification.

Combinatorial approaches We now begin our discussion on combinatorial algorithms. One of the earliest combinatorial algorithms for BMF is in [23]. The method is called 8M and it is part of a package called "Bio-Medical Data Package" (BMDP). 8M is revisited in [9] and expanded upon with a new algorithm, 8M+.

A seminal work on BMF is in [87], where the authors refer to the BMF problem as the *Discrete Basis problem* (DBP). A greedy algorithm called ASSO is presented that computes the factors based on the correlations between the columns of the input matrix. It is also proved that approximating the BMF problem is NP-hard. An extended journal paper of this work is in [88], where the authors present improvements to the ASSO algorithm and also examine a closely related problem which is called *Discrete basis partitioning problem* (DBPP). DBPP is like the DBP with the additional constraint that $\mathbf{H}$ is a partition. A matrix is a partition, if each column has exactly one non-zero element across all the rows. In contrast to DBP, DBPP can be approximated in polynomial time.

PANDA [81] is an algorithm that sequentially retrieves the rank-1 components one by one. While the user specifies the rank of the approximation, as is usually the case with matrix factorization algorithms, PANDA will stop as soon as adding an additional rank-1 component does not decrease the error. This means that it may

return an approximation with a number of components that is less than the rank given by the user. PANDA+ [80] extends the PANDA algorithm so it can handle a greater range of cost functions. Proximus [67] is an earlier algorithm that recursively extracts sequentially rank-1 components for an input matrix.

We will next mention some algorithms that make use of *Formal concept analysis* [26, 44], which is a mathematical framework that is used for data analysis. In [10], the authors present two algorithms: A greedy approximation algorithm that is based on an algorithm that solves the set covering problem. The second algorithm is a modified version of the first algorithm, where the search space is significantly reduced. While in [10] the algorithms are not named, in later works [8, 110] they are referred to, respectively, as GRECON and GRECOND. We also note that both of these algorithms are "from-below" algorithms. This means that if an entry in the input matrix is zero, then the corresponding entry of the approximation matrix will always be zero. GRECOND+ [8] is an extension of GRECOND. One of its new features over GRECOND is that it is not a "from-below" algorithm, it may fill an element of the approximation matrix with 1, even though the corresponding element in the input was zero. GRECON2 [110] is an improvement over GRECON in terms of execution time. The experiments of the paper show that the algorithm is considerably faster than both GRECON and GRECOND, while not compromising on its performance.

In [52] three formulations of the BMF problem are compared that are Integer Programs (IPs). They expand on a formulation that they initially presented in [65], and compare with other BMF algorithms, including other IP formulations, like from [66, 79]. Our previous work in [63] also falls into this category of combinatorial algorithms that solve IPs to compute a BMF.

Other works The work in [5] aims to simplify the computation of a BMF through the following method: reduce the number of 1-entries of the input matrix. Two versions of this method are presented: one in which the rank of the original matrix is not preserved, and one where it is. For the latter, guarantees are shown that an optimal BMF for the simplified matrix is also an optimal BMF for the original input matrix. CX and CUR decompositions [82] for binary matrices are presented in [86].

A different and unique approach is presented in [118]. In this paper, the authors permute the input matrix to transform it into "Upper-Triangular Like". Through geometric segmentation, they extract sequentially the rank-one factors. Genetic algorithms for BMF can be found in [38, 102, 103]. Another heuristic algorithm, which based on hillclimbing, is in [121]. This is another algorithm that falls into the category of "generalized" BMF algorithms, in this case the "OR" operation of the Boolean matrix product is substituted by the "XOR" operation.

Works on BMF that provide algorithms with theoretical guarantees (but sometimes also provide impractical algorithms), as well as approximating algorithms for BMF (also with theoretical guarantees) are in [7, 11, 18, 42, 43]. Finally, a recent work that uses collaborative neurodynamic optimization can be found in [76].

An interested reader is referred to a recent detailed survey on Boolean matrix factorization in [89]. The authors not only analyze various algorithms for the BMF problem, but also show problems that are equivalent to BMF (like bi-clique covering

[75] and the set cover problem [30, 45]), theoretical results, and several open problems.

3.1.2.2 Identifiability

Uniqueness results on BMF are limited. To the best of our knowledge, the only two papers that present uniqueness results are [35, 90]. Each paper presents a different approach.

The work in [90] proposes the definition of *partial uniqueness* with respect to a rank-1 factor. To be more precise, let us consider a Boolean decomposition of rank r and let us fix all rank-1 factors but one (we will refer to the non-fixed factor as $\mathbf{x}_k = \mathbf{w}_k \circ \mathbf{h}_k^\top$). If $\mathbf{x}_k$ is the only rank-1 factor that satisfies the decomposition, given the other fixed rank-1 factors, we say that the Boolean decomposition is *partially unique* with respect to $\mathbf{x}_k$. A Boolean decomposition is then considered unique if and only if it is partially unique with respect to all its rank-1 factors.

On the other hand, in [35] the authors use the notions of *freeness* and the *free rank* of a Boolean matrix. The freeness of two or mode Boolean vectors is presented as an extension of the well-known property of linear independence among vectors. Vectors that satisfy the freeness property are called free. Multiple equivalent definitions of the free rank are given. One of them is the following: A Boolean matrix $\mathbf{X} \in \{0,1\}^{m \times n}$ has a free rank equal to r if it contains a $r \times r$ permutation matrix. Furthermore, the authors distinguish between the free rank of a Boolean matrix and the Boolean rank of a Boolean matrix.

3.1.2.3 Related factorizations

We conclude our discussion on related works by referencing extensions to the BMF model that do not fall into the "generalized" BMF category of the previous paragraph. Our recent work in [64] presents Boolean Matrix tri-factorization (BMTF). It is defined as

$$\mathbf{X} = \mathbf{W} \circ \mathbf{S} \circ \mathbf{H} \in \{0,1\}^{n \times m},$$

where $\mathbf{W} \in \{0,1\}^{n \times r_1}$, $\mathbf{S} \in \{0,1\}^{r_1 \times r_2}$ and $\mathbf{H} \in \mathbb{R}^{r_2 \times m}$ as well an algorithm to compute it. We also propose identifiability conditions. The Boolean CUR model that we mentioned before is closely related to BMTF. A main difference between this and our work is that identifiability or interpretability are not a main focus of the latter's work. BMTF is the main focus of Chapter 4 of this thesis.

Finally, we will briefly refer to Boolean Tensor factorization (BTF). Tensor factorizations are generalizations of matrix factorizations, where the factor matrices are more than two, and furthermore, the operations that are used form a tensor. Matrix factorizations are, in fact, a specific case of tensor factorizations, where we are computing two factors. An interested reader may refer to [40, 85, 93, 97, 117] for works on BTF.

3.2 Alternating optimization (AO) for BMF

Most algorithms for LRMAs rely on iterative block coordinate descent methods: the subproblem in $\mathbf{H}$ is solved for $\mathbf{W}$ fixed, and vice versa. The reason is that these subproblems are typically convex in contrast to the main problem which is usually much more difficult to solve. For BMF, this is of course not the case, because of the Boolean matrix product. However, the advances in IP solvers, such as Gurobi [53], allow one to tackle medium-scale problems efficiently.

3.2.1 IP formulation for BMF subproblems

Assuming $\mathbf{W}$ is fixed in (3.2), we would like to solve the following Boolean least squares (BoolLS) problem in $\mathbf{H}$, that is, solve

$$\min_{\mathbf{H}\in\{0,1\}^{r\times n}} \|\mathbf{X} - \min(1, \mathbf{WH})\|_F^2.$$

Because of the nonlinearity in the objective, this cannot be solved directly with standard IP solvers. Note that the problem in each column of $\mathbf{H}$ is independent:

$$\min_{\mathbf{H}(:,j)\in\{0,1\}^{r}} \|\mathbf{X}(:,j) - \min(1, \mathbf{WH}(:,j))\|_2^2. \tag{3.3}$$

For simplicity, let $\mathbf{h} = \mathbf{H}(:,j)$ and $\mathbf{x} = \mathbf{X}(:,j)$. Given $\mathbf{W}$ and $\mathbf{x}$, we need to solve $\min_{\mathbf{h}\in\{0,1\}^r} \|\mathbf{x} - \min(1, \mathbf{Wh})\|_2^2$. Introducing the variable $\mathbf{z} = \min(1, \mathbf{Wh})$, (3.3) can be reformulated as follows:

$$\min_{\mathbf{h}\in\{0,1\}^{r},\mathbf{z}\in\{0,1\}^{m}} \|\mathbf{x} - \mathbf{z}\|_2^2 \quad \text{s.t.} \quad \frac{\mathbf{Wh}}{r} \leq \mathbf{z} \leq \mathbf{Wh}. \tag{3.4}$$

In fact, for $\mathbf{W}$, $\mathbf{h}$ and $\mathbf{z}$ binary, $\frac{\mathbf{Wh}}{r} \leq \mathbf{z} \leq \mathbf{Wh}$ if and only if $\mathbf{z} = \min(1, \mathbf{Wh})$, since $\mathbf{Wh} \in \{0, 1, \ldots, r\}^m$. Now (3.4) is a convex quadratic optimization problem with linear constraints over binary variables. Such problems can be solved with commercial software, and we make use of Gurobi [53]. Other alternatives that are commercial IP solvers are CPLEX [31] and Mosek [2]. An open source alternative is miOSQP [106]. For the case of missing data, we consider the following cost function

$$\min_{\mathbf{h}\in\{0,1\}^{r},\mathbf{z}\in\{0,1\}^{m}} \|\mathbf{m} \odot (\mathbf{x} - \mathbf{z})\|_2^2 \quad \text{s.t.} \quad \frac{\mathbf{Wh}}{r} \leq \mathbf{z} \leq \mathbf{Wh}, \tag{3.5}$$

where $\odot$ denotes the Hadamard (or elementwise) product, and $\mathbf{m}$ is a weight matrix such that $\mathbf{m}_i = 1$ if $\mathbf{x}_i$ is not missing, and 0 otherwise. Our BoolLS algorithm for the case where we have all the data available can be easily adjusted and applied to the missing data case.

To give an idea of the computational time required for Gurobi to solve (3.4), let us perform the following experiment for various values of m and r. The setting is as follows: we generate the entries of $\mathbf{W}$ and $\mathbf{h}$ using the uniform distribution in $[0, 1]$, and then threshold all elements to convert them to binary. For all ranks tested, apart from $r = 50$, if an element is larger than 0.7, it is converted to 1, otherwise we convert

it to 0. For $r = 50$, the threshold is set to 0.8. We chose relatively sparse $\mathbf{W}$ and $\mathbf{h}$ to make sure $\min(1, \mathbf{Wh})$ is not the all-one vector (in fact, we regenerate $\mathbf{W}$ and $\mathbf{h}$ if $\min(1, \mathbf{Wh})$ is the all-one or all-zero vector). Then we set $\mathbf{x} = \min(1, \mathbf{Wh})$, and 10% of the entries of $\mathbf{x}$ are flipped randomly[1]. Table 3.1 reports the results.

$r \backslash m$	100	1000	5000	10000
2	0.002	0.02	0.47	1.8
5	0.003	0.03	0.47	1.87
10	0.005	0.04	0.56	2.2
20	0.012	0.12	2.4	12.0
50	0.049	2.52	38.0	235

Table 3.1: Average execution time in seconds over 30 trials of Gurobi to solve noisy BoolLS problems (3.4) for various values of m and r.

The results are encouraging since solving (3.4) can be done exactly with Gurobi in a reasonable amount of time, even for relatively large problems, e.g., it takes on average Gurobi 12 seconds to solve this problem with $m = 10^4$ and $r = 20$. Note that one could also use a timelimit for Gurobi, so that Gurobi would return the best solution found within the allotted time (often IP solvers take much more time to guarantee global optimality rather than finding the optimal solution).

3.2.2 AO for BMF

We can now solve BMF via AO over the factors $\mathbf{W}$ and $\mathbf{H}$ alternatively. Since $\|\mathbf{X} - \min(1, \mathbf{WH})\|_F^2 = \|\mathbf{X}^\top - \min(1, \mathbf{H}^\top \mathbf{W}^\top)\|_F^2$, the problem in $\mathbf{W}$ for $\mathbf{H}$ fixed has the same form. We update $\mathbf{H}$ in a column-by-column fashion by solving the independent BoolLS of the form (3.3), and similarly for $\mathbf{W}$ row by row. Algorithm 12 summarizes the AO strategy. We have added a safety procedure within AO (steps 4-8): it may happen that some rows of $\mathbf{H}$ are set to zero (for example, if $\mathbf{W}$ is not well initialized). In that case, we reinitialize these rows as the rows of the residual $\mathbf{R} = \max\big(0, \mathbf{X} - \max(1, \mathbf{W}_i \mathbf{H}_i)\big)$ whose entries have the largest sum. This guarantees that the error will decrease after the update of $\mathbf{W}$. Typically, AO needs a very small number of iterations to converge to a local minimum, given the combinatorial nature of the problem.

3.2.3 Initialization of AO

In this section, we provide two initialization strategies for AO-BMF, that is, Algorithm 12.

[1]The noiseless BoolLS problem is much easier to solve since $\mathbf{h}_k = 1$ if and only if the support of the kth column of $\mathbf{W}$ is contained in that of $\mathbf{x}$, that is, $\mathbf{W}(:, k) \leq \mathbf{x}$.

Algorithm 12 AO algorithm for BMF - `AO-BMF`

Require: Input matrix $\mathbf{X} \in \{0,1\}^{m \times n}$, initial factor matrix $\mathbf{W}_0 \in \{0,1\}^{m \times r}$, maximum number of iterations maxiter.
Ensure: $\mathbf{W} \in \{0,1\}^{m \times r}$ and $\mathbf{H} \in \{0,1\}^{r \times n}$ such that $\mathbf{X} \approx \min(1, \mathbf{WH})$.

$i = 1$, $e(0) = \|\mathbf{X}\|_F^2$, $e(1) = \|\mathbf{X}\|_F^2 - 1$.
while $e(i) < e(i-1)$ and $i \leq$ maxiter **do**
  $\mathbf{H}_i = \texttt{BoolLS}(\mathbf{X}, \mathbf{W}_{i-1})$.
  $\mathcal{K} = \{k \mid \mathbf{H}_i(k,:) = 0\}$.
  if $\mathcal{K} \neq \emptyset$ **then**
    $\mathbf{R} = \max\big(0, \mathbf{X} - \max(1, \mathbf{W}_i\mathbf{H}_i)\big)$.
    $\mathbf{H}_i(\mathcal{K},:) = \mathbf{R}(\mathcal{I},:)$, where $\mathcal{I}$ contains the indices
    of the $|\mathcal{K}|$ rows of $\mathbf{R}$ with largest sum.
  end if
  $\mathbf{W}_i = \texttt{BoolLS}(\mathbf{X}^\top, \mathbf{H}_i^\top)^\top$.
  $i = i + 1$.
  $e(i) = \|\mathbf{X} - \min(1, \mathbf{W}_i\mathbf{H}_i)\|_F^2$.
end while
return $(\mathbf{W}, \mathbf{H}) = (\mathbf{W}_i, \mathbf{H}_i)$.

Randomly selecting columns or rows of X AO-BMF only requires $\mathbf{W}$ to be initialized. By symmetry, it could also be initialized only with $\mathbf{H}$, starting the AO algorithm by optimizing over $\mathbf{W}$. A simple, fast and meaningful strategy to initialize AO-BMF is to initialize $\mathbf{W}$ (resp. $\mathbf{H}$) with a subset of the columns (resp. rows) of $\mathbf{X}$, that is, set $\mathbf{W} = \mathbf{X}(:, \mathcal{K})$ (resp. $\mathbf{H} = \mathbf{X}(\mathcal{K}, :)$) where $\mathcal{K}$ is a randomly selected set of r indices of the columns (resp. rows) of $\mathbf{X}$.

NMF-based initialization The second initialization we propose relies on NMF. NMF approximates $\mathbf{X}$ with $\mathbf{WH}$ where $\mathbf{W}$ and $\mathbf{H}$ are nonnegative. We use an NMF algorithm from `https://gitlab.com/ngillis/nmfbook/` which itself initializes the entries of $\mathbf{W}$ and $\mathbf{H}$ using the uniform distribution in $[0, 1]$. Once an NMF solution is computed, we binarize it using the following two steps:

- Normalize the columns of $\mathbf{W}$ and rows of $\mathbf{H}$ such that

$$\max(\mathbf{W}(:,k)) = \max(\mathbf{H}(k,:)) \text{ for all } k,$$

  using the scaling degree of freedom in NMF, that is,

$$\mathbf{W}(:,k)\mathbf{H}(k,:) = (\alpha\mathbf{W}(:,k))(\alpha^{-1}\mathbf{H}(k,:)) \text{ for } \alpha > 0.$$

- Set the entries of $\mathbf{W}$ and $\mathbf{H}$ to 0 or 1 using a given threshold δ which is generated uniformly at random in the interval $[0.3, 0.7]$. An alternative would be to use a randomized grid search approach to determine δ, similar to [111, 125]. However, we observed that selecting δ randomly performs better on average.

It is important to note here that if the input matrix has missing elements, we can only use the NMF-based initialization.

Finally, due to the difficulty of the BMF problem, when testing the AO BMF algorithm, we perform multiple AO BMF runs, under a time limit, and report on the best performing. We refer to this as Multistart AO (MS-AO).

3.3 Combining multiple BMF solutions

In this section, we present algorithms whose aim is to improve the performance of AO-BMF. With BMF being an NP-hard problem, it is expected that any solution we find is a local minimum. To avoid this, we use combining schemes, which gather multiple solutions $(\mathbf{W}, \mathbf{H})$ and try to pick the r best rank-1 factors to minimize the error. In this section we are extending on our proposals from [63] with new algorithms.

3.3.1 MS-Comb-AO

Algorithm 12, AO-BMF, is able to relatively quickly generate locally optimal solutions for (3.2), in the sense that they cannot be improved by optimizing $\mathbf{W}$ or $\mathbf{H}$ alone. A first natural approach to generate good solutions to BMF is using multiple initializations, and keeping the best solution.

