Department of Computer Science
Series of Publications A
Report A-2009-4

Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms

Pauli Miettinen

Academic Dissertation

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XII, University Main Building, on 20 May 2009 at twelve o'clock noon.

University of Helsinki
Finland

ISSN 1238-8645
ISBN 978-952-10-5497-6 (paperback)
ISBN 978-952-10-5498-3 (PDF)
http://ethesis.helsinki.fi/

Computing Reviews (1998) Classification: H.2.8, F.2.1, F.2.2

Helsinki University Print
Helsinki, May 2009 (164+6 pages)

Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms

Pauli Miettinen
Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
Pauli.Miettinen@cs.Helsinki.FI
http://www.cs.helsinki.fi/pauli.miettinen/

Abstract

Matrix decompositions, where a given matrix is represented as a product of two other matrices, are regularly used in data mining. Most matrix decompositions have their roots in linear algebra, but the needs of data mining are not always those of linear algebra. In data mining one needs to have results that are interpretable – and what is considered interpretable in data mining can be very different to what is considered interpretable in linear algebra.

The purpose of this thesis is to study matrix decompositions that directly address the issue of interpretability. An example is a decomposition of binary matrices where the factor matrices are assumed to be binary and the matrix multiplication is Boolean. The restriction to binary factor matrices increases interpretability – factor matrices are of the same type as the original matrix – and allows the use of Boolean matrix multiplication, which is often more intuitive than normal matrix multiplication with binary matrices.

Also several other decomposition methods are described, and the computational complexity of computing them is studied together with the hardness of approximating the related optimization problems. Based on these studies, algorithms for constructing the decompositions are proposed.

Constructing the decompositions turns out to be computationally hard, and the proposed algorithms are mostly based on various heuristics. Nevertheless, the algorithms are shown to be capable of finding good results in empirical experiments conducted with both synthetic and real-world data.

Computing Reviews (1998) Categories and Subject Descriptors:

H.2.8	[Database Management]: Database Applications—data mining
F.2.1	[Analysis of Algorithms and Problem Complexity]: Numerical Algorithms and Problems—computations on matrices
F.2.2	[Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—computations on discrete structures

General Terms: Algorithm, Experimentation, Theory

Additional Key Words and Phrases: Matrix Decompositions, Set Cover, Binary Matrices, Column Decompositions, Column–Row Decompositions

Acknowledgements

Behind every PhD thesis there is a student, and behind every student there is a supervisor. I am grateful to my supervisor, Professor Heikki Mannila, who managed to find such a good balance between the seemingly contradicting requirements of a supervisor: he guided me firmly through the process while still providing me with the liberty to make my own decisions and mistakes.

The research in this thesis was done at HIIT and at the Department of Computer Science, University of Helsinki. I gratefully acknowledge the financial support from them, as well as from Helsinki Graduate School in Computer Science and Engineering.

But the place itself, buildings, infrastructure, means nothing without the people. And I have been lucky to have such great people around me. I thank all my co-authors for letting me work with you and learn from you. Especially I shall never forget the deep influence Aristides Gionis and Taneli Mielikäinen had on me when I was starting my PhD studies. And I am forever grateful to Professor Stefano Leonardi for giving me the possibility to visit Rome and for his hospitality during that visit.

I am grateful to many friends and colleagues of mine for their readiness to exchange ideas and dwell upon random topics: Esa, with whom I have shared an office all these years, Niina, with whom I have had many a pleasant discussion, Jussi, Jarkko, Jaana, and Matti, I thank you all.

I owe much to my parents who taught me the importance of education and who have always supported me.

But above all I must thank my beloved wife Anu. Her patience was needed when I was writing this thesis; her impatience helped me to finish it.

Contents

CHAPTER 1

Introduction

Data mining is about finding new and interesting information from data [HK01, pp. 5*ff.*]. The underlying assumption is that there is too much data for a human to interpret, and thus one needs an automated method to find that new and interesting information. What constitutes a new and interesting information is – if not completely ill-defined – at least very subjective and data- and application-dependent. This, of course, has not stopped data miners from defining various metrics for the quality of the obtained information, if for no other reason, to facilitate the comparison of various methods and their outputs. (See, e.g. [HK01, p. 27] for a characterization of what makes a data mining result interesting, and e.g. [GH06] for a survey of some metrics developed for measuring the interestingness.)