However, it is possible to combine a set of solutions in a more effective way. Assume we have generated p rank-r BMFs: $\mathbf{W}_1\mathbf{H}_1, \ldots, \mathbf{W}_p\mathbf{H}_p$. This gives rp binary rank-one factors, namely $\mathbf{W}_\ell(:,k)\mathbf{H}_\ell(k,:)$ for $k = 1, \ldots, r$ and $\ell = 1, \ldots, p$. Let us denote these rank-one binary matrices $\mathbf{R}_i$ for $i = 1, \ldots, N$ with[2] $N = rp$. To generate a better rank-r BMF, we can pick r rank-one binary factors among the $\mathbf{R}_i$'s, by solving the following combinatorial problem:

$$\min_{\mathbf{y}\in\{0,1\}^N} \left\| \mathbf{X} - \min\left(1, \sum_i y_i \mathbf{R}_i\right) \right\|_F^2 \text{ such that } \sum_i y_i = r.$$

The variable $y \in \{0,1\}^N$ encodes the r selected rank-one factors, that is, $y_i = 1$ if $\mathbf{R}_i$ is selected in the BMF. As for BoolLS, we can reformulate this problem as a quadratic IP:

$$\begin{aligned} &\min_{\mathbf{y}.\in\{0,1\}^N, \mathbf{Z}\in\{0,1\}^{m\times n}} \|\mathbf{X} - \mathbf{Z}\|_F^2 \\ &\text{such that } \sum_{i=1}^N y_i = r \text{ and } \frac{\sum_{i=1}^N y_i \mathbf{R}_i}{r} \leq \mathbf{Z} \leq \sum_{i=1}^N y_i \mathbf{R}_i. \end{aligned} \tag{3.6}$$

Denoting $\mathbf{y}^*$ the optimal solution of (3.6), the rank-r BMF obtained, $\sum_i y_i^* \mathbf{R}_i$, is guaranteed to be at least as good as all the solutions $\{\mathbf{W}_i\mathbf{H}_i\}_{i=1}^p$, since they are feasible solutions of (3.6). Once a solution combining several BMFs is computed, we will further improve it using AO.

[2] In practice, we delete duplicated rank-one factors so that $N \ll rp$.

We will refer to this algorithm, namely generating p solutions with AO-BMF, then combining them solving (3.6), and then applying AO to that solution, as MS-Comb-AO.
Finally, similarly to (3.5), we can also apply this combining scheme for the case where we have missing data.

Gurobi can solve medium-scale problems of the form (3.6) in a reasonable amount of time. Table 3.2 reports the time to solve (3.6) for $m = n = 101$, $r = 5$, and $N = 50 * 2^k$ for $k = 0, 1, \ldots, 4$. For example, it takes about 30 seconds, to combine 160 rank-5 solutions (hence 800 rank-one factors) of a 105-by-105 matrix.

N	50	100	200	400	800
time (s.)	4.0	4.3	7.6	14.2	30.6

Table 3.2: Run times to combine N rank-one factors for a 105-by-105 matrix with $r = 5$ (namely, the apb data set, see Section 3.5).

In practice, if N is too large, we do not need to take into account all rank-one factors, and can only consider rank-one factors corresponding to the best BMFs.

3.3.2 Tree-BMF

We now propose another method that uses MS-Comb-AO as a building block which we refer to it as Tree BMF. This method trades memory demands for time demands, that is, our goal is to reduce memory demands by combining less solutions, but perform the combining step multiple times, which may in return increase the time of the method. Figure 3.3 illustrates how our method works.
The method requires two parameters: the depth of the tree as well as the number of solutions for the MS-Comb-AO algorithm used by the leaf nodes. The procedure is as follows: At the leaf nodes, we solve multiple independent MS-Comb-AO problems according to the number of solutions specified by the user. Then, the leaf nodes send their best solutions to their parent and the parent combines using only two pairs of $(\mathbf{W}, \mathbf{H})$ factors. This continues until we reach the root node, where, after the combination step is performed, the final factors are returned. We can see that the bulk of memory demands is at the leaf nodes.

3.4 Greedy BMF heuristics

As we will show in Section 3.5, the IP based methods proposed in Sections 3.2.2 and 3.3 are competitive in comparison to the state of the art. However, using Gurobi prevents us from scaling our inputs. For this reason, we present greedy heuristics for the computation of a BMF. Greedy algorithms are often used as a way to solve NP-hard problems in a quick manner. The trade-off is that, usually, the solutions are suboptimal [30]. We start this section by proposing greedy algorithms to solve the Boolean least squares problem, which we then use to solve BMF. We continue

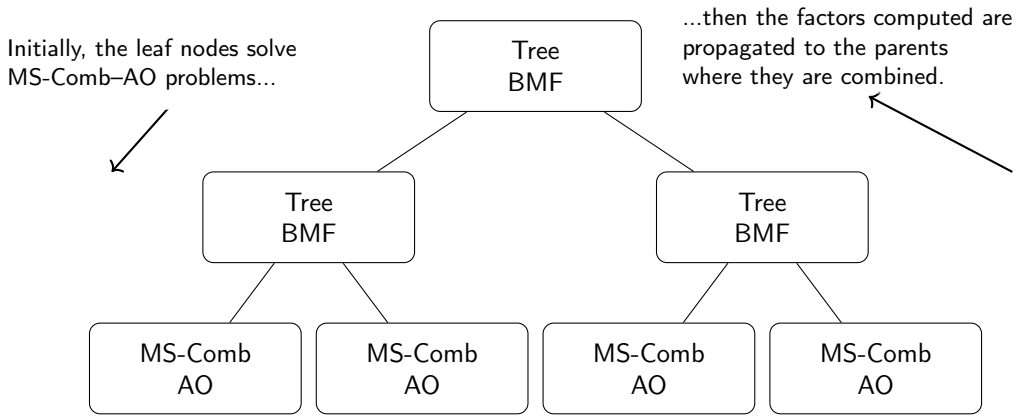

Figure 3.3: Sample Tree BMF with depth = 2. The order of the function calls for this tree is depth-first postorder. Initially, the leaf nodes solve instances of MS-Comb-AO and then propagate their solutions to the parent node, for it to combine. This procedure continues until we reach the root node, when the final factors are returned.

by proposing combining schemes, similar to Section 3.3, that we can use to further improve our solutions. These combining algorithms do not make any use of an IP solver, such as Gurobi.

3.4.1 Greedy Boolean LS

We start our discussion with a first step in designing greedy algorithms for BMF, that is, greedy algorithms to solve Boolean LS.

3.4.1.1 A first naive greedy algorithm

Recall the BoolLS problem:

$$\min_{\mathbf{h}\in\{0,1\}^r} \|\mathbf{x} - \mathbf{W} \circ \mathbf{h}\|_2^2. \tag{3.7}$$

The algorithm works as follows: $\mathbf{h}$ is initialized at the all-zero vector. The algorithm sets entries to one in a greedy manner: We test every column of $\mathbf{W}$ as part of the solution. The column that reduces the error the most is added as part of the solution in $\mathbf{h}$. If no column reduces the error, the algorithm terminates and returns the current $\mathbf{h}$. Any entries set to 1 are not set to zero later on.

At each iteration of this algorithm, the selection of the best column requires to compare the r vectors of $\mathbf{W}$ (this can be done in $\mathcal{O}(rm)$) and to update the element of $\mathbf{h}$ corresponding (this can be done in $\mathcal{O}(1)$). The number of iterations is bounded by the number of elements of $\mathbf{h}$ because we set to 1 at least one component per iteration. Therefore, the total complexity is in $\mathcal{O}(mr^2)$.

For $r = 1$, the greedy algorithm is optimal. There are only two possible solutions, $h = 0$ or $h = 1$. However, for $r > 1$, due to its greedy nature, it can miss the optimal

solution to a Boolean LS problem.
Consider the following

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ \mathbf{W} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The greedy algorithm will initially pick the first column and the error will be 2. It will also test the second and third columns, which give an error of 3. After testing all the columns, the optimal choice is the first column, so we set $\mathbf{h} = [1 \ 0 \ 0]^\top$. The algorithm continues by checking if we can add the second or the third columns as part of the solution. Adding the second column does not further decrease the error, while adding the third column increases the error by 1. Since none of the possible choices further decreases the error, the algorithm terminates and returns $\mathbf{h} = [1 \ 0 \ 0]^\top$ as the solution. However, $\mathbf{h} = [0 \ 1 \ 1]^\top$ is the optimal solution, which gives an error of 1. In the remainder of this section, we discuss additional algorithms to circumvent suboptimality and get better performance.

3.4.1.2 Local Search Strategy

We propose an additional step to this greedy algorithm. The solution obtained is perturbed by switching certain entries to improve the solution. We limit ourselves to switch at most q bits that are chosen randomly. In our implementation we choose $q = \lceil \log r \rceil$, where $\log r$ has a base of 2. With this q, once an approximation $\mathbf{h} \in \{0,1\}^r$ is found we search randomly in the q-radius ball centered on this vector to find if there is a better solution. The ball is defined as

$$B(\mathbf{h}) = \{\mathbf{s} \in \{0,1\}^r \mid \|\mathbf{h} - \mathbf{s}\|_2^2 \leq q\}.$$

If we obtain a better solution, we recursively call the algorithm on this solution. We set the limit of recursive calls as $T = r$. The algorithm is presented in Algorithm 13. We will refer to the combination of the greedy algorithm and LOCALSEARCH as *Greedy-BLS*.

It is worth noting that the inner for loop of the algorithm (which is the part of the algorithm that performs the perturbations on $\mathbf{h}$) starts at $k = 2$ and not $k = 1$. This means that if we are to run the Greedy-BLS algorithm for $r = 2$, the LOCALSEARCH step will effectively be skipped. This is because all possible solutions of $\mathbf{h}$ ([0 0], [1 0], [1 1] or [0 1]) are explored and there is no need to use LOCALSEARCH.

3.4.2 Greedy combining methods

This algorithm is a loose adaptation of MS-Comb-AO, where the "building block" algorithm is *Greedy-BoolLS*. We generate many solutions and store all the rank-one

Algorithm 13 LocalSearch algorithm

Require: $\mathbf{h} \in \{0,1\}^r, \mathbf{x} \in \{0,1\}^m, \mathbf{W} \in \{0,1\}^{m \times r}, T \in \mathbb{N}$
T is the max number of recursive calls, q the radius of the search.
Ensure: $\mathbf{h} \in \{0,1\}^r$
if $T = 0$ **then**
return **h**
end if
for $i \leftarrow 1, 2, \ldots, T$ **do**
for $k \leftarrow 2, \ldots, q$ **do**
$\mathbf{u} \leftarrow \mathbf{0}_{\{0,1\}^r}$ // We want to look at another solution vector at distance k from h.
while $\sum_j^r u(j) < k$ **do**
$idx \leftarrow \mathcal{U}(\{1, 2, \ldots, r\})$ // uniform distribution of the set $\{1, 2, \ldots, r\}$
$\mathbf{u}(idx) \leftarrow 1$
end while
h' $\leftarrow \mathbf{h} \oplus \mathbf{u}$ (XOR operator)
if $||\mathbf{x} - \mathbf{W} \circ \mathbf{h'}|| < ||\mathbf{x} - \mathbf{W} \circ \mathbf{h}||$ **then**
$\mathbf{h} \leftarrow$ LocalSearch($\mathbf{x}$, $\mathbf{W}$, **h'**, $T - 1$)
end if
end for
end for
return **h**

factors. The rank-one factors are stored, in a vectorized form, as the columns of a matrix that we call $\mathbf{U}$. Let us assume that the number of all rank-one factors collected is N and let us also consider a vector $\mathbf{y}_2 \in \mathbb{N}^r$. The vector $\mathbf{y}_2$ functions similarly to the vector $\mathbf{y}$ of MS-Comb-AO (it is a vector that picks the best rank-1 factors among all of the collected ones) but is implemented differently. $\mathbf{y}$ is in $\{0,1\}^N$, where $\mathbf{y}_j = 1$ means that the j-th rank-one factor was chosen, and with the additional constraint that the values of y must sum up to r. $\mathbf{y}_2$ is a vector in $\mathbb{N}^r$ whose elements designate which rank-1 factors we are picking. If, for example, the j–th rank-one factor is collected, then one of the values of $\mathbf{y}_2$ will be j.

We then consider the initial best solution (the best rank-r solution among the solutions gathered by the multiple *Greedy-BoolLS* calls), and try to improve it by randomly switching some of its rank-one factors. The algorithm is very similar to the LocalSearch algorithm. However rather than applying it vector-per-vector, it is applied to the rank-one factors. This is shown in Algorithm 14 and we will refer to it as *Heur-Comb*. We refer to the whole algorithm (the gathering of solutions through the *Greedy-BoolLS* algorithm and the collection of r rank-1 factors through Algorithm 14) as *Greedy-Comb*. The total number of rank-1 factors collected is N. Furthermore, similarly to section 3.3.2, we can also collect solutions gathered from multiple calls to Greedy-Comb and greedily pick the best solutions. This means substituting the gathering algorithm from *Greedy-BoolLS* to *Greedy-Comb*. We will refer to this version of the algorithm as *Greedy-TreeBMF*.

Regarding the two for loops of the algorithm, the maximum number of iterations is a pair of two parameters set by the user.

Algorithm 14 Heur-Comb

Require: $\text{vec}(\mathbf{X}) \in \{0,1\}^{mn}$, columns of $\mathbf{U} \in \{0,1\}^{mn \times N}$ are vectorized rank-one factors, $\mathbf{y}_2 \in [N]^r$ collects the r selected rank-one indices, $T_{\max} \in \mathbb{N}$ is the maximum number of improvements, $n_$trials is the number of trials per potential improvement.

Ensure: $\mathbf{y}_2 \in [N]^r$

best_err $= \|\mathbf{X} - \sum_{j=1}^{r} \mathbf{U}(\mathbf{y}_2(j), :)\|_F$ // initial error of the selected rank-one factors
$T = 1$, iter $= 1$
while iter $\leq n_$trials and $T \leq T_{\max}$ **do**
  $u\_idx \leftarrow \mathcal{U}(\{1, 2, \ldots, N\} \backslash \mathbf{y}_2)$ // pick the index of a column of U not in $\mathbf{y_2}$.
  $idx \leftarrow \mathcal{U}(\{1, 2, \ldots, r\})$ // pick an index of a column of U in $\mathbf{y_2}$
  $\mathbf{y}_2' = \mathbf{y}_2$, $\mathbf{y}_2'(idx) \leftarrow u\_idx$ // $\mathbf{y}_2'$ swaps the two above indices.
  if $\|\mathbf{X} - \sum_{j=1}^{r} \mathbf{U}(\mathbf{y}_2'(j), :)\| <$ best_err **then**
    $\mathbf{y}_2 \leftarrow \mathbf{y}_2'$
    best_err $= \|\mathbf{X} - \sum_{j=1}^{r} \mathbf{U}(\mathbf{y}_2(j), :)\|_F$
    T = T+1, iter = 1
  else
    iter = iter+1
  end if
end while
return $\mathbf{y}_2$

3.4.3 Custom data structure for Boolean vectors and matrices in C++

The proposed greedy algorithms were written in C++. C++ is a low-level programming language, which means that it allows for control over hardware (for example, directly reading and writing into memory with greater efficiency than in a language like Python or Julia). Although the language provides a default `bool` data type, we created a custom data structure for Boolean matrices that uses less space than a two-dimensional array that contains `bool` elements and can perform operations faster. Furthermore, while there exist libraries for linear algebra in C++, like Armadillo [101] or eigen [51], they do not take full advantage of the properties of the Boolean algebra.

In this section, we are going to provide more details regarding the data structures used for the implementation of Boolean matrices and the corresponding operations in C++ that will allow for greater scalability. We will make a brief mention of some of the fundamental datatypes in C++, as well as their storage requirements. As a low-level programming language, unlike MATLAB, Python, or Julia, declaring a variable requires a data type as an additional argument. A variable can be, among others, a `char` (which can store a single character and its size is 1 byte, i.e., 8 bits),

an `int` (which can store any integer from -2^{31} to 2^{31} and its size is 4 bytes), a `double` (which can store high precision floating point numbers within the intervals $\pm(2.23 * 10^{-308}, 1.797 * 10^{308})$ and its size is 8 bytes) and a `bool` (which takes the values 'true' and 'false' and its size is 1 byte). We notice that even in the case of the `bool` variable, there is the least requirement of 8 bits (= 1 byte). Due to Boolean operations only using the values '0' and '1', our goal is to create a data structure that solely uses 1 bit to represent them. For example, if we define a two-dimensional array of Boolean values, each element requires 8 bits instead of 1. This gives considerable improvements in terms of memory usage and computation time for Boolean matrix operations.

Our design starts by initially creating a custom data structure which we will refer to as a `bitset` data structure which we will use as a one dimensional array. Normally, if we define an array of `bool` elements of size m, we use m bytes without the ability to manipulate individual bits separately. So we are using 8 bits to represent a single `bool` variable. With the `bitset` data structure, our goal is to modify bits separately, as well as to represent each element of the array with only one bit. Our `bitset` data structure uses an `int` array together with bitwise operations (for example, bitwise-or, bitwise-and, as well as functions that modify an element or print an element of the array). When we define a `bitset` array of m elements, since each bit corresponds to one element of the array, the size of this custom array is of $\lceil \frac{m}{32} \rceil$ elements. If, for example, we define a `bitset` array of 30 elements, although to the user it will seem as they can normally modify and access 30 elements, in reality, we are only using just one `int` variable. To illustrate further the memory benefits, we will use an additional example, where we compare with a standard int array.

Let us compare a `bool` array with 50 elements and a `bitset` array with 50 elements. For the former, we are using 50*(8 bits) = 400 bits. For the latter, the array is of size $\lceil \frac{50}{32} \rceil = 2$. This means that we are only using 64 bits instead of 400 bits. 50 bits are used for the array, while the other 14 remain unused. To the user, the bitset array still appears as an array of 50 elements, each element is just a bit. This can also be viewed as compression; instead of using an array of n elements, our final array is much smaller. All the operations will be done in a bitwise manner - and, or, xor operations.

Our final goal, however, is to create Boolean matrices in an efficient manner, not just one dimensional arrays. To that end, we define another data structure `bitmatrix`, which contains a two dimensional bitset array that we will use to represent Boolean matrices, as well as the definitions of the various operators that we need to use - Boolean matrix multiplication, addition, element access and element retrieval, to mention a few. We are going to observe in Section 3.5 that this allows for great scalability in comparison to our Gurobi-based mixed integer programming (MIP) methods.

In the next section, we present an experiment that shows that our custom implementation outperforms other linear algebra libraries in C++. We emphasize that this implementation was developed in-house, without initially being aware of existing, more general, bit-level matrix representations. Our objective was not to provide a full-featured Boolean linear algebra framework, but rather a lightweight and easily

adaptable data structure implementing only the operations required for our purposes. Another example of a custom data structure specifically designed to handle Boolean matrix operations can be found in [88].

3.5 Numerical Experiments

All experiments are performed with a 12th Gen Intel(R) Core(TM) i7-1255U 1.70 GHz, 16GB RAM. In the experiments where we use the methods that solve MIPs, we are using Julia v. 1.10.

3.5.1 Comparison of our `bitmatrix` data structure

We start this section with an experiment that shows the gains made from our custom data type. Our setting is the following: We perform Boolean matrix multiplication between two square matrices of size $n \times n$, for $n \in \{100, 500, 1000, 5000, 10000, 20000\}$, and whose entries are rounding of the uniform distribution in $[0, 1]$ (hence these matrices have on average as many zeros as ones)[3]. We choose this operation as this is one of the most demanding and ubiquitous matrix operations. We are comparing with the Eigen and Armadillo C++ linear libraries. From these libraries, the matrices that we are using are of `float` type, which means that each element represents a real number, and uses 32 bits to store it.

The reason why we choose each element to be of `float` type, instead of `bool` or `uint_8` (where each element would instead be represented by 8 bits), is because of the use of the BLAS routines [13], which are only available for `float` and `double` matrices. BLAS (which stands for "Basic Linear Algebra Subprograms") are routines that provide standard building blocks for performing basic vector and matrix operations. Its matrix multiplication routine, which is called GEMM (which stands for 'general matrix multiplication'), is the heavily optimized function that is widely used for dense matrix–matrix multiplication.

We present the time it took to execute the multiplications (in seconds), for each data size and each data structure considered in Tables 3.3, 3.4. We are reporting averages over 10 trials.

n	100	500	1000	5000
`bitmatrix`	7.21×10^{-5} s.	**0.003 s.**	**0.008 s.**	**0.375 s.**
Eigen [51]	$\mathbf{5.093 \times 10^{-5}}$ **s.**	0.004 s.	0.032 s.	3.05 s.
Armadillo [101]	0.0004 s.	0.017 s.	0.047 s.	3.44 s.

Table 3.3: Results of the Boolean matrix multiplications for the multiple sizes considered and all the data structures considered. Average run times for 10 such multiplications.

[3]Note that the Boolean product of such matrices will be, with very high probability, the all-one matrix. The goal here is to compare computational and memory demand of different data structures.

n	10000	20000
`bitmatrix`	**3.71 s.**	**40.49 s.**
Eigen [51]	24.88 s.	195.23 s.
Armadillo [101]	26.11 s.	203.86 s.

Table 3.4: Results of the Boolean matrix multiplications for the multiple sizes considered and all the data structures considered. Average run times for 10 such multiplications.

We see that our `bitmatrix` data structure performs better than the other libraries, for all data sizes considered except for $n = 100$, where Eigen has the best performance. For $n = 500$, `bitmatrix` performs slightly better than Eigen and much better than Armadillo. Finally, for all $n \geq 1000$ considered, our `bitmatrix` data structure performs much better than the other two libraries. We observe that taking into account the properties of Boolean algebra allows for greater performance gains. This experiment confirms that our decision to create a custom data structure, specifically for Boolean matrices and their operations, to obtain memory and performance gains was worth it.

3.5.2 Experiments from the datasets in [52]

We next perform experiments on four real binary data sets with no missing data and four real binary data sets with missing data used in [52], which come from [39, 68]; see Tables 3.5 and 3.6. As in [52], we use $r = 2, 5, 10$ for all data sets.

	zoo	heart	lymp	apb
$m \times n$	101×17	242×22	148×44	105×105

Table 3.5: Four binary real-world data sets.

	tumor	hepatitis	audio	votes
$m \times n$	339×24	155×38	226×92	435×16
#missing	670	334	899	392
%observed	24.3	47.2	11.3	49.2

Table 3.6: Four binary real-world incomplete data sets.

In [52], the authors proposed two non-trivial IP-based approaches for BMF that perform well against the state of the art (they used a 20 minutes time limit for their method), namely against a greedy scheme [52], ASSO and ASSO++ [88], a penalty formulation from [125], and an NMF-based heuristic. Table 3.7 reports the best result

of all these methods for the four data sets. From now on, we will report the result of our methods, by comparing it with the best solution from [52] provided in Table 3.7.

	$r = 2$	$r = 5$	$r = 10$
zoo	271	126	39
heart	1185	737	419
lymp	1184	982	728
apb	776	684	573
tumor	1352	962	514
hepatitis	1264	1138	907
audio	1419	1064	765
votes	1246	779	240

Table 3.7: Objective function $\|\mathbf{M} \odot (\mathbf{X} - \min(1, \mathbf{WH}))\|_F^2$ of the best solution found by various algorithms in [52, Table 4, page 20].

Results from two other papers We have also run two recent algorithms, from [33] and [21], on these data sets. The method in [33] is based on a continuous reformulation of BMF and uses alternating proximal projected gradient method to solve it. We have used the parameters of the algorithm recommended by the authors. Table 3.8 reports the results. Out of curiosity, we initialized AO with the best solutions found. We observe in Table 3.8 that AO is able to significantly improve these solutions, showing that the continuous reformulations of [33] is not able to generate locally optimal solutions.

	$r = 2$	$r = 5$	$r = 10$
zoo	+11 | 0	+5 | -1	+18 | +5
heart	+65 | +18	+29 | -1	+33 | +26
lymp	+64 | +25	+88 | -10	+203 | +60
apb	+72 | +30	+59 | +43	+33 | +15

Table 3.8: Best result obtained with the method in [33] (left), and result of this best solution improved by AO (right). The numbers indicate the difference compared with the best values in Table 3.7. A negative value means an improvement, a positive value means a worse solution.

Next, we show the results for the method in [21]. Note that this paper provides a rather general approach, allowing for other Boolean operations between the factors to approximate the entries of $\mathbf{X}$. Their approach is also based on some continuous reformulations, and the use of gradient and descent methods. Table 3.9 reports their result, using 15 trials where each trial uses 10 random initializations, each optimized

with 2000 iterations. We also use AO to improve the best solutions found by this method, and observe a similar behavior as for the method from [33].

	$r = 2$	$r = 5$	$r = 10$
zoo	+2 | 0	+7 | +4	+10 | +4
heart	+165 | +9	+33 | +4	+86 | +57
lymp	+214 | -3	+49 | -16	+123 | +5
apb	+48 | 0	+51 | +23	+112 | +43

Table 3.9: Best result obtained with the method in [21] (left), and result of this best solution improved by AO (right). The numbers indicate the difference compared with the best values in Table 3.7.

In summary, on these data sets, we observe that the algorithms from [21, 33] provide solutions which are much worse than the best solutions provided in [52], and that AO can considerably improve the solutions of these two methods, sometimes obtaining better solution than the best from [52] (e.g., for the lymp data set with $r = 5$).

Before introducing the results from our algorithms, we explain in more detail how we set up the parameters based on the time limits.

- **MS-AO:** We generate as many BMFs as possible with AO-BMF within the time T, and return the best solution. The initialization of AO-BMF is chosen alternatively as one of the two strategies (NMF-based or random columns/rows of $\mathbf{X}$).

- **MS-Comb-AO:** As explained in Section 3.3, we generate as many BMFs as possible with AO-BMF (as for MS-AO) within time $3T/4$, and then combine them by solving (3.6) with a time limit of $T/4$. We used the same random seed so that the solutions generated are the same as for MS-AO, except that less solutions are generated, since only 3/4 of the total time is spent for that.

- **Tree-BMF:** We consider the depth of the tree to be 1. For $T = 30$ seconds, the leaf nodes will gather 5 solutions, while for $T = 5$ minutes, they will gather 15 solutions. When moving to the next level of the tree, the given time to compute a BMF is divided by two. Furthermore, in a non-leaf node, the combining step is also performed in $T/2$ time.

- **Greedy-Comb:** For both $T = 30$ seconds and $T = 5$ minutes, we gather solutions using Greedy-BoolLS for T seconds to fill the $\mathbf{U}$ matrix. This constitutes a Monte Carlo trial. We repeat 5 trials and report the best solution.

- **Greedy-TreeBMF:** We again consider the same time limits. We gather the solutions from several Greedy-Comb calls and greedily pick the final solution among them. For $T = 30$ seconds, Greedy-Comb is called 3 times and each call has a limit of 10 seconds. For $T = 5$ minutes, Greedy-Comb is called 5 times

and each call has a limit of 60 seconds, to fill the **U** matrix. This is a Monte Carlo trial and we report the best results among 5 trials.

3.5.2.1 Results and discussion

We present the results in Tables 3.10 - 3.12. For all methods we test for $T = 30$ seconds and for $T = 5$ minutes, like in [63].

- **MS-AO:** Quite surprisingly, MS-AO is already able to perform on par or improve upon the state of the art. With a 30 seconds time limit, it does on 8 out of 12 cases: 5/12 cases with improvements, sometimes significant as for the lymp data set, and 3/12 cases with the same objective. However, for 4 data sets, it is not able to achieve the best solution reported in Table 3.7, although the generated solutions are only slightly worse (+6 at most). With a 5 minutes time limit, it performs better or on par on 10 out of the 12 cases, and it produces a slightly worse solution in two cases (+2).
- For the incomplete datasets, when T = 30 seconds, there are 4/12 cases where our method performs better, 2/12 cases where it performs as well as the competing methods and 6/12 cases where it provides worse results. There is considerable improvement when we let our algorithm run for $T = 5$ minutes: In 7/12 cases, the algorithm provides a better solution, in 2/12 cases the error is as good as the best competing solution and there are only 3/12 cases where we get a worse solution.

		MS-AO		MS-Comb-AO		Tree BMF		Greedy Comb		Greedy TreeBMF	
$r = 2$	zoo	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	heart	+2	+2	+2	**0**	+2	+2	+2	+2	+2	+2
	lymp	**-10**	**-10**	**-10**	**-10**	**-10**	**-10**	**-10**	**-10**	-7	-7
	apb	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$r = 5$	zoo	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**
	heart	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**
	lymp	-25	-32	-25	-32	-27	-34	**-38**	**-38**	-33	**-38**
	apb	+6	-6	-1	-6	-6	**-7**	-6	-6	**-7**	-6
$r = 10$	zoo	+3	**0**	**0**	**0**	**0**	**0**	+7	+3	+8	+6
	heart	**0**	**0**	**0**	**0**	**0**	**0**	+18	+18	+18	+18
	lymp	-15	**-34**	-15	**-34**	-19	**-34**	-12	-24	-9	-31
	apb	+4	+2	+2	**-7**	**-7**	**-7**	-1	-2	-2	-4

Table 3.10: Results for all our proposed methods for both timelimits considered. This Table is for the datasets with no missing elements. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined.

		MS-AO		MS-Comb-AO		Tree BMF	
r=2	tumor	+2	**-1**	**-1**	**-1**	**-1**	**-1**
	hep/tis	**0**	**0**	**0**	**0**	**0**	**0**
	audio	**-8**	**-8**	**-8**	**-8**	**-8**	**-8**
	votes	**0**	**0**	**0**	**0**	**0**	**0**
r=5	tumor	+17	-7	-3	-10	-11	-10
	hep/tis	-119	-139	-120	-129	-128	-135
	audio	-24	-25	-17	-25	-25	-25
	votes	+35	-12	+27	-8	-60	**-78**
r=10	tumor	+68	+18	-2	**-4**	**-4**	-2
	hep/tis	-113	-145	-107	-151	-145	-154
	audio	+16	+16	+2	-12	-3	**-16**
	votes	+295	+213	+135	+104	+153	+52

Table 3.11: Part of our results for all our proposed methods for both timelimits considered. This Table is for the datasets with missing elements. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined.

		Greedy Comb		Greedy TreeBMF	
r=2	tumor	**-1**	**-1**	**-1**	**-1**
	hep/tis	**0**	**0**	**0**	**0**
	audio	**-8**	**-8**	**-8**	**-8**
	votes	**0**	**0**	**0**	**0**
r=5	tumor	-5	**-19**	-1	**-19**
	hep/tis	-138	-143	-136	**-148**
	audio	-26	**-29**	-9	-27
	votes	-66	-66	-66	-66
r=10	tumor	+15	+8	+3	+3
	hep/tis	-147	-157	-146	**-158**
	audio	+41	+39	+37	+30
	votes	+20	-4	+27	**-15**

Table 3.12: The remaining part of our results for all our proposed methods for both timelimits considered. This Table is for the datasets with missing elements. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined.

- **MS-Comb-AO:** For some data sets, AO is already able to generate very good solutions, and hence solving (3.6) is not useful, e.g., for the lymp data set. However, for most cases, this combination is beneficial, sometimes significantly. In particular, with the timelimit of 30 seconds on the apb dataset for $r = 5$, the

best solution found by AO-MS has error 689 (+6) while the combination leads to an error of 682 (-1); for the zoo data set with $r = 10$, it goes from 42 (+3) to 39 (0). A similar behavior is observed for 5 minutes: for the apb data set with $r = 10$ from +2 to -7, and the heart data set from +2 to 0.

MS-Comb-AO with a 5-minute timelimit is able to either perform on par with the state of the art (we suspect that for these data sets, the corresponding solutions are optimal), or outperform it, sometimes significantly (in particular for the lymp data set).

On the incomplete datasets, for $T = 30$ seconds, we observe that in $7/12$ cases we get an improved solution, in $2/12$ cases we get a solution with an error as good as the competing methods and in $3/12$ cases we get a solution that is worse. When we increase the time limit to $T = 5$ minutes, we have $9/12$ cases where we get an improved solution, $2/12$ cases where our solution does not improve on the competitors and only 1 case where our method performs worse.

- **Tree-BMF:** For the "heart" dataset and for $r = 2$ it seems to be giving a result that is not competitive. However, for all other cases it offers a considerable improvement over MS-Comb-AO. For $6/24$ cases have a result as good with the competing methods and for the rest $17/24$ cases, Tree BMF provides a better result.

- **Greedy-Comb:** The results are encouraging. We observe that in $13/24$ cases the method gives us improvements over the solutions in [52], in $5/24$ cases we get solutions as good as the competing methods. Finally, we have $6/24$ cases where our solutions are not as good. We observe that this method performs considerably better on the "votes" dataset than our MIP based methods.

- **Greedy-TreeBMF:** It provides an improvement over Greedy-Comb. For $15/24$ cases, the method gives us improvements over the solutions in [52], in $4/24$ cases we get solutions as good as the competing methods. Finally, we have $5/24$ cases where our solutions are not as good. A noticeable result is that Greedy-TreeBMF is the only method that can provide an improvement over the 'votes' dataset.

Conclusions: Both for the greedy methods as well as for the MIP Gurobi methods, adding complexity (in the sense that we are utilizing methods that combine more solutions) can improve our results. The greedy methods generally give competitive results (with some exceptions, namely for the 'heart' dataset for $r = 2$ and $r = 10$ and for the 'zoo' dataset for $r = 10$.) It is interesting that in general the greedy methods, despite being less complex than the Gurobi methods are competitive. Similar conclusions can be drawn for the incomplete datasets. There are even some cases where the greedy methods outperform the Gurobi ones - most notably for the 'votes' dataset, for all ranks considered, or the 'hepatitis' dataset for $r = 5$. For these datasets, all of our methods outperform the state-of-the-art, with a slight edge for the Gurobi methods on the complete datasets and a slight edge for the greedy methods on the incomplete datasets.

3.5.3 Application to topic modeling

In this section, we wish to test BMF's performance on large datasets. We consider the NIST tdt2 dataset which contains news stories from 1998 among 30 different topics [22]. Our input matrix $\mathbf{X}$ consists of 9394 documents with 19528 words. Throughout this section, we are going to work with subsets of the original dataset that we create as follows: We perform NMF on the original dataset - let us refer to it as $\mathbf{X}$ - where we consider the NMF rank at r_{NMF}, obtain the w most frequent words per topic from the $\mathbf{W}$ factor and the d most frequent documents per topic from the $\mathbf{H}$. We then work on this subset of $\mathbf{X}$ which has only those particular words and documents, which we will denote as $\hat{\mathbf{X}}$. We later binarize the matrix by assigning all nonzero values to 1. We will denote this binarized matrix as $\hat{\mathbf{X}}_b$. We consider three subsets according to Table 3.13.

	r_{NMF}	w	d	Dimensions of $\hat{\mathbf{X}}_b$
small dataset	20	20	50	302×917
medium dataset	20	150	30	1608×517
large dataset	30	1300	800	10729×8200

Table 3.13: The three subsets of $\mathbf{X}$ we will consider in this section and the parameters that we used to create them.

Algorithms used for the larger datasets For the rest of this chapter, other than our proposed algorithms, we are also comparing with the following algorithms:

- **ELBMF** [33]: An algorithm which uses proximal gradient with both an l_1 and an l_2 regularizing term. We picked this algorithm as it is a recent algorithm that falls into the category of continuous algorithms for BMF. As we will see later, it scales very well.
- **Methods in [6].** This paper introduces methods that search for undercover approximations for the BMF problem (also known as "from-below" approximations). The authors use a MAXSat encoding to solve the optimal Boolean k-undercover problem. MaxSAT is the optimization version of the Boolean Satisfiability Testing (SAT) problem. We compare with two of the methods in the paper: `FastUndercover`, which is a greedy algorithm, as well as `optiblock*` which uses `FastUndercover` as an initialization. Even though these methods are placing emphasis on avoiding the type of errors that we previously mentioned, the experiments in [6] show that they are comparable to other state-of-the-art algorithms that solve BMF.
- **ASSO [88].** As a reference we are also comparing with the seminal ASSO algorithm [88].

We picked the methods in [33] and [6] as they are more recent methods and the approaches differ among each other. We picked ASSO, as it is a seminal work and is also an algorithm in the combinatorial category.

Summary of results We provide a summary of the results of all the methods that we tested in Tables 3.14 and 3.15. If an entry is filled with this symbol '-', this means that we could not run the method, because the memory requirements were too great.

What we immediately notice is that the methods in [6] are noticeably behind the other methods. Among all the methods we compared, ASSO appears, on average, to perform very well and also to be fast. Our IP based methods seem to generally perform the best. However, these methods could not be tested on all datasets, and this was especially the case for MS-Comb-AO and Tree-BMF, which could only be tested on the small dataset. Our Greedy methods are performing well, being on average close second algorithms to our IP algorithms. ELBMF performs better than FastUndercover and Optiblock*, but it generally trails our proposed methods and ASSO. In the rest of the section, we give more details on the parameters used for our methods and we also show the topics extracted from the large dataset by using the GreedyComb algorithm.