The type of information sought can be coarsely divided into two classes: local and global [HMS01, p. 9]. Local information is about a specific (not necessarily small) area of the data, and does not take the other parts of the data into account. Global information, on the other hand, is about the data on the whole. It should be noted that it might not be possible to distinguish between local and global information by just examining the results of some data mining methods; it is the process how the information was obtained that defines the type, i.e. whether the whole data was taken into account when the (parts of the) results were constructed. In the parlance of data mining, it is typical to speak about local patterns and global models (see e.g. [HMS01]) to further distinguish these two types. This thesis considers only global information (i.e. models).

Especially when global information is sought, the aspect of the size of the results, i.e. the amount of information returned to the user, becomes important (whereas, for local patterns, the optimum result would contain exactly the interesting patterns – for whatever measure of interestingness is used [HK01, p. 28]). If the volume of the information returned by the data mining method used exceeds the volume of the original information it is very unlikely that the results ease the user's burden on interpreting the data. Thus, a condensed representation of the original data is sought. But while constructing a global information about the input data the method used should take into account the whole data, the resulting representation almost never represents the whole data. Indeed, condensed representations of the whole data are called (lossless) compressions of the data, and that is not what data mining is about. Data mining is about finding the most important aspects of the data, and reporting them to the user.

A typical, and indeed natural, requirement for data mining method's results is that the user should be able to interpret them in the context of the input data [HK01, p. 27] (indeed, Hand et al. include the requirement of understandability in their very definition of data mining [HMS01, p. 1]). For example, a typical data compression method is not a good data mining method, as the user is probably unable to make any sense of the resulting bit string. Another, less extreme example would be a user inputting student information together with the courses the students have taken, and getting as an output a set of high-dimensional real vectors. Especially if the possible values in these vectors are not restricted, the results can be very hard to interpret. The interpretability of results is an ongoing topic of this thesis and the *principalis causa* of the methods presented.

Matrices are basic concepts of elementary linear algebra. In linear algebra, an n-by-m matrix is usually interpreted as a linear map from n-dimensional space to m-dimensional (see e.g. [GVL96, Chapter 2]). But in data mining, and in this thesis, matrices are a convenient way to store and manipulate data (e.g. storing text documents as term frequency matrices [HK01, p. 430]). Matrix decompositions also originate to linear algebra. In short, when performing a matrix decomposition on some matrix, it is represented as a product of two or more factor matrices. Matrix decompositions have been developed for many purposes. The purposes, stemming from problems in linear algebra (e.g. solving linear systems [GVL96, p. 94]), are not necessarily well-suited for a data mining task. Specifically, the interpretability of the factor matrices is, at best, a by-product of the definition of the decomposition and the application in hand. This has motivated many new matrix decomposition definitions and methods that aim at higher interpretability of the results.

This thesis is about new, interpretable matrix decompositions. How is the interpretability ensured? All matrix decomposition formulations presented in this thesis have certain characteristics in common. First, it is assumed that the factor matrices are smaller than the original matrix, although defining the actual size of the factor matrices is left to the user. Second, the values of the factor matrices are restricted to come from the same set of values where the original data came from. That is to say, if the original matrix was nonnegative, then so are the factor matrices; and if the original matrix was binary-valued, then so are the factor matrices.

Other tools to increase interpretability apply only to some of the proposed decompositions. With binary data, the matrix multiplication used is Boolean, that is, the addition is defined as $1 + 1 = 1$. This captures the intuition that when reconstructing the original matrix by multiplying the factor matrices, the resulting matrix should still be binary-valued. With normal matrix multiplication this would be much harder to achieve.

Another idea is to restrict one factor matrix to a subset of columns of the original matrix. Such decompositions are called column decompositions, and it is easy to see that they preserve some interpretability: if the columns of the original matrix were interpretable, then so should be their subsets.

None of the above ideas are new, but many of their combinations are. The contribution of this thesis is to formulate the new decompositions, analyse the computational complexity of constructing them, and provide algorithms for them. The analysis of the problems concentrates on the NP-hardness and on the hardness of approximability. Usually, the problems related to the decompositions are hard even to approximate, and thus the proposed algorithms are based on various heuristics. Algorithms' performance and the interpretability of the results are studied in empirical experiments conducted with synthetic and real-world data.