	Small		Medium	
Algorithm	$r = 10$	$r = 20$	$r = 10$	$r = 20$
AO-BMF	84.54% (83s)	78.19% (125.7s)	<u>94.05%</u> (224.11s)	**90.71%** (363.71s)
Ms-Comb-AO	<u>84.36%</u> (357.77s)	<u>78.13%</u> (582.66s)	-	-
Tree-BMF	**84.31%** (780.7s)	**78.12%** (610.66s)	-	-
Greedy-Comb	84.94% (7.24s)	79.27% (23.69s)	94.24% (16.75s)	91.01% (79.84s)
Greedy-TreeBMF	84.61% (11.1s)	79.25% (37.21s)	**93.93%** (40.71s)	<u>90.96%</u> (111.82s)
FastUndercover [6]	98.43% (2.623s)	97.66% (4.45s)	98.8% (17.79s)	98.81% (11.79s)
OptiBlock* [6]	97.87% (2248.96s)	96.52% (5869.15s)	98.52% (18573s)	-
ELBMF [33]	91.5% (0.28 s)	88.93% (0.36s)	95.08% (0.68s)	94.1% (0.95s)
ASSO [88]	85.43% (0.64s)	82.26% (1.21s)	95.43% (1.05s)	94.11% (1.91s)

Table 3.14: Best results on the relative errors for all the document subsets and all the different algorithms considered. In parenthesis, we note the mean time over all Monte Carlo trials (if multiple trials are performed for a method). In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined.

	Large	
Algorithm	$r = 10$	$r = 20$
AO-BMF	-	-
Ms-Comb-AO	-	-
Tree-BMF	-	-
Greedy-Comb	**98.56%** (989.17s)	98.11% (7497.2s)
Greedy-TreeBMF	98.68% (1786s)	98.45% (5162.74s)
FastUndercover [6]	-	-
OptiBlock* [6]	-	-
ELBMF [33]	99.23% (66.69s)	98.29% (74.05s)
ASSO [88]	98.66% (333.41s)	**98.1%** (686.3s)

Table 3.15: Best results on the relative errors for all the document subsets and all the different algorithms considered. In parenthesis, we note the mean time over all Monte Carlo trials (if multiple trials are performed for a method). In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined.

We notice that the relative errors are high. For this reason, we also show tables that show the percentage of 1s on the original input matrix that are covered by each algorithm.

	Small		Medium		Large	
Algorithm	$r = 10$	$r = 20$	$r = 10$	$r = 20$	$r = 10$	$r = 20$
AO-BMF	45.39%	55.85%	19.79%	**27.73%**	-	-
Ms-Comb-AO	45.39%	57%	-	-	-	-
Tree-BMF	45.39%	**57.1%**	-	-	-	-
Greedy-Comb	43.1%	54.96%	18.66%	25.61%	4.96%	6.22%
Greedy-TreeBMF	**46.2%**	55.84%	**19.82%**	26.04%	4.54%	5.29%
FastUndercover [6]	13.25%	22.1%	5.27%	8.7%	-	-
OptiBlock* [6]	15.48%	25.09%	7.65%	-	-	-
ELBMF [33]	22.21%	25.05%	21%	16.38%	**5.46%**	5.99%
ASSO [88]	44.86%	55.68%	19.32%	25.69%	4.85%	**6.42%**

Table 3.16: Best results on the number of 1s covered, for all the document subsets and all the different algorithms considered. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined.

We see that this metric can also provide interesting insights. While the results mostly follow the previous results, where our metric is the relative error, there are a few interesting observations. For example, for the large dataset, for $r = 10$, ELBMF is the best performing algorithm. Furthermore, we also noticed that for $r = 20$, ELBMF

covers a smaller number of 1s, than for $r = 10$, which is a strange result. Finally, the number of 1s covered on the medium and large datasets are small (in accordance to the high relative errors in the previous tables). However, as we will see in the next subsection, we can still use BMF to extract meaningful topics from these datasets.

3.5.3.1 Topic mining through our methods

IP methods For AO-BMF we initially consider $r = 10$. Because AO-BMF gives different results depending on the initialization, we will be performing 10 runs and examine the results of the algorithm. The size of the input matrix is 302×917. If we consider $\mathbf{W}_b$ and $\mathbf{H}_b$ to be the factors obtained from AO-BMF, we cannot extract topics through observing the factor $\mathbf{W}_b$, since it has solely $\{0, 1\}$ values and cannot distinguish words based on their importance. An alternative way to do this extraction is through the matrix

$$\mathbf{W}_t = \mathbf{W}_b \odot (\hat{\mathbf{X}}\mathbf{H}_b^\top). \tag{3.8}$$

Each topic is made of a set of words in documents in $\mathbf{W}(:, k)\mathbf{H}(k, :)$. To retrieve the importance of a word we can count how many times it is used by the documents which can be computed via the relation (3.8).

We present a Table which shows all the topics retrieved through all the runs and how many times each topic appears.

For the run with the least relative error, we present at Tables 3.18 and 3.19 the words which contain the 10 most frequent words per topic. The columns correspond to topics and the i–th row corresponds to the word with the i–th highest frequency. Finally, we note that the average relative error, $\frac{\|\mathbf{X}_b - \min(1, \mathbf{W}_b\mathbf{H}_b)\|_F}{\|\mathbf{X}_b\|_F}$, is 84.54%.

Topic	#times the topic was retrieved
1998 US-Iraq crisis	10
Clinton - Lewinsky scandal	10
Israeli - Palestinian conflict	10
1998 visit of the pope to Cuba	10
1998 winter Olympics	10
1998 Southeast Asian economic crisis	10
Tobacco master settlement agreement	10
1998 Indian elections	8
1998 IMF help to Indonesia	6
1998 Indian - Pakistani nuclear tests	3
1998 Super Bowl	3
Asian stock markets	3
Ted Kaczynski trial	2
Indiscernible topic #1	2
Indiscernible topic #2	1

Table 3.17: List of topics retrieved and number of times each topic appears among all the 10 runs.

Indian elections	Winter Olympics	Asian stock markets	Indian - Pakistani nuclear tests	Southeast Asian economic crisis
india	nagano	market	india	crisis
party	olympic	percent	nuclear	economic
hindu	olympics	stock	pakistan	economy
indias	games	asian	tests	financial
government	japan	prices	minister	asia
congress	winter	investors	states	international
delhi	team	asia	prime	government
parliament	gold	economy	united	asian
election	medal	end	indias	market
political	win	average	test	percent

Table 3.18: Table for topics 1 - 5 after using AO-BMF.

Tobacco master settlement agreement	Israel Palestine conflict	Clinton - Lewinsky scandal	Visit of the pope to Cuba	1998 US Iraq crisis
companies	israel	president	cuba	iraq
tobacco	israeli	clinton	pope	united
industry	netanyahu	house	cuban	saddam
smoking	palestinian	white	john	weapons
congress	arafat	washington	paul	iraqi
legislation	bank	lewinsky	visit	states
senate	minister	monica	havana	gulf
washington	palestinians	counsel	castro	military
cigarette	prime	independent	fidel	nations
government	west	case	popes	officials

Table 3.19: Table for topics 6 - 10 after using AO-BMF.

We now wish to explore how MS-Comb-AO performs under this task. We consider the matrix $\hat{\mathbf{X}}_b$ from the run that gave us the previous two Tables and apply our combining algorithm. We set a timelimit of $T = 60$ seconds, and then set the number of solutions that MS-Comb-AO will consider to 10. The best relative error from all AO-BMF runs was 84.54%, and MS-Comb-AO improves it to 84.36%. The average time for the BMF runs is 13.2 seconds, while the average error was 85.45%. The words retrieved for each topic are presented in Tables 3.20 and 3.21.

As a reference, we are also comparing with the performance of an NMF algorithm, implemented according to [28] and [1]. The average relative error over 50 experiments is 71.09%. It is lower than the corresponding error of our BMF methods, however it should be noted that the factors computed from NMF are much more expressive.
We notice that the 1998 winter Olympics topic is switched with an indiscernible topic related to the presidency of Bill Clinton. Furthermore the other retrieved topics are the same, with some minor changes in the words retrieved.

Indian elections	Indiscernible topic	Southeast Asian economic crisis	Indian - Pakistani nuclear tests	Asian stock markets
india	president	economic	india	market
party	clinton	crisis	nuclear	percent
hindu	states	economy	pakistan	stock
indias	washington	financial	tests	asian
congress	united	government	minister	prices
government	end	international	indias	investors
parliament	officials	billion	states	economy
delhi	american	south	test	asia
election	house	united	united	average
political	political	banks	weapons	economic

Table 3.20: Table for topics 1 - 5 after using MS-Comb-AO.

Tobacco master settlement agreement	Israel Palestine conflict	Clinton - Lewinsky scandal	Visit of the pope to Cuba	1998 US Iraq crisis
companies	israel	president	cuba	iraq
tobacco	israeli	clinton	pope	united
industry	netanyahu	house	cuban	saddam
smoking	palestinian	lewinsky	john	iraqi
congress	arafat	washington	paul	weapons
legislation	bank	white	visit	states
senate	minister	monica	havana	gulf
washington	palestinians	counsel	castro	military
cigarette	prime	independent	fidel	nations
settlement	west	starr	popes	officials

Table 3.21: Table for topics 6 - 10 after using MS-Comb-AO.

Finally, we also tested Tree BMF. For $r = 10$, we used the following parameters: depth $= 2$, number of solutions for the leaf nodes are set to 10, the number of children nodes are set to 3, the upper time limit was set to $T = 1000$ seconds. Furthermore, we choose the alternating initialization scheme when we solve the AO-BMF problems in the MS-Comb-AO nodes. The topics found are the same, with minor changes in the top words retrieved. The relative error was improved further to 84.31% and the algorithm finished after 780.696 seconds.

Greedy methods We are testing the Greedy-Comb on the following setting: Due to the scalability of the method, we will be using the 'large' subset. When we run Greedy-Comb, to further showcase the scalability of this method in comparison to the previous ones, we will be considering $r = 20$. We will also choose among 25 initial factorizations.

Modifying Greedy-Comb to enhance the diversity of solutions In several initial experiments that we ran, we noticed that several topics may be mined multiple times (that is, most words belonging to two columns of $\mathbf{W}$ are the same). For this reason, in order to get more diverse topics, we introduce an additional procedure, which is performed as an additional step at the end of the Greedy-Comb algorithm. Its steps are the following:

After getting an initial solution $(\mathbf{W}, \mathbf{H})$, we construct $\mathbf{H}^\top\mathbf{H}$. This matrix represents the correlation among the clusters retrieved for the topics. More specifically, $(\mathbf{H}^\top\mathbf{H})_{i,j}$ is a measure of how similar the clusters i and j are (i.e., the number of common documents between these two clusters). When $i = j$, then this product represents the amount of documents that are assigned to cluster i. Ideally, we want the values in the diagonal to be non-zero. If there is a diagonal element equal to zero, this would mean that we extracted a topic for which there are no documents that correspond to it. Furthermore, we also desire for them to not have values that are close to zero, we would like to extract topics (i.e., clusters) for which there are many

documents about it in the dataset. Finally, we would like for non-diagonal values to be as close to zero as possible. This would mean that the topics extracted are different from each other. To this end, we propose the following procedure:

- For $\mathbf{H}^\top\mathbf{H}$, we initially check the diagonal elements. Through trial and error, the threshold that we consider acceptable for a diagonal value is 10, that is, if any diagonal element is less than 10, we return its index idx and through the mapping $\mathbf{h}(idx)$, we remove it as a possible solution from the collection of solutions $\mathbf{U}$. We pick a new solution through a modification of algorithm 14 where instead of finding a completely new h, we only substitute the index that was removed.

- If all diagonal elements are greater or equal to 10, we move to the next step. If we find a non-diagonal element of $(\mathbf{H}^\top\mathbf{H})_{i,j}$ such that it is 'close' to $(\mathbf{H}^\top\mathbf{H})_{i,i}$ we also return the corresponding index and remove it from $\mathbf{U}$. This is because that would mean that the clusters i and j have a closely related topic and this is something that we want to avoid. Our measure of 'close' is the ratio $\frac{(\mathbf{H}^\top\mathbf{H})_{i,i}}{(\mathbf{H}^\top\mathbf{H})_{i,j}}$. As before, through trial and error, the threshold we consider is 8, that is, if the ratio is less or equal to 8, we remove the solution from the collection and then find a substitute.

This procedure continues until either: The conditions above are satisfied, or the number of remaining rank-1 factors in the collection is equal to r, in which case we return the solution. This version of algorithm 14 has the following distinctions:

- We only change one index from $\mathbf{U}$.

- As soon as we find another rank-1 factor that reduces error, we immediately terminate the algorithm and return the new index.

The topics mined from our experiment in Tables 3.22-3.24. Because we are handling a larger dataset and we are using a higher rank than in the previous subsection, there will be topics mined that were not originally retrieved in Table 3.17.

We can see that the method succeeds in retrieving unique topics. The relative error is at 97.02% and the method finished after 23884.8 seconds, a little less than 6 hours and 40 minutes. While this relative error can be considered high, it should be noted that the original dataset is unstructured and sparse. Even at this level of relative error we can still mine meaningful topics.

One thing we would like to note is that the introduction of these thresholds for $\mathbf{H}^\top\mathbf{H}$ and our aim for discrete topics mined may give us a solution that is not as good in terms of relative error. In cases where we removed these thresholds or made them much less restrictive, the relative error was smaller. This is examined in more detail in the next subsection.

Clinton Lewinsky scandal	Israeli Palestinian conflict	Ted Kaczynski trial	Indian Pakistani nuclear tests	1998 Superbowl	Southeast Asian econ. crisis	1998 US Iraq crisis
'president'	'israel'	'kaczynski'	'nuclear'	'denver'	'economic'	'iraq'
'lewinsky'	'netanyahu'	'lawyers'	'india'	'packers'	'percent'	'weapons'
'clinton'	'israeli'	'defense'	'pakistan'	'super'	'crisis'	'united'
'house'	'palestinian'	'trial'	'tests'	'game'	'government'	'iraqi'
'white'	'peace'	'judge'	'indias'	'bowl'	'indonesia'	'inspectors'
'starr'	'arafat'	'kaczynskis'	'test'	'broncos'	'asian'	'annan'
'lawyers'	'palestinians'	'case'	'weapons'	'green'	'economy'	'council'
'jones'	'talks'	'prosecutors'	'indian'	'bay'	'billion'	'baghdad'
'case'	'minister'	'burrell'	'united'	'elway'	'financial'	'saddam'
'sexual'	'west'	'court'	'minister'	'yards'	'asia'	'security'

Table 3.22: Table for topics 1 - 7 after using Greedy-Comb.

Jonesboro school shooting	Tobacco master settlement agreement	Winter Olympic games	Easing of US embargo of Cuba	Court martial of Gene McKinney	1998 Cavalese cable car crash	1998 NBA finals
'school'	'tobacco'	'olympic'	'cuba'	'mckinney'	'italian'	'game'
'students'	'industry'	'nagano'	'cuban'	'sergeant'	'italy'	'bulls'
'boys'	'bill'	'olympics'	'castro'	'sexual'	'marine'	'jordan'
'jonesboro'	'companies'	'games'	'embargo'	'major'	'crew'	'jazz'
'middle'	'legislation'	'gold'	'american'	'army'	'plane'	'malone'
'teacher'	'smoking'	'medal'	'visit'	'court'	'cable'	'chicago'
'mitchell'	'congress'	'team'	'government'	'gene'	'military'	'points'
'shooting'	'senate'	'won'	'clinton'	'women'	'accident'	'pippen'
'fire'	'tax'	'japan'	'flights'	'martial'	'flying'	'left'
'arkansas'	'cigarette'	'winter'	'humanitarian'	'misconduct'	'feet'	'team'

Table 3.23: Table for topics 8 - 14 after using Greedy-Comb.

Death of MLK's assassin	Visit of the pope to Cuba	Execution of a US convict	Lawsuit against Oprah Winfrey	Indian elections	Algerian civil war
'ray'	'cuba'	'death'	'winfrey'	'party'	'algeria'
'king'	'pope'	'tucker'	'texas'	'india'	'government'
'james'	'cuban'	'penalty'	'oprah'	'government'	'algerian'
'earl'	'castro'	'texas'	'show'	'hindu'	'islamic'
'martin'	'visit'	'execution'	'cattle'	'congress'	'algiers'
'luther'	'church'	'executed'	'beef'	'bjp'	'killed'
'assassination'	'john'	'row'	'disease'	'election'	'violence'
'kings'	'paul'	'clemency'	'cow'	'seats'	'militants'
'family'	'havana'	'woman'	'amarillo'	'indias'	'armed'
'rays'	'popes'	'case'	'mad'	'parliament'	'military'

Table 3.24: Table for topics 15 - 20 after using Greedy-Comb.

3.5.3.2 Performance of other competing methods

In this section, we are testing the performance of other methods in the literature. Our measure of comparison is the relative error. We are comparing with recent methods that use different approaches. As a reference point, we also compare with the ASSO method of [88].

ELBMF [33]. We initially test the method in [33]. The method has many parameters that are assigned by the user. The parameters used are the following: `l1_reg` = `l2_reg` = 0.01, `maxiter` = 100, `tol` = 10^{-10}, `beta` = 10^{-4} and regularization rate $\lambda = 1.0133$. For the remainder of this paper, we consider that for all reporting of the ELBMF algorithm, the number of Monte Carlo trials is 15.

- For the small dataset, for $r = 10$, the best relative error was 91.5%, the mean time is 0.28 seconds and the median time is 0.26 seconds. For $r = 20$, the best relative error was 88.93%, the mean time is 0.36 seconds and the median time is 0.34 seconds.

- For the medium dataset, for $r = 10$, the best relative error was 95.08%, the mean time is 0.68 seconds and the median time is 0.66 seconds. For $r = 20$, the best relative error was 94.1%, the mean time is 0.952 seconds and the median time is 0.945 seconds.

- For the large dataset, for $r = 10$, the best relative error was 99.23%. Furthermore, the average time needed for a trial to finish is at 66.69 seconds, while the median time is at 66.45 seconds. For $r = 20$, the best relative error that this method gave us was 98.29%. Furthermore, the average time needed for a trial to finish is at 74.05 seconds, while the median time is at 73.35 seconds.

Greedy-Comb and Greedy-TreeBMF. In the previous subsection when we tested the performance of Greedy-Comb, we mentioned that we made changes to the algorithms in order to retrieve meaningful topics. We also noted that this can lead to a suboptimal solution in terms of relative error. We will now test Greedy-Comb and Greedy-TreeBMF on all document datasets considered, without these changes, and see whether the relative reconstruction error can be improved.

- For the small dataset, for Greedy-Comb, when considering $r = 10$, it finished after 7.24 seconds with a relative error of 84.94%. For $r = 20$, it finished after 23.69 seconds with a relative error of 79.27%. For Greedy-TreeBMF, for $r = 10$, we are setting a soft limit of 8 seconds. The results were a relative error of 84.61% and it finished after 11.1 seconds. For $r = 20$, we are setting a soft limit of 24 seconds. The results were a relative error of 79.25% and it finished after 37.21 seconds.

- For the medium dataset, as before, we gather 10 solutions when using Greedy-Comb. For $r = 10$, the method finishes after 16.75 seconds with a relative error of 94.24%, while for $r = 20$, the method finishes after 79.84 seconds with a relative error of 91.01%. For Greedy-TreeBMF, for $r = 10$, we are setting a limit of 30 seconds. The relative error was at 93.93% and finished after 40.71 seconds. For $r = 20$, we are setting a limit of 80 seconds. The relative error was at 90.96% and finished after 111.82 seconds.

- For the large dataset, for Greedy-Comb, we gather 10 solutions for $r = 10$ and 15 for $r = 20$. For the former rank, the method finished after 989.165 seconds (approximately 16 and a half minutes) with a relative error of 98.56%. For the latter rank, the method finished after 7497.2 seconds (approximately 2 hours and 4 minutes and 57 seconds) with a relative error of 98.11%.
  For Greedy-TreeBMF, when $r = 10$ we set the timelimit to 1000 seconds. The algorithm finished after 1786 seconds (a little less than half an hour) with a relative error of 98.68%. For $r = 20$, the timelimit of the Greedy-TreeBMF

algorithm was set to 3000 seconds. It finished after 5162.74 seconds (around 1 hour and 26 minutes) with a relative error of 98.45%. We notice that Greedy-TreeBMF did not perform as well as Greedy-Comb. A reason for this could be that it needs more solutions and more time to provide a better solution.

We are noticing that for the last two smaller datasets, Greedy-TreeBMF performs better than Greedy-Comb, as it has more time to explore solutions and the size of the datasets is not prohibitively large. Furthermore, while the terminating condition for the gathering step for *Greedy-Comb* is different, in comparison to the previous run in section 3.5.3.1, for a shorter run time (6796.41 seconds vs. 23884.8 seconds), a better relative error was achieved (96.34% vs. 97.02%).

Methods in [6]. We move on to the algorithms from [6].

- For the small dataset, the `FastUndercover` method, for $r = 10$, the method finished after 2.623 seconds and gave a relative error of 98.43%. For $r = 20$, we got an error of 97.66% and the method finished after 4.45 seconds. For `Optiblock*`, for $r = 10$ we got a relative error of 97.87% and it finished after 37 minutes and 28.96 seconds. For $r = 20$, we got a relative error of 96.52% and the method finished after 1 hour, 34 minutes and 49.15 seconds.

- For the medium dataset, the `FastUndercover` method, for $r = 10$, got a relative error of 98.8% and the method finished after 17.79 seconds. For $r = 20$, we got a relative error of 98.81%. Interestingly, for a higher rank, the method finished quicker, at 11.79 seconds. For `Optiblock*` we could only test for $r = 10$. The relative error was 98.57% and finished after 5 hours 9 minutes and 33 seconds. For $r = 20$, the method needed a very long time and did not finish after two days of running. As a result, we terminated it before it finished.

- Finally, we could not test the two methods on the large dataset. The method requirements were too large for our machine to handle.

ASSO [88]. Finally, we examine the performance of ASSO. The algorithm requires a threshold parameter to search for associations among columns in the input matrix. Through trial and error, we found a good threshold parameter for each of the datasets considered.

- For the small dataset, we considered the threshold equal to 0.4. For $r = 10$, it finished after 0.641 seconds, with a relative error of 85.43%. For $r = 20$, it finished after 1.214 seconds, with a relative error of 82.26%.

- For the medium dataset, we considered the threshold equal to 0.3. For $r = 10$, it finished after 1.045 seconds, with a relative error of 94.64%. For $r = 20$, it finished after 1.911 seconds, with a relative error of 93.04%.

- For the large dataset, we considered the threshold equal to 0.3. For $r = 10$, it finished after 333.406 seconds, with a relative error of 98.66%. For $r = 20$, it finished after 686.297 seconds, with a relative error of 98.10%.

We can see in Tables 3.14 and 3.15 that ASSO provides competitive results and it is one of the fastest algorithms tested. In particular, it is the best performing method among all those considered in this case, with Greedy-TreeBMF being very close.

3.5.4 Application on facial images

Summary of results We show a summary of the results of methods in Table 3.25. MS-Comb-AO, Tree BMF and the methods in [6] will not be in the Table since due to the memory and computational demands of the methods.

Method	Best result for $r = 10$	Best result for $r = 20$
AO-BMF	**59.61%** (1318.86s)	**54.74%** (4233.59s)
Greedy-Comb	60.2% (47.83s)	56.75% (222.49s)
Greedy-TreeBMF	<u>60.18%</u> (55.74s)	<u>56.65%</u> (159.96s)
ELBMF [33]	74.66% (7.7s)	69.66% (19.36s)
ASSO [88]	67.19% (23.45s)	65.62% (36.55s)

Table 3.25: Table that compares the best relative error of most of the methods tested for the binarized CBCL dataset. In parenthesis, we note the mean time over all Monte Carlo trials. In bold is the best performing method for a given dataset and rank, with the second best performing method being underlined.

We see that our AO-BMF method generally performs the best. Our greedy methods are then a close second, with Greedy-TreeBMF being slightly better than Greedy-Comb. It is also worth noting that AO-BMF is considerably slower than the other methods. For $r = 10$, a single ao-iteration is comparable to the whole runtime of the other methods and for $r = 20$, a single ao-iteration is slower than the whole runtime of the other algorithms. ELBMF and ASSO trail greatly to our proposed methods.

3.5.4.1 Gurobi-based methods

As in [63], we apply our algorithms to the CBCL facial image data set. Each column of the data matrix $\mathbf{X} \in \mathbb{R}^{361\times 2429}$ contains a vectorized facial image of size 19×19, and is not binary but satisfies $\mathbf{X}(i, j) \in [0, 1]$ for all (i, j). Our AO-BMF algorithm can be applied to any input matrix $\mathbf{X}$, even if it is not binary. We use the NMF initialization scheme for AO-BMF with $r = 20$ only, since it provides better results. We report the average time per AO iteration over 5 trials and the best relative error $\frac{\|\mathbf{X} - \min(1, \mathbf{WH})\|_F}{\|\mathbf{X}\|_F}$ among them. The maximum number of AO iterations is set to 15. The best relative error achieved is 55.94% and the average ao iteration lasted 244.309

seconds. To note the performance boost provided by using Julia, we repeated the experiment with the NMF initialization using the MATLAB version of [63]. The average time per ao iteration is 516.806 seconds. We see that Julia provides a significant speedup.

Not all of the other methods that we are going to be testing can handle real valued data (our greedy methods are such methods). For this reason, for the remainder of this section, we will be utilizing a binarized version of the dataset. We binarize it by using the mean value of each image as a threshold (if the pixel is above black, otherwise it is white). We are testing for both $r = 10$ and $r = 20$. The initialization scheme that we use is only through the use of NMF, since we have noticed that this yields better results. For $r = 10$, the best relative error that we get is 59.61% and the average AO iteration is at 98.84 seconds. For $r = 20$, the best relative error that we get is 54.74% and the average AO iteration is at 474.42 seconds. Figure 3.4 displays the meaningful binary facial features extracted by AO-BMF as the columns of $\mathbf{W}$ from the trial with the best relative error approximation from the binarized CBCL dataset.

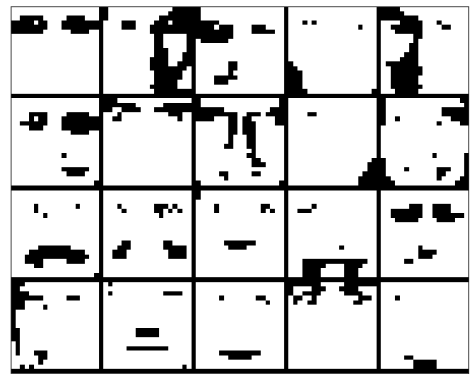

Figure 3.4: Facial features extracted by AO-BMF on the CBCL data set for $r = 20$.

We also tried applying MS-Comb-AO. For $r = 10$, due to the memory demands of the method, we were not able to gather more than two solutions. In addition, it was not able to improve the relative error of the individual AO-BMF functions used. We observed the same when we tried to use Tree BMF. Similarly, we had to constrain the number of solutions of MS-Comb-AO. In both cases when we set the depth of the tree to 1 or 2, we could not observe improvements on the initial relative errors by the AO-BMF methods. As these methods are heuristics, there is only a guarantee that they cannot worsen the initial result - there is no certainty on strictly improving.

Under the same constraints we tried to apply the MS-Comb-AO method for $r = 20$, however due to high memory demands for the combining step, it could not run on our system. As a result, we can also not test the Tree BMF method.

3.5.4.2 Greedy-Comb

We move on to our proposed greedy methods. We start with the Greedy-Comb method. The matrix obtained is the one that will be factorised, which we will refer to as $\hat{\mathbf{X}}$. To demonstrate the scalability of the method, we will test for both $r = 10$ and $r = 20$. For $r = 10$, we gathered 5 solutions. The method finished after 47.83 seconds and gave back a relative error of 36.24%. For $r = 20$, we gathered 15 solutions. The method finished after 222.487 seconds and had a relative error of 32.2%. Finally, we show the facial features extracted in Figure 3.5.

3.5.4.3 Greedy-TreeBMF

We now report on the results of our Greedy-TreeBMF method, again for both $r = 10$ and $r = 20$. We are again using $\hat{\mathbf{X}}$ as input. For $r = 10$, we are setting a maximum timelimit of 50 seconds and solve 5 Greedy-Comb problems that each have a timelimit of 10 seconds. The method finished after 55.74 seconds with a relative error of 36.22%. For $r = 20$, we are setting a maximum timelimit of 150 seconds and solve 5 Greedy-Comb problems that each have a timelimit of 30 seconds. The method finished after 159.955 seconds with a relative error of 32.09%. Finally, we show the extracted facial features in Figure 3.6.
For both greedy methods, we can see that the scalability improvement is substantial in comparison to the Gurobi-based AO-BMF. Furthermore, the facial features extracted are sensible.

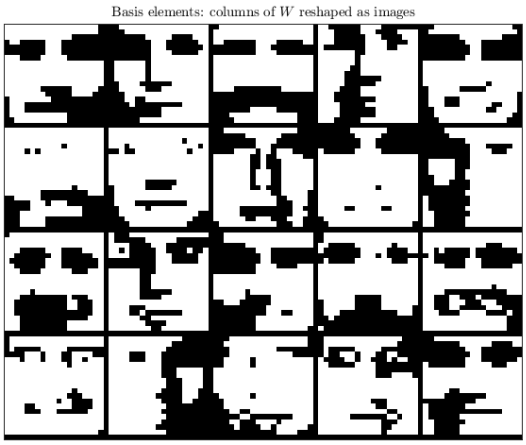

Figure 3.5: Facial features extracted by Greedy-Comb on the binarized CBCL data set $\hat{\mathbf{X}}$.

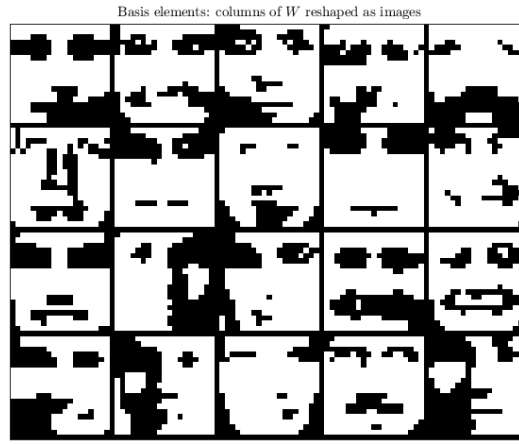

Figure 3.6: Facial features extracted by Greedy-TreeBMF on the binarized CBCL data set $\hat{\mathbf{X}}$.

3.5.4.4 Performance from competing methods

Just like with the document datasets that we tested in the previous subsection we will also be testing other methods. We initially test the ELBMF method in [33]. We are using the binarized version of the CBCL input just like in the previous two subsections. The parameters used are the following: `l1_reg` = `l2_reg` = 0.01, `maxiter` = 2000, `tol` = 10^{-10}, $r = 10$, `beta` = 10^{-4}. The regularization rate for the l_2 regularization parameter is at $\lambda = 1.0033$.
For $r = 10$, the best relative error that this method gave us was 55.74%. The average time over the 15 trials was 7.7 seconds, while the median time was 7.81 seconds. For $r = 20$, we initially tried with the same parameters as before. However we were getting worse results. Over 15 trials, the best result had a relative error of 61.51%. We changed the l_1 and l_2 regularization parameters to 0.05, as well as the regularization rate to 1.0133. This leads to an improvement in the performance of the algorithm. Over 15 trials, the average time to compute a solution is 19.36 seconds and the best solution has a relative error of 48.52%.

For ASSO, we set the threshold to 0.7. For $r = 10$, it achieved a relative error of 67.19%, after 23.45 seconds. For $r = 20$, the error was reduced to 65.62% and the algorithm finished after 36.55 seconds. We see that ASSO performs better than ELBMF but trails all our proposed methods on this dataset.

Finally, we also tried running the method in [6], for $r = 10$. Initially, we tested solely the `fast undercover` algorithm. Unfortunately, over the course of 2 days, the method could not finish. As a result, we could not test the more demanding, in terms of time and memory, `optiblock` algorithm of the paper, neither can we test for

$r = 20$.

We conclude that our methods can find sensible facial features for the CBCL dataset and furthermore, our methods perform well in terms of relative error versus the state-of-the-art.

3.6 Conclusion

In this Chapter, we have designed an alternating optimization (AO) strategy to tackle Boolean matrix factorization (BMF) using interger programming (IP). We have also shown how to combine several solutions in an optimal way, using IP as well. We showed how these two strategies are able to outperform the state of the art on 4 real-world medium-scale data sets. Furthermore, we also proposed new greedy methods that are competitive despite their simplicity. Our experiments on various real datasets and with also various other BMF algorithms display the efficiency of our methods versus the state of the art. Our tests in topic modelling show that BMF can be applied in this task and offer meaningful results. Our BMF methods also work well in the facial reconstruction task, as they can also compute interesting facial features.

We thank Sebastian Miron for sending us his BMF code [21], and Sebastian Dalleiger for providing us with good parameters for his BMF algorithm [33].

Chapter 4

Boolean Matrix Tri-Factorization

Matrix tri-factorizations (MTFs) aim to decompose an input matrix $\mathbf{X}$ into the product of three factor matrices, instead of only two as in standard matrix factorization (MF). In contrast to MF, MTF is able to cluster both rows and columns of $\mathbf{X}$ while quantifying the relationship among these two groups of clusters. In this chapter we focus on Boolean matrix tri-factorization (BMTF) that extends BMF to the tri-factorization framework. We first show an identifiability result for BMTF, namely, we show that the factors are unique under certain sparsity conditions. Then we propose an algorithm to compute the factors of BMTF, and perform numerical experiments to show how it performs on synthetic and real data. The content of this chapter is mainly from:

- C. Kolomvakis, A. Vandaele and N. Gillis, "Boolean Matrix Tri-Factorization," - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). [64]

4.1 Introduction

As an extension to matrix factorizations, matrix tri-factorization (MTF) models aim to decompose the input matrix into three factors. We use the Frobenius norm to quantify the error of the approximation.

Definition 4.1 (MTF) *Given a matrix* $\mathbf{X} \in \mathbb{R}^{m \times n}$ *and factorization ranks* (r_1, r_2)*, MTF aims to find matrices* $\mathbf{W} \in \mathbb{R}^{m \times r_1}$*,* $\mathbf{S} \in \mathbb{R}^{r_1 \times r_2}$ *and* $\mathbf{H} \in \mathbb{R}^{r_2 \times n}$ *that solve*

$$\min_{\mathbf{W} \in \mathbb{R}^{m \times r_1}, \mathbf{S} \in \mathbb{R}^{r_1 \times r_2}, \mathbf{H} \in \mathbb{R}^{r_2 \times n}} \|\mathbf{X} - \mathbf{WSH}\|_F^2.$$

Nonnegative MTF (NMTF) requires that the factors, $\mathbf{W}$, $\mathbf{S}$ and $\mathbf{H}$, are componentwise nonnegative. This leads to an easy interpretation of NMTF: the columns of $\mathbf{W}$

(resp. rows of $\mathbf{H}$) provide a soft clustering of the rows (resp. columns) of $\mathbf{X}$ into r_1 (resp. r_2) clusters: $\mathbf{W}_{ik}$ indicates the membership value of the ith row of $\mathbf{X}$ in the kth row cluster, $\mathbf{H}_{\ell j}$ indicates the membership value of the jth column of $\mathbf{X}$ in the ℓth column cluster, while $\mathbf{S}_{k\ell}$ measures the relation between the kth row cluster and the ℓth column cluster. This is an advantage of such a formulation where interaction between different clusters can be incorporated in the model.

A first variant of NMTF was explored in [36], adding orthogonality constraints, $\mathbf{W}^\top\mathbf{W} = \mathbf{I}_{r_1}$ and $\mathbf{H}^\top\mathbf{H} = \mathbf{I}_{r_2}$. Orthogonality imposes that the clusters of rows and columns are disjoint since $\mathbf{W} \geq 0$ and $\mathbf{W}^\top\mathbf{W} = \mathbf{I}_{r_1}$ implies that $\mathbf{W}$ has at most one non-zero entry per row, and similarly for the columns of $\mathbf{H}$. They proposed multiplicative update algorithms and applied them for document clustering. There are numerous other works exploring NMTF in other contexts; see, e.g., [27, 32, 92, 123] and the references therein.