Experimental evaluation of the interpretability of results is not an easy task – after all, interpretability is in the eye of the beholder. The approach taken in this thesis is to take some real-world data sets that present information accessible to non-experts, apply the proposed algorithms to these data sets, and study the results. This approach cannot guarantee that the algorithms will produce interpretable results on other data sets – a goal that, without formal definition of 'interpretability', seems hard to obtain.

Scalability is one important characteristic of good data mining algorithms. In this thesis scalability is addressed by examining the asymptotic time complexity of the algorithms. This approach can yield somewhat pessimistic results compared to what one could achieve with real-world data, but it has the benefit of being immune to implementation details.

The algorithms presented in this thesis are not optimized for speed and scalability. Instead, the focus is on succinct and clear representation of the ideas underlying the algorithms. Also, the algorithms used in the experiments present a heterogeneous set of proprietary third-party algorithms (such as those that come with Matlab), algorithms implemented by their inventors, and algorithms implemented by the author, all with varying implementation methods ranging from high-level languages such as Matlab to low-level languages such as C. Hence, the empirical time complexity of the proposed algorithms is not usually studied separately, although if an algorithm was considerably slow (or exceptionally fast) it is mentioned in the experiments.

Overview of the thesis. The first new matrix decomposition formulations are given in Chapter 4, and the following two chapters continue to introduce new formulations. The first chapter to contain novel research, Chapter 3, does not discuss matrix decomposition formulations, but its contents are vital background information for Chapters 4 and 6. Chapter 2 contains preliminaries: notation, initial definitions and related work. Last chapter, Chapter 7, contains concluding remarks.

Relation to the author's prior work. This thesis is mainly based on four articles by the author, listed below. The notation and terminology have been unified, the connections between the articles have been made more clear, and some missing details have been supplied. Also all experiments with synthetic data, and most experiments with real-world data, are new, and replace those presented in the original publications.

The four articles on which this thesis is based are:

[HMT08]	Saara Hyvönen, Pauli Miettinen, and Evimaria Terzi. Interpretable nonnegative matrix decompositions. In *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*, pages 345–353, 2008.
[Mie08a]	Pauli Miettinen. The Boolean column and column-row matrix decompositions. *Data Mining and Knowledge Discovery,* 17:39–56, 2008.
[Mie08b]	Pauli Miettinen. On the positive–negative partial set cover problem. *Information Processing Letters,* 108:219–221, 2008.
[MMG⁺08]	Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering,* 20(10):1348–1362, 2008.

The ±PSC problem, discussed in Chapter 3, and results concerning it, were presented in [Mie08b]. Most of the discussion about the BU problem is new, although the problem itself was introduced in [Mie06]. Also the proof of Lemma 3.4 is new. The new algorithm presented in that chapter, `IterX`, was mentioned informally in [MMG$^+$08] and more formally explained in [Mie08a].

The contents of Chapter 4 are mostly from [MMG$^+$08], but Theorems 4.3 and 4.4 are new. Chapter 5 is based on [HMT08] with the addition of `LocNCUR` algorithm. Chapter 6 is based on [Mie08a] with the addition of Proposition 6.2, its proof, and `LocBCUR` algorithm.

CHAPTER 2

Notation and Related Work

Where we learn the notation used, and about the work related to this thesis.

2.1 Notation and Initial Definitions

This section covers the basic notation used throughout this thesis. More specific notation is introduced when it is encountered the first time. This section contains also definitions for some high-level concepts that are used in this thesis.

2.1.1 Notation

Matrices are used throughout this thesis, and they are set with capital bold type letters, as in $\mathbf{M} = (m_{ij})$. Vectors are set with lower-case bold type, $\mathbf{v}$. The ith row vector of $\mathbf{M}$ is written as $\mathbf{m}_i$, while the jth column vector is written as $\mathbf{m}^j$. The (i, j)-element of $\mathbf{M}$ is m_{ij}. Sometimes also $(\mathbf{M})_{ij}$ is used, especially when $\mathbf{M}$ is a result of an operation, as in $(\mathbf{N} + \mathbf{P})_{ij}$. If I is a set of positive integers, then $\mathbf{M}_I$ is the submatrix of $\mathbf