In this Chapter we introduce Boolean matrix tri-factorization (BMTF) which, to the best of our knowledge, had not been explored before our contribution in [64]. A closely related factorization was proposed in [86] where $\mathbf{W}$ and $\mathbf{H}$ were restricted to be columns and rows of $\mathbf{X}$, resp., which did not consider identifiability nor interpretability aspects which are a central focus of our work. In Section 4.2, we recall the definition of BMF and formally define BMTF. In Section 4.3, we discuss the identifiability of BMTF and prove identifiability with orthogonality constraints. In Section 4.4, we describe our proposed block coordinate descent algorithm for BMTF. We also provide a refinement procedure after updating the factors $\mathbf{W}$ and $\mathbf{H}$ to generate sparser and more expressive solutions. In Section 4.5, we perform numerical experiments to assess the performance of our proposed algorithm on synthetic and real datasets. Finally, in Section 4.6 we conclude the paper with some observations and future research directions.

4.2 Boolean matrix tri-factorization

A two-factor model as BMF decomposes the data matrix $\mathbf{X}$ as the sum of r communities, $\mathbf{W}(:,k)\mathbf{H}(k,:)$ for $k = 1, 2, \ldots, r$, where a community is made of a cluster of rows and a cluster of columns. The kth cluster of rows, defined as $\{i \mid \mathbf{W}(i,k) = 1\}$, can only interact with the kth cluster of columns, $\{j \mid \mathbf{H}(k,j) = 1\}$, and the number of cluster of rows and columns must be the same. A three-factorization model allows for any interactions between these clusters, and for a different number of clusters in both dimensions. Let us define BMTF formally.

Definition 4.2 (BMTF) *Given a Boolean matrix $\mathbf{X} \in \{0,1\}^{m\times n}$ and factorization ranks (r_1, r_2), BMTF aims to find $\mathbf{W} \in \{0,1\}^{m\times r_1}$, $\mathbf{S} \in \{0,1\}^{r_1\times r_2}$ and $\mathbf{H} \in \{0,1\}^{r_2\times n}$ that minimize $\|\mathbf{X} - \mathbf{W} \circ \mathbf{S} \circ \mathbf{H}\|_F^2$.*

BMTF is able to detect

- r_1 clusters[1] for the rows of $\mathbf{X}$, defined as $\{i \mid \mathbf{W}(i,k) = 1\}$ for $k = 1, 2, \ldots, r_1$,

[1] Note that the clusters do not need to be disjoint.

- r_2 clusters for the columns of $\mathbf{X}$, defined as $\{j \mid \mathbf{H}(\ell, j) = 1\}$ for $\ell = 1, 2, \ldots, r_2$,
- the interactions between these clusters via the matrix $\mathbf{S}$ since $\mathbf{X} \approx \sum_{k=1}^{r_1} \sum_{\ell=1}^{r_2} \mathbf{W}(:, k)\mathbf{S}(k, \ell)\mathbf{H}(\ell, :)$.

For example, let $\mathbf{X}$ be a data set where the rows correspond to animals and the columns represent characteristics (e.g., 'has fins', 'flies', 'has 4 legs'), while $\mathbf{X}(i, j) = 1$ if animal i has the characteristic j. BMTF can not only find clusters of animals (in $\mathbf{W}$) and characteristics (in $\mathbf{H}$), but also it can link the two sets of clusters through the factor $\mathbf{S}$; e.g., a cluster of fishes to a cluster of their characteristics such as 'aquatic', and 'has fins'; see Section 4.5.2 for a real-world example. In the next two sections, we discuss the identifiability of BMTF, and then we propose an algorithm to compute solutions to BMTF.

4.3 Identifiability via Orthogonality

A BMTF, $\mathbf{X}{=}\mathbf{W}{\circ}\mathbf{S}{\circ}\mathbf{H}$, is identifiable/unique if any other BMTF of $\mathbf{X}$ can only be obtained via permutations, that is, for any other BMTF $\mathbf{X}{=}\mathbf{W}'{\circ}\mathbf{S}'{\circ}\mathbf{H}'$ of the same size, we have $\mathbf{W}'{=}\mathbf{W}(:, \pi_1)$, $\mathbf{S}'{=}\mathbf{S}(\pi_1, \pi_2)$, and $\mathbf{H}'{=}\mathbf{H}(\pi_2, :)$, for some permutations π_1 of $\{1, 2, \ldots, r_1\}$ and π_2 of $\{1, 2, \ldots, r_2\}$.

It is crucial to note that when $r_1 \neq r_2$, plain BMTF is never identifiable (that is, BMTF without additional constraints). Assume w.l.o.g. that $r_1 > r_2$ and let $\mathbf{X} = \mathbf{W} \circ \mathbf{S} \circ \mathbf{H}$ be a BMTF. Let us show that we can always construct another BMTF of $\mathbf{X}$ which cannot be obtained as a permutation of $\mathbf{W} \circ \mathbf{S} \circ \mathbf{H}$. There are two cases:

- If a column of $\mathbf{W}$ is equal to zero, say $\mathbf{W}(:, k) = 0$, then the corresponding row of $\mathbf{S}$, $\mathbf{S}(k, :)$, can take any value and the BMTF is not identifiable.
- Otherwise, another BMTF is given by

$$\mathbf{W}' = [\mathbf{W} \circ \mathbf{S}, \mathbf{0}_{m \times (r_1 - r_2)}], \mathbf{S}' = [\mathbf{I}_{r_2}; \mathbf{0}_{(r_1 - r_2) \times r_2}], \mathbf{H}' = \mathbf{H},$$

  where $\mathbf{0}_{a \times b}$ is the a-by-b all-zero matrix, so that $\mathbf{W}'$ has $r_1 - r_2$ zero columns.

Another way to make this observation is to realize that BMTF is an overparametrized BMF, since $\mathbf{W} \circ \mathbf{S} \circ \mathbf{H} = (\mathbf{W} \circ \mathbf{S}) \circ \mathbf{H} = \mathbf{W} \circ (\mathbf{S} \circ \mathbf{H})$. Hence, to be able to provide additional insight on the data set and to be identifiable, BMTF requires additional constraints. For example, a natural goal would be to look for the sparsest $\mathbf{W}$ and $\mathbf{H}$ to identify the smallest clusters that explain the data. Let us prove that BMTF is identifiable under the conditions that the clusters are disjoint, or equivalently that the columns of $\mathbf{W}$ (resp. rows of $\mathbf{H}$) are orthogonal. Before proving this result, let us provide a definition and a lemma.

Definition 4.3 (Orthogonal BMTF) *Orthogonal BMTF is the BMTF problem with the additional constraints that* $\mathbf{W}(:, i)^\top \mathbf{W}(:, j) = 0$ *for all* $i \neq j$ *and* $\mathbf{H}(k, :)\mathbf{H}(p, :)^\top = 0$ *for all* $k \neq p$.

Lemma 4.1 *Let* $\mathbf{S} \in \{0,1\}^{r_1 \times r_2}$ *have distinct non-zero rows and columns. Then the unique exact orthogonal BMTF of* $\mathbf{S}$ *with ranks* (r_1, r_2) *is* $\mathbf{I}_{r_1} \circ \mathbf{S} \circ \mathbf{I}_{r_2}$*, up to permutations.*

Proof 4.1 *Let* $\mathbf{S}{=}\mathbf{W}{\circ}\mathbf{S}{\circ}\mathbf{H}$ *be an orthogonal BMTF of* $\mathbf{S}$*. Because* $\mathbf{S}$ *has no zero columns, and* $\mathbf{H}$ *has at most a single non-zero entry per column (due to orthogonality), each column of* $\mathbf{S}$ *is equal to a column of* $\mathbf{W}{\circ}\mathbf{S}'$*. Moreover, since the columns of* $\mathbf{S}$ *are distinct and non-zero, and* $\mathbf{W}{\circ}\mathbf{S}'$ *has* r_2 *columns,* $\mathbf{H}$ *must be a permutation of the identity. Using the same argument on the rows, we conclude that* $\mathbf{W}$ *must be a permutation of the identity.*

Theorem 4.1 *Let* $\mathbf{X} = \mathbf{W} \circ \mathbf{S} \circ \mathbf{H}$ *be an orthogonal BMTF with ranks* (r_1, r_2) *where each column of* $\mathbf{W}$ *and* $\mathbf{H}^\top$ *have a least one non-zero element (no cluster is empty), and* $\mathbf{S} \in \{0,1\}^{r_1 \times r_2}$ *has distinct non-zero rows and columns. Then* $\mathbf{X}$ *has a unique orthogonal BMTF.*

Proof 4.2 *The uniqueness of* $\mathbf{W}$ *and* $\mathbf{H}$ *follows from the uniqueness of ONMF [46, Th 4.40, p.136]. Let us recall this result: if* $\mathbf{X} = \mathbf{AB}$ *where* $\mathbf{B} \geq \mathbf{0}$ *has orthogonal rows, and* $\mathbf{A}$ *has non-multiple columns (that is,* $\mathbf{A}(:,\mathbf{i}) \neq \alpha\mathbf{A}(:,j)$ *for any* $\alpha \in \mathbb{R}$*, for* $i \neq j$*), then* $(\mathbf{A}, \mathbf{B})$ *is an identifiable ONMF. Since* $\mathbf{W}$ *and* $\mathbf{H}^\top$ *has orthogonal columns, we have* $\mathbf{W} \circ \mathbf{S} \circ \mathbf{H} = \mathbf{WSH}$*. Applying the uniqueness result of ONMF to* $(\mathbf{WS})\mathbf{H}$ *and* $\mathbf{W}(\mathbf{SH})$*, we have that* $\mathbf{H}$ *is unique if* $\mathbf{WS}$ *has non-multiple columns and* $\mathbf{W}$ *is unique if* $\mathbf{SH}$ *has non-multiple rows. Since* $\mathbf{W}$ *has no zero column and is binary,* $\mathbf{W}$ *contains the identity matrix, hence the matrix* $\mathbf{S}$ *appears as a submatrix of* $\mathbf{WS}$*. Since* $\mathbf{S}$ *has distinct columns,* $\mathbf{WS}$ *also has. The same argument applies to* $\mathbf{SH}$*.*

Finally, since $\mathbf{W}$ *and* $\mathbf{H}$ *are unique and contain the identity as a submatrix,* $\mathbf{S}$ *has to be unique since* $\mathbf{I}_{r_1}\mathbf{S}\mathbf{I}_{r_2}$ *is the only orthogonal BMTF of* $\mathbf{S}$*; see Lemma 4.1.*

4.4 Block-coordinate descent for BMTF

In this section, we propose a block-coordinate descent (BCD) method to solve BMTF. It relies on our previous work that proposed a BCD scheme for BMF, as in chapter 3.

4.4.1 Introduction

The scheme is a standard approach for matrix and tensor factorizations: optimize over each factor individually, in our case $\mathbf{W}$, $\mathbf{S}$ and then $\mathbf{H}$, while the others are fixed. To optimize $\mathbf{W}$, we need to solve m independent Boolean least squares (BoolLS) problem:

$$\min_{\mathbf{W}(i,:)\in\{0,1\}^{r_1}} \|\mathbf{X}(i,:) - \mathbf{W}(i,:)(\mathbf{S} \circ \mathbf{H})\|_2^2. \tag{4.1}$$

Each subproblem has only r_1 binary variables and can be solved relatively fast using an integer programming software. For $\mathbf{H}$, we need to solve n BoolLS in r_2 variables, one for each column of $\mathbf{H}$. These problems are solved using Gurobi [53] but, in the worst case, the complexity for updating both factors once is $O(m2^{r_1} + n2^{r_2})$ (by brute

force). The update of $\mathbf{S}$ needs to be handled slightly differently. Using the property $\text{vec}(\mathbf{ABC}) = (\mathbf{C}^\top \otimes \mathbf{A})\,\text{vec}(\mathbf{B})$, where $\otimes$ is the Kronecker product and vec vectorizes a matrix as a vector column wise, the problem in $\mathbf{S}$ can written as a BoolLS in $r_1 r_2$ variables:

$$\min_{\mathbf{S}\in\{0,1\}^{r_1\times r_2}} \|\,\text{vec}(\mathbf{X}) - \min\left(1, (\mathbf{H}^\top \otimes \mathbf{W})\,\text{vec}(\mathbf{S})\right)\|_2^2, \tag{4.2}$$

with worst-case complexity $O(2^{r_1 r_2})$. We then reshape the retrieved vector to obtain $\mathbf{S}$.

4.4.2 Imposing sparsity

As explained in Section 4.3, BMTF is in general not identifiable unless additional constraints are imposed. We consider sparsity constraints, and generate sparse solutions by adding explicit constraints on the rows of $\mathbf{W}$ (resp. columns of $\mathbf{H}$) when solving (4.1): for all i, $\sum_{k=1}^{r_1} \mathbf{W}(i,k) \leq K_W$, where $1 \leq K_W \leq r_1$ is a sparsity parameter, and similarly for $\mathbf{H}$. For example, setting K_W=1 corresponds to the orthogonality constraint discussed in Section 4.3: each row of $\mathbf{X}$ belongs to at most one cluster. Sparsity is a natural constraint as it corresponds to identifying the smallest clusters that explain the data.

4.4.3 Generating sparser and more expressive solutions

When $r_1 > r_2$, we have observed that BCD often generates $\mathbf{W}$'s with zero columns, because of the identifiability issues discussed in Section 4.3. In order to generate sparser solutions and avoid rank-deficient ones, we resort to a refinement procedure that will generate sparser and more expressive solutions. This procedure is inspired by [47], and has been used recently in [50] for tensor factorizations. Let $\mathbf{W}\in\{0,1\}^{m\times r_1}$, and assume that the support of $\mathbf{W}(:,i)$ (that is, the set of indices corresponding to non-zero entries) contains that of $\mathbf{W}(:,j)$. Then we construct $(\mathbf{W}', \mathbf{S}')$ such that $\mathbf{W}'\circ\mathbf{S}'=\mathbf{W}\circ\mathbf{S}$ and $\mathbf{W}'$ is sparser than $\mathbf{W}$: $\mathbf{W}'(:,i)=\mathbf{W}(:,i)-\mathbf{W}(:,j)$, $\mathbf{S}'(i,:)=\mathbf{S}(i,:) \vee \mathbf{S}(j,:)$, and the other columns of $\mathbf{W}'$ (resp. rows of $\mathbf{S}'$) are equal to that of $\mathbf{W}$ (resp. $\mathbf{S}$). Using this observation, the lemma below follows.

Lemma 4.2 *Let $\mathbf{W} \in \{0,1\}^{m\times r_1}$ have distinct columns. Let $\mathbf{P} \in \{-1,0,1\}^{r_1\times r_1}$ and $\mathbf{Q} \in \{0,1\}^{r_1\times r_1}$ be as follows:*
(1) $\mathbf{P}(i,i) = \mathbf{Q}(i,i) = 1$ for all i, and
(2) $\mathbf{P}(j,i) = -1$ and $\mathbf{Q}(j,i) = 1$ for all $i \neq j$ such that the support of $\mathbf{W}(:,i)$ contains that of $\mathbf{W}(:,j)$.
Then $\mathbf{W}=\mathbf{W}'\circ\mathbf{Q}$, where $\mathbf{W}'=\max(0,\mathbf{WP}) \in \{0,1\}^{m\times r_1}$ is sparser than $\mathbf{W}$ (since $\mathbf{P}$ has negative entries when there is an inclusion between the supports of some columns of W) and more expressive[2] in BMTF since $\mathbf{W}\circ\mathbf{S} = \mathbf{W}'\circ(\mathbf{Q}\circ\mathbf{S}) = \mathbf{W}'\circ\mathbf{S}'$ for any $\mathbf{S}$ and for $\mathbf{S}' = \mathbf{Q}\circ\mathbf{S}$.

[2]That is, any BMTF generated using $\mathbf{W}$ can also be generated using $\mathbf{W}'$.

Note that if $\mathbf{W}$ has two identical columns, one of them can be set to zero, and $\mathbf{S}$ can be updated accordingly without changing $\mathbf{W} \circ \mathbf{S}$. When $\mathbf{W}$ has zero columns (and similarly for rows of $\mathbf{H}$), we set a single entry to 1, at the position where the row of the residual $\mathbf{R} = \mathbf{X} - \mathbf{W} \circ \mathbf{S} \circ \mathbf{H}$ has the most entries equal to one.

Example of Lemma 4.2. Let $m = 4$ and $r_1 = 3$, and consider

$$\mathbf{W} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \{0,1\}^{4\times 3},$$

whose columns are distinct and satisfy $\operatorname{supp}(\mathbf{W}(:,1)) = \{1,2,3\} \supset \operatorname{supp}(\mathbf{W}(:,2)) = \{1,2\}$. Following Lemma 4.2, define $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{3\times 3}$ by $\mathbf{P}(i,i) = \mathbf{Q}(i,i) = 1$ for all i, and since $\operatorname{supp}(\mathbf{W}(:,1))$ contains $\operatorname{supp}(\mathbf{W}(:,2))$, set $\mathbf{P}(2,1) = -1$ and $\mathbf{Q}(2,1) = 1$ (all other off-diagonal entries are 0). Thus

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{Q} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Then $\mathbf{WP}$ has first column $\mathbf{W}(:,1) - \mathbf{W}(:,2)$, hence

$$\mathbf{W}' = \max(0, \mathbf{WP}) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \{0,1\}^{4\times 3}.$$

In particular, $\|\mathbf{W}(:,1)\|_0 = 3$ whereas $\|\mathbf{W}'(:,1)\|_0 = 1$, so $\|\mathbf{W}'\|_0 < \|\mathbf{W}\|_0$ and $\mathbf{W}'$ is strictly sparser than $\mathbf{W}$. Moreover, we can check that $\mathbf{W} = \mathbf{W}' \circ \mathbf{Q}$ since $\mathbf{Q}(:,1) = (1,1,0)^\top$ implies $(\mathbf{W}' \circ \mathbf{Q})(:,1) = \mathbf{W}'(:,1) \vee \mathbf{W}'(:,2) = \mathbf{W}(:,1)$, while the other columns are unchanged.
To summarize, we propose the following **refinement procedure**:

1. set to zero duplicated columns of $\mathbf{W}$,
2. make the $\mathbf{W}$ sparser and more expressive via Lemma 4.2,
3. reinitialize zero columns with a single non-zero entry as described above.

4.4.4 Initialization of the algorithm

We initialize the matrix $\mathbf{W}$ either by randomly sampling r_1 columns of $\mathbf{X}$ or by binarizing an NMF solution of $\mathbf{X}$, as in [63]. W.l.o.g. we assume $r_1 \geq r_2$ (otherwise, transpose $\mathbf{X}$), and the matrix $\mathbf{S}$ is initialized with the identity matrix $\mathbf{I}_{r_2}$ while the remaining rows are filled with a single 1 in random positions. Finally, here is our proposed algorithm:

The above steps ensure that the BCD scheme generates a sequence of solutions with non-increasing objective function.

Algorithm 15 BCD algorithm for BMTF

Require: $\mathbf{X} \in \{0,1\}^{m\times n}$, ranks (r_1, r_2), sparsity parameters K_W, K_H.
Ensure: $\mathbf{W} \in \{0,1\}^{m\times r_1}$, $\mathbf{S} \in \{0,1\}^{r_1\times r_2}$ and $\mathbf{H} \in \{0,1\}^{r_2\times n}$

1: Initialize $\mathbf{W}\in\{0,1\}^{m\times r_1}$ and $\mathbf{S}\in\{0,1\}^{r_1\times r_2}$ as described above.
2: **while** $\mathbf{W}$, $\mathbf{S}$ or $\mathbf{H}$ change **do**
3: Update $\mathbf{H}$ by column with BoolLS with sparsity K_H, and perform the refinement procedure.
4: Update $\mathbf{S}$ by solving the BoolLS (4.2).
5: Update $\mathbf{W}$ by row (4.1) with sparsity K_W, and perform the refinement procedure.
6: Update $\mathbf{S}$ by solving the BoolLS (4.2) (because $\mathbf{W}$ has changed).
7: $i = i + 1$
8: **end while**
9: **return** $(\mathbf{W}, \mathbf{S}, \mathbf{H})$

4.5 Numerical Experiments

All experiments in this section are performed on Julia v.1.9.2 with Gurobi version 11 on a laptop with an intel i7 1255U processor @ 1.7 GHz and 16 GB RAM. The code is available in `https://gitlab.com/ckolomvakis/boolean-matrix-tri-factorization`.

4.5.1 Experiments on synthetic data

We first construct an orthogonal BMTF as $\mathbf{X}=\mathbf{W}\circ\mathbf{S}\circ\mathbf{H}\in\{0,1\}^{120\times 80}$ where $\mathbf{W} \in \{0,1\}^{120\times 7}$ has one non-zero element per row. Each column of $\mathbf{W}$ has $17=\lfloor 120/7\rfloor$ non-zero elements, and the last one has 18. The same procedure is used to generate the rows of $\mathbf{H}$. For $\mathbf{S}$, each row has 2 non-zero elements: We list all the possible binary vectors of dimension $5(= r_2)$ with 2 entries equal to one, and pick $7(= r_1)$ of them randomly as the rows of $\mathbf{S}$. Then, to make the problem more challenging, we add to $\mathbf{W}$ one nonzero per row at a random position, and then one nonzero per column of $\mathbf{H}$. We denote (z_W, z_S, z_H) the number of non-zeros per row of $\mathbf{W}$, per row of $\mathbf{S}$ and per column of $\mathbf{H}$. For each experimental setting, we run our algorithm with 25 random initializations (we use the NMF-based initialization for $\mathbf{W}$) and keep the best solution (unless it finds a solution with zero error in which case we stop it), and we repeat this procedure 50 times (50 trials).

Table 4.1 reports the percentage of times the ground truth $\mathbf{W}$, $\mathbf{S}$ and $\mathbf{H}$ were found correctly among the 50 runs, and in parenthesis the percentage of elements found wrong on average. The fifth column (err.) reports the percentage of runs when the average relative error $\frac{\|\mathbf{X}-\mathbf{W}\circ\mathbf{S}\circ\mathbf{H}\|_F}{\|\mathbf{X}\|_F}$ is equal to zero, and in brackets it reports its value. The last column reports the average time in seconds needed to generate one BMTF solution with one initialization.

For the case where $\mathbf{W}$ and $\mathbf{H}$ are orthogonal (that is, $z_W{=}z_H{=}1$, first row), the ground-truth factors are always recovered (on an average using less than 7 initializa-

(z_W, z_S, z_H)	**W**	**S**	**H**	err.	time
(1,2,1)	100%	100%	100%	100%	1.13 s.
(2,2,1)	0% (21%)	0% (16%)	99.8% (0.16‰)	45% (9%)	1.43 s.
(2,2,2)	0% (24%)	0% (24%)	0% (26%)	0.7% (28%)	1.32 s.

Table 4.1: Percentage of time the ground-truth factors, or zero errors, are found among the 50 Monte Carlo runs on synthetic data sets. In brackets, we report the percentage of entries found wrong on average for the factors **W**, **S** and **H**, and report the average relative error.

tions). This illustrates two facts: our identifiability results (Theorem 4.1) and the fact that our algorithm performs well as it is able to recover the unique solution in all 50 trials. For the case **W** is not orthogonal but **H** is (second row), the algorithm can still find the ground-truth **H** in most cases, although it cannot recover **W** and **S**, which are non-unique: in fact, in 45% of the trials, BCD finds a solution with a zero relative error but the groundtruth **W** and **S** are not recovered, because **W** is not orthogonal. The problem becomes even harder when we add the extra nonzeros on **H** as well (third row). However, although we cannot recover the ground truth, the recovered factors share many entries with it (about 75%).

4.5.2 Clustering a real dataset with BMTF

Let us consider an experiment on the “zoo" real dataset [39]. Each row represents an animal, and each column represents a characteristic; see Tables 4.2-4.3 for examples. We have removed some characteristics and some animals from the dataset as done in [119] (e.g., the characteristic ‘breathes’ which is equal to one for almost all animals) to obtain a matrix $\mathbf{X} \in \{0,1\}^{99\times 14}$.

Here, we are considering $(r_1, r_2) = (5, 3)$. Over 1500 trials, each of which took on average 0.58 seconds, we report the factors with the lowest error $\|\mathbf{X} - \mathbf{W} \circ \mathbf{S} \circ \mathbf{H}\|_F / \|\mathbf{X}\|_F = 33\%$, which is relatively high since the ranks chosen are small ($r_2 = 3$) and real data do not perfectly fit low-rank models. The parameters considered for **W** and **H** were $(K_W, K_H) = (2, 2)$. Tables 4.2-4.3 show the clusters. We call 'mixed cluster' the clusters were multiple types of animals are included.
We will now present the matchings that we receive from the matrix **S**:

W clusters \ **H** clusters	Birds	Mixed	Mammals
Aquatic animals	0	1	0
Birds	1	0	0
Mixed cluster	0	0	0
Aquatic birds	1	1	0
Mammals	0	0	1

.

Aquatic animals: bass, carp, catfish, chub, crayfish, dogfish, ..., octopus, penguin, pike, ..., stingray, tuna
Birds: chicken, crow, dove, duck, flamingo, gnat, ..., parakeet, penguin, pheasant, ..., vulture, wasp, wren
Mixed cluster: crab, crayfish, flea, frog, fruitbat, gnat, gorilla, honeybee, ..., wasp
Aquatic birds: gull, skimmer, skua
Mammals: aardvark, antelope, bear, boar, buffalo, calf, ..., vole, wallaby, wolf

Table 4.2: Clusters of animals.

Avian (rel. to birds): feathers, eggs, airborne, less than 4 legs
Mixed charactetistics: eggs, aquatic, predator, toothed, fins, less than 4 legs, tail
Mammalian: hair, milk, toothed, 4 legs, tail

Table 4.3: Clusters of characteristics.

We observe that the clusters make sense. Furthermore, many animals are assigned to multiple clusters per the property of BMTF, e.g., the "penguin" is assigned to aquatic animals and birds, both being sensible assignments.

4.6 Conclusion

In this chapter, we introduced Boolean matrix tri-factorization (BMTF). We gave motivations as to why this model is useful, discussed identifiability, and proposed a BCD algorithm that includes a clever refinement procedure. Our numerical experiments show that the BCD algorithm is able to find good solutions and can be used meaningfully on a real-world data set. Further work includes a deeper understanding of the conditions under which BMTF is identifiable, as done for example for BMF in [35, 90], the use of BMTF for other applications, and the development of algorithms scalable to large data.

Chapter 5

Conclusions

In this chapter, we summarize the contributions of this thesis and propose possible future research directions.

5.1 Summary of the contributions

This thesis mostly focused on the design of algorithms for matrix factorization models with binary and Boolean constraints: We proposed new algorithms for semi-bMF, BMF, and BMTF.

More specifically:

- In Chapter 2, we proposed new algorithms for semi-bMF, which imposes only one factor to be binary. We extended the algorithms of [70, 71] to handle noisy data, as well as algorithms that are scalable. More specifically, we explained how the robust versions of ASCD and ABCD can be obtained. We then presented a first-order algorithm for the SDPs involved, to make the algorithms more scalable, and later made it even more scalable by using the Burer-Monteiro approach [16, 29]. We explained how our new algorithms work and presented the scalability gains.
- In Chapter 3, we shifted our focus to BMF. We introduced new algorithms with different methodologies. We first presented alternating optimization algorithms that solve integer programs as part of the computation of a BMF, and combining schemes to improve the solutions. To address the problem of scalability that comes with the use of integer programming methods, we then proposed greedy and local search heuristics. We showed that our latter class of proposed algorithms is competitive with the state of the art. We also showed the considerable scalability gains. We examined various applications, including facial image reconstruction and topic modeling.

- In Chapter 4, we proposed the new Boolean Matrix Tri-Factorization (BMTF) model. We first discussed why it is interesting to consider it and what additional interpretation it can give us in comparison to BMF. We proved conditions under which BMTF is identifiable, and then proposed an IP-based alternating optimization algorithm. Our experiments confirm that, under the proposed conditions, our algorithm can recover the ground truth. Furthermore, we showcase that in a classifying application of animals and their characteristics, it offers meaningful results.

5.2 Future Work

In Chapter 2, we already discussed some potential future work directions. Additional research directions could be the following:

- There is limited work on metaheuristics like genetic algorithms, simulated annealing or TABU search for BMF. It could be interesting to explore these alternatives further, especially since we have shown in Chapter 3 that simple greedy heuristics already perform very well.
- Designing branch-and-bound algorithms specifically designed for the Boolean least squares problem in an attempt to improve the performance of the IP-based algorithms we proposed.
- Examining algorithms that compute globally optimal solutions of BMF, without solving for one factor at a time.
- For BMTF, while we proposed conditions for the uniqueness of the decomposition, these are rather restrictive. Examining whether more relaxed conditions can guarantee the uniqueness of BMTF would be particularly interesting.
- Develop greedy algorithms for BMTF, since they perform well for BMF, while our current IP-based algorithms are limited for small and medium sized datasets.

Bibliography

1. Ang, A. M. S. & Gillis, N. Accelerating Nonnegative Matrix Factorization Algorithms Using Extrapolation. *Neural Computation* **31,** 417–439 (2019).
2. ApS, M. *The MOSEK optimization toolbox for MATLAB manual. Version 10.0.* (2022). `http://docs.mosek.com/9.0/toolbox/index.html`.
3. Araujo, M., Ribeiro, P. & Faloutsos, C. *FastStep: Scalable Boolean Matrix Decomposition* in *Pacific-Asia Conf. on Knowledge Discovery and Data Mining* (2016).
4. Asteris, M., Papailiopoulos, D. & Dimakis, A. G. *Orthogonal NMF through Subspace Exploration* in *Advances in Neural Information Processing Systems* (eds Cortes, C., Lawrence, N., Lee, D., Sugiyama, M. & Garnett, R.) **28** (Curran Associates, Inc., 2015). `https://proceedings.neurips.cc/paper_files/paper/2015/file/eae27d77ca20db309e056e3d2dcd7d69-Paper.pdf`.
5. Avellaneda, F. & Villemaire, R. Delegation-Relegation for Boolean Matrix Factorization. *Proceedings of the AAAI Conference on Artificial Intelligence* **38,** 20632–20639. `https://ojs.aaai.org/index.php/AAAI/article/view/30049` (Mar. 2024).
6. Avellaneda, F. & Villemaire, R. Undercover Boolean Matrix Factorization with MaxSAT. *Proceedings of the AAAI Conference on Artificial Intelligence* **36,** 3672–3681. `https://ojs.aaai.org/index.php/AAAI/article/view/20280` (June 2022).
7. Ban, F., Bhattiprolu, V., Bringmann, K., Kolev, P., Lee, E. & Woodruff, D. P. *A PTAS for lp-low rank approximation* in (Society for Industrial and Applied Mathematics, San Diego, California, 2019), 747–766.
8. Belohlavek, R. & Trnecka, M. A new algorithm for Boolean matrix factorization which admits overcovering. *Discrete Applied Mathematics* **249.** Concept Lattices and Applications: Recent Advances and New Opportunities, 36–52. ISSN: 0166-218X. `https://www.sciencedirect.com/science/article/pii/S0166218X18303755` (2018).
9. Belohlavek, R. & Trnecka, M. The 8M Algorithm from Today's Perspective. *ACM Trans. Knowl. Discov. Data* **15.** ISSN: 1556-4681. `https://doi.org/10.1145/3428078` (Jan. 2021).

10. Belohlavek, R. & Vychodil, V. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computer and System Sciences* **76.** Special Issue on Intelligent Data Analysis, 3–20. ISSN: 0022-0000. https://www.sciencedirect.com/science/article/pii/S0022000009000415 (2010).

11. Bhattacharya, A., Goyal, D., Jaiswal, R. & Kumar, A. Streaming PTAS for Binary ℓ_0-Low Rank Approximation. *arXiv e-prints,* arXiv:1909.11744. arXiv: 1909.11744 [cs.DS] (Sept. 2019).

12. Bishop, C. M. & Nasrabadi, N. M. *Pattern recognition and machine learning* **4** (Springer, 2006).

13. Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., *et al.* An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software* **28,** 135–151 (2002).

14. Bonnans, J. F. & Shapiro, A. *Perturbation analysis of optimization problems* (Springer Science & Business Media, 2013).

15. Boumal, N., Voroninski, V. & Bandeira, A. *The non-convex Burer-Monteiro approach works on smooth semidefinite programs* in *Advances in Neural Information Processing Systems* (eds Lee, D., Sugiyama, M., Luxburg, U., Guyon, I. & Garnett, R.) **29** (Curran Associates, Inc., 2016).

16. Boumal, N., Voroninski, V. & Bandeira, A. S. *The non-convex Burer–Monteiro approach works on smooth semidefinite programs* in *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Curran Associates Inc., Barcelona, Spain, 2016), 2765–2773. ISBN: 9781510838819.

17. Boyle, J. P. & Dykstra, R. L. *A Method for Finding Projections onto the Intersection of Convex Sets in Hilbert Spaces* in *Advances in Order Restricted Statistical Inference* (eds Dykstra, R., Robertson, T. & Wright, F. T.) (Springer New York, New York, NY, 1986), 28–47. ISBN: 978-1-4613-9940-7.

18. Bringmann, K., Kolev, P. & Woodruff, D. *Approximation Algorithms for \l_ 0-Low Rank Approximation* in *Advances in Neural Information Processing Systems* (eds Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. & Garnett, R.) **30** (Curran Associates, Inc., 2017).

19. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. *Language models are few-shot learners* in *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Curran Associates Inc., Vancouver, BC, Canada, 2020). ISBN: 9781713829546.

20. Burer, S. & Monteiro, R. D. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming* **95,** 329–357 (2003).

21. Cabral Farias, R. & Miron, S. A generalized approach for Boolean matrix factorization. *Signal Processing* **206,** 108887. ISSN: 0165-1684. https://www.sciencedirect.com/science/article/pii/S0165168422004261 (2023).

22. Cai, D., Mei, Q., Han, J. & Zhai, C. *Modeling hidden topics on document manifold* in *Proceedings of the 17th ACM Conference on Information and Knowledge Management* (Association for Computing Machinery, Napa Valley, California, USA, 2008), 911–920. ISBN: 9781595939913. https://doi.org/10.1145/1458082.1458202.

23. Campbell, M. J. BMDP Statistical Software. *Bioinformatics* **4,** 427–430. ISSN: 1367-4803. eprint: https://academic.oup.com/bioinformatics/article-pdf/4/3/427/548262/4-3-427.pdf (Aug. 1988).

24. Candes, E. J. & Plan, Y. Matrix completion with noise. *Proceedings of the IEEE* **98,** 925–936 (2010).

25. Candès, E. J., Li, X., Ma, Y. & Wright, J. Robust principal component analysis? *Journal of the ACM (JACM)* **58,** 1–37 (2011).

26. Carpineto, C. & Romano, G. *Concept Data Analysis: Theory and Applications* ISBN: 0470850558 (John Wiley & Sons, Inc., Hoboken, NJ, USA, 2004).

27. Chen, G., Wang, F. & Zhang, C. Collaborative filtering using orthogonal nonnegative matrix tri-factorization. *Information Processing & Management* **45,** 368–379. ISSN: 0306-4573. https://www.sciencedirect.com/science/article/pii/S0306457308001167 (2009).

28. Cichocki, A., Zdunek, R. & Amari, S.-i. *Hierarchical ALS Algorithms for Nonnegative Matrix and 3D Tensor Factorization* in *Independent Component Analysis and Signal Separation* (eds Davies, M. E., James, C. J., Abdallah, S. A. & Plumbley, M. D.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2007), 169–176. ISBN: 978-3-540-74494-8.

29. Cifuentes, D., Boumal, N., Granda, M., C., L. & S., L. On the Burer–Monteiro method for general semidefinite programs. *Optimization Letters* **15,** 2299–2309 (2021).

30. Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. *Introduction to Algorithms, Third Edition* 3rd. ISBN: 0262033844 (The MIT Press, 2009).

31. Cplex, I. I. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation* **46,** 157 (2009).

32. Dache, A., Vandaele, A. & Gillis, N. *Orthogonal Symmetric Nonnegative Matrix Tri-Factorization* in *34th IEEE International Workshop on Machine Learning for Signal Processing (MLSP)* (2024).

33. Dalleiger, S. & Vreeken, J. Efficiently Factorizing Boolean Matrices using Proximal Gradient Descent. *Advances in Neural Information Processing Systems* (2022).

34. Dalleiger, S., Vreeken, J. & Kamp, M. Federated Binary Matrix Factorization Using Proximal Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence* **39,** 16144–16152. https://ojs.aaai.org/index.php/AAAI/article/view/33773 (Apr. 2025).

35. Desantis, D., Skau, E., Truong, D. P. & Alexandrov, B. Factorization of Binary Matrices: Rank Relations, Uniqueness and Model Selection of Boolean Decomposition. *ACM Trans. Knowl. Discov. Data* **16.** ISSN: 1556-4681. https://doi.org/10.1145/3522594 (2022).

36. Ding, C., Li, T., Peng, W. & Park, H. *Orthogonal nonnegative matrix t-factorizations for clustering* in *12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (2006), 126–135.

37. Dorffer, C., Puigt, M., Delmaire, G. & Roussel, G. *Fast nonnegative matrix factorization and completion using Nesterov iterations* in *International Conference on Latent Variable Analysis and Signal Separation* (2017), 26–35.

38. Drakopoulos, G. & Mylonas, P. *A Genetic Algorithm For Boolean Semiring Matrix Factorization With Applications To Graph Mining* in *2022 IEEE International Conference on Big Data (Big Data)* (2022), 3864–3870.

39. Dua, D. & Graff, C. *UCI Machine Learning Repository* 2017. http://archive.ics.uci.edu/ml.

40. Erdos, D. & Miettinen, P. *Walk 'n' Merge: A Scalable Algorithm for Boolean Tensor Factorization* in *2013 IEEE 13th International Conference on Data Mining* (2013), 1037–1042.

41. Farias, R. C. & Miron, S. *Projected Hierarchical ALS for Generalized Boolean Matrix Factorization* in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2023), 1–5.

42. Fomin, F. V., Golovach, P. A., Lokshtanov, D., Panolan, F. & Saurabh, S. Approximation Schemes for Low-rank Binary Matrix Approximation Problems. **16.** ISSN: 1549-6325. https://doi.org/10.1145/3365653 (Nov. 2019).

43. Fomin, F. V., Golovach, P. A. & Panolan, F. Parameterized low-rank binary matrix approximation. *Data Min. Knowl. Discov.* **34,** 478–532. ISSN: 1384-5810. https://doi.org/10.1007/s10618-019-00669-5 (Mar. 2020).

44. Ganter, B. & Wille, R. *Formal concept analysis : mathematical foundations* ISBN: 3540627715 9783540627715 (Springer, Berlin; New York, 1999).

45. Geerts, F., Goethals, B. & Mielikäinen, T. *Tiling Databases* in *Discovery Science* (eds Suzuki, E. & Arikawa, S.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004), 278–289. ISBN: 978-3-540-30214-8.

46. Gillis, N. *Nonnegative Matrix Factorization* eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611976410. https://epubs.siam.org/doi/abs/10.1137/1.9781611976410 (Society for Industrial and Applied Mathematics, Philadelphia, PA, 2020).

47. Gillis, N. Sparse and unique nonnegative matrix factorization through data preprocessing. *The Journal of Machine Learning Research* **13,** 3349–3386 (2012).

48. Gillis, N. & Kumar, A. Exact and Heuristic Algorithms for Semi-Nonnegative Matrix Factorization. *SIAM Journal on Matrix Analysis and Applications* **36,** 1404–1424 (2015).

49. Grant, M. & Boyd, S. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1* `http:/cvxr.com/cvx`. Mar. 2014.

50. Grasedyck, L., Klever, M. & Krämer, S. Quasi-orthogonalization for alternating non-negative tensor factorization. *Electronic Transactions on Numerical Analysis* **62,** 22–57 (2024).

51. Guennebaud, G., Jacob, B., *et al. Eigen v3* http://eigen.tuxfamily.org. 2010.

52. Günlük, O., Hauser, R. A. & Kovács, R. Á. Binary Matrix Factorisation and Completion via Integer Programming. *arXiv preprint arXiv:2106.13434* (2021).

53. Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual* `https://www.gurobi.com`. 2023.

54. Haddad, A., Shamsi, F., Zhu, L. & Najafizadeh, L. *Identifying dynamics of brain function via Boolean matrix factorization* in *Asilomar Conference on Signals, Systems, and Computers* (2018).

55. Hess, S. & Morik, K. English. in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (eds Ceci, M., Dzeroski, S., Vens, C., Todorovski, L. & Hollmen, J.) 547–563 (Springer, Germany, 2017). ISBN: 978-3-319-71248-2.

56. Hess, S., Morik, K. & Piatkowski, N. The PRIMPING routine–Tiling through proximal alternating linearized minimization. *Data Min. Knowl. Discov.* **31,** 1090–1131. ISSN: 1384-5810. `https://doi.org/10.1007/s10618-017-0508-z` (July 2017).

57. Hien, L. T. K. & Gillis, N. Algorithms for Nonnegative Matrix Factorization with the Kullback–Leibler Divergence. *J. Sci. Comput.* **87.** ISSN: 0885-7474. `https://doi.org/10.1007/s10915-021-01504-0` (June 2021).

58. Higham, N. J. Computing the nearest correlation matrix—a problem from finance. *IMA Journal of Numerical Analysis* **22,** 329–343. ISSN: 0272-4979. eprint: `https://academic.oup.com/imajna/article-pdf/22/3/329/2089524/220329.pdf`. `https://doi.org/10.1093/imanum/22.3.329` (July 2002).

59. Hoyer, P. O. Non-negative Matrix Factorization with Sparseness Constraints. *J. Mach. Learn. Res.* **5,** 1457–1469. ISSN: 1532-4435 (Dec. 2004).

60. Huang, K., Sidiropoulos, N. D. & Liavas, A. P. A Flexible and Efficient Algorithmic Framework for Constrained Matrix and Tensor Factorization. *Trans. Sig. Proc.* **64,** 5052–5065. ISSN: 1053-587X. `https://doi.org/10.1109/TSP.2016.2576427` (Oct. 2016).

61. Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. & Amodei, D. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361.* `https://arxiv.org/abs/2001.08361` (2020).

62. Kolomvakis, C. & Gillis, N. *Robust Binary Component Decompositions* in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2023), 1–5.

63. Kolomvakis, C., Vandaele, A. & Gillis, N. *Algorithms for Boolean Matrix Factorization using Integer Programming* in *33rd International Workshop on Machine Learning for Signal Processing (MLSP)* (2023).

64. Kolomvakis, C., Vandaele, A. & Gillis, N. *Boolean Matrix Tri-Factorization* in *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2025), 1–5.

65. Kovacs, R. A., Gunluk, O. & Hauser, R. A. *Binary Matrix Factorisation via Column Generation.* in *AAAI* (2021), 3823–3831. ISBN: 978-1-57735-866-4.

66. Kovács, R., Günlük, O. & Hauser, R. A. Low-Rank Boolean Matrix Approximation by Integer Programming. *CoRR* **abs/1803.04825.** arXiv: 1803.04825. http://arxiv.org/abs/1803.04825 (2018).

67. Koyutürk, M. & Grama, A. *PROXIMUS: a framework for analyzing very high dimensional discrete-attributed datasets* in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, Washington, D.C., 2003), 147–156. ISBN: 1581137370. https://doi.org/10.1145/956750.956770.

68. Krebs, V. A network of books about recent us politics sold by the online bookseller amazon. com. *Unpublished http://www. orgnet. com* (2008).

69. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks* in *Advances in Neural Information Processing Systems 25* (eds Pereira, F., Burges, C. J. C., Bottou, L. & Weinberger, K. Q.) (Curran Associates, Inc., 2012), 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

70. Kueng, R. & Tropp, J. A. Binary Component Decomposition Part I: The Positive-Semidefinite Case. *SIAM Journal on Mathematics of Data Science* **3,** 544–572. eprint: https://doi.org/10.1137/19M1278612. https://doi.org/10.1137/19M1278612 (2021).

71. Kueng, R. & Tropp, J. A. Binary component decomposition Part II: The asymmetric case. *Arxiv preprint.* arXiv: 1907.13602. http://arxiv.org/abs/1907.13602 (2019).

72. Laurent, M. & Poljak, S. On the facial structure of the set of correlation matrices. *SIAM Journal on Matrix Analysis and Applications* **17,** 530–547 (1996).

73. Lázaro-Gredilla, M., Liu, Y., Phoenix, D. S. & George, D. Hierarchical compositional feature learning. *arXiv preprint arXiv:1611.02252* (2016).

74. Lee, D. D. & Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature* **401,** 788–791 (1999).

75. Lehmann, S., Schwartz, M. & Hansen, L. K. Biclique communities. *Phys. Rev. E* **78,** 016108 (1 July 2008).

76. Li, X., Wang, J. & Kwong, S. Boolean matrix factorization based on collaborative neurodynamic optimization with Boltzmann machines. *Neural Networks* **153,** 142–151. ISSN: 0893-6080. https://www.sciencedirect.com/science/article/pii/S0893608022002118 (2022).

77. Liang, L. & Lu, S. *Noisy and Incomplete Boolean Matrix Factorizationvia Expectation Maximization* 2019. arXiv: 1905.12766 [stat.ML]. https://arxiv.org/abs/1905.12766.

78. Liang, L., Zhu, K. & Lu, S. BEM: mining coregulation patterns in transcriptomics via Boolean matrix factorization. *Bioinformatics 36 (13),* 4030–4037 (2020).

79. Lu, H., Vaidya, J. & Atluri, V. *Optimal Boolean Matrix Decomposition: Application to Role Engineering* in *International Conference on Data Engineering* (2008).

80. Lucchese, C., Orlando, S. & Perego, R. A Unifying Framework for Mining Approximate Top- k Binary Patterns. *IEEE Transactions on Knowledge and Data Engineering* **26,** 2900–2913 (2014).

81. Lucchese, C., Orlando, S. & Perego, R. in *Proceedings of the 2010 SIAM International Conference on Data Mining (SDM)* 165–176 (). eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611972801.15. https://epubs.siam.org/doi/abs/10.1137/1.9781611972801.15.

82. Mahoney, M. W. & Drineas, P. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences* **106,** 697–702. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0803205106 (2009).

83. Markovsky, I. *Low rank approximation: algorithms, implementation, applications* (Springer, 2012).

84. McCulloch, W. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology* **5,** 115–133. ISSN: 0007-4985. http://dx.doi.org/10.1007/bf02478259 (Dec. 1943).

85. Miettinen, P. *Boolean Tensor Factorizations* in *2011 IEEE 11th International Conference on Data Mining* (2011), 447–456.

86. Miettinen, P. The Boolean column and column-row matrix decompositions. *Data Mining and Knowledge Discovery* **17,** 39–56. ISSN: 1384-5810. https://doi.org/10.1007/s10618-008-0107-0 (Aug. 2008).

87. Miettinen, P., Mielikäinen, T., Gionis, A., Das, G. & Mannila, H. *The Discrete Basis Problem* in *Knowledge Discovery in Databases: PKDD 2006* (eds Fürnkranz, J., Scheffer, T. & Spiliopoulou, M.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2006), 335–346.

88. Miettinen, P., Mielikäinen, T., Gionis, A., Das, G. & Mannila, H. The discrete basis problem. *IEEE Trans. Knowl. Data Eng.* **20,** 1348–1362 (2008).

89. Miettinen, P. & Neumann, S. *Recent Developments in Boolean Matrix Factorization* in *International Joint Conference on Artificial Intelligence* (2021).

90. Miron, S., Diop, M., Larue, A., Robin, E. & Brie, D. Boolean decomposition of binary matrices using a post-nonlinear mixture approach. *Signal Processing* **178,** 107809 (2021).

91. Nesterov, Y. & Nemirovskii, A. *Interior-Point Polynomial Algorithms in Convex Programming* eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970791. https://epubs.siam.org/doi/abs/10.1137/1.9781611970791 (Society for Industrial and Applied Mathematics, 1994).

92. Ortiz-Bouza, M. & Aviyente, S. Community detection in multiplex networks based on orthogonal nonnegative matrix tri-factorization. *IEEE Access* (2024).

93. Park, N., Oh, S. & Kang, U. Fast and scalable method for distributed Boolean tensor factorization. *The VLDB Journal* **28,** 549–574. ISSN: 1066-8888. https://doi.org/10.1007/s00778-019-00538-z (Aug. 2019).

94. Ravanbakhsh, S., Poczos, B. & Greiner, R. *Boolean Matrix Factorization and Noisy Completion via Message Passing* in *Proceedings of The 33rd International Conference on Machine Learning* (eds Balcan, M. F. & Weinberger, K. Q.) **48** (PMLR, New York, New York, USA, 20–22 Jun 2016), 945–954. https://proceedings.mlr.press/v48/ravanbakhsha16.html.

95. Recht, B. A simpler approach to matrix completion. *Journal of Machine Learning Research* **12** (2011).

96. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* **65,** 386–408. ISSN: 0033-295X. http://dx.doi.org/10.1037/h0042519 (1958).

97. Rukat, T., Holmes, C. & Yau, C. *Probabilistic Boolean Tensor Decomposition* in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, Oct. 2018), 4413–4422. https://proceedings.mlr.press/v80/rukat18a.html.

98. Rukat, T., Holmes, C. C., Titsias, M. K. & Yau, C. *Bayesian Boolean Matrix Factorisation* in *International Conference on Machine Learning* (2017).

99. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. in *Neurocomputing: Foundations of Research* 696–699 (MIT Press, Cambridge, MA, USA, 1988). ISBN: 0262010976.

100. Russell, S. & Norvig, P. *Artificial Intelligence: A Modern Approach* 3rd ed. (Prentice Hall, 2010).

101. Sanderson, C. & Curtin, R. R. Armadillo: a template-based C++ library for linear algebra. *J. Open Source Softw.* **1,** 26 (2016).

102. Snasel, V., Platos, J. & Kromer, P. *Developing Genetic Algorithms for Boolean Matrix Factorization* in *Databases, Texts, Specifications, Objects* (2008). https://api.semanticscholar.org/CorpusID:10617385.

103. Snášel, V., Platoš, J. & Krömer, P. *On Genetic Algorithms for Boolean Matrix Factorization* in *2008 Eighth International Conference on Intelligent Systems Design and Applications* **2** (2008), 170–175.

104. Sørensen, M., De Lathauwer, L. & Sidiropoulos, N. D. Bilinear factorizations subject to monomial equality constraints via tensor decompositions. *Linear Algebra and its Applications* **621,** 296–333 (2021).

105. Sørensen, M., Sidiropoulos, N. D. & Swami, A. Overlapping Community Detection via Semi-Binary Matrix Factorization: Identifiability and Algorithms. *IEEE Transactions on Signal Processing* **70,** 4321–4336 (2022).

106. Stellato, B., Naik, V. V., Bemporad, A., Goulart, P. & Boyd, S. *Embedded Mixed-Integer Quadratic Optimization Using the OSQP Solver* in *European Control Conference (ECC)* (July 2018).

107. Stewart, G. & Sun, J.-g. *Matrix Perturbation Theory* (Academic Press Inc, 1990).

108. Thanh, O. V. & Gillis, N. *Minimum-Volume Nonnegative Matrix Completion* in *2024 32nd European Signal Processing Conference (EUSIPCO)* (2024), 2452–2456.

109. Trigeorgis, G., Bousmalis, K., Zafeiriou, S. & Schuller, B. W. *A deep semi-NMF model for learning hidden representations* in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32* (JMLR.org, Beijing, China, 2014), II–1692–II–1700.

110. Trnecka, M. & Vyjidacek, R. Revisiting the GreCon algorithm for Boolean matrix factorization. *Knowledge-Based Systems* **249,** 108895. ISSN: 0950-7051. https://www.sciencedirect.com/science/article/pii/S0950705122004282 (2022).

111. Truong, D. P., Skau, E., Desantis, D. & Alexandrov, B. Boolean matrix factorization via nonnegative auxiliary optimization. *IEEE Access* **9,** 117169–117177 (2021).

112. Turing, A. M. Computing Machinery and Intelligence. English. *Mind. New Series* **59,** 433–460. ISSN: 00264423. http://www.jstor.org/stable/2251299 (1950).

113. Udell, M., Horn, C., Zadeh, R., Boyd, S., *et al.* Generalized low rank models. *Foundations and Trends® in Machine Learning* **9,** 1–118 (2016).

114. Vaidya, J., Atluri, V. & Guo, Q. *The role mining problem: finding a minimal descriptive set of roles* in *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies* (Association for Computing Machinery, Sophia Antipolis, France, 2007), 175–184. ISBN: 9781595937452. https://doi.org/10.1145/1266840.1266870.

115. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. *Attention is All you Need* in *Advances in Neural Information Processing Systems* **30** (2017). https://arxiv.org/abs/1706.03762.

116. Vavasis, S. A. On the Complexity of Nonnegative Matrix Factorization. *SIAM Journal on Optimization* **20,** 1364–1377. eprint: https://doi.org/10.1137/070709967. https://doi.org/10.1137/070709967 (2010).

117. Wan, C., Chang, W., Zhao, T., Cao, S. & Zhang, C. *Geometric All-way Boolean Tensor Decomposition* in *Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) **33** (Curran Associates, Inc., 2020), 2848–2857. https://proceedings.neurips.cc/paper_files/paper/2020/file/1def1713ebf17722cbe300cfc1c88558-Paper.pdf.

118. Wan, C., Chang, W., Zhao, T., Li, M., Cao, S. & Zhang, C. *Fast and Efficient Boolean Matrix Factorization by Geometric Segmentation* in *AAAI Conference on Artificial Intelligence* (2019). https://api.semanticscholar.org/CorpusID:211076357.

119. Wan, C., Dang, P., Zhao, T., Zang, Y., Zhang, C. & Cao, S. *Bias aware probabilistic Boolean matrix factorization* in *Thirty-Eighth Conference on Uncertainty in Artificial Intelligence* (2022), 2035–2044.

120. Wang, S., Chang, T.-H., Cui, Y. & Pang, J.-S. Clustering by Orthogonal NMF Model and Non-Convex Penalty Optimization. *IEEE Transactions on Signal Processing* **69,** 5273–5288 (2021).

121. Wicker, J., Hua, Y. C., Rebello, R. & Pfahringer, B. *XOR-Based Boolean Matrix Decomposition* in *2019 IEEE International Conference on Data Mining (ICDM)* (2019), 638–647.

122. Xu, Y., Yin, W., Wen, Z. & Zhang, Y. An alternating direction algorithm for matrix completion with nonnegative factors. *Frontiers of Mathematics in China* **7,** 365–384 (2012).

123. Yoo, J. & Choi, S. Orthogonal nonnegative matrix tri-factorization for co-clustering: Multiplicative updates on Stiefel manifolds. *Information Processing & Management* **46,** 559–570. ISSN: 0306-4573. https://www.sciencedirect.com/science/article/pii/S0306457310000038 (2010).

124. Yoo, J. & Choi, S. *Probabilistic matrix tri-factorization* in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing* (2009), 1553–1556.

125. Zhang, Z., Li, T., Ding, C. & Zhang, X. *Binary matrix factorization with applications* in *IEEE Int. Conf. on Data Mining* (2007), 391–400.

126. Zou, H., Hastie, T. & Tibshirani, R. Sparse principal component analysis. *Journal of Computational and Graphical Statistics* **15,** 265–286 (2006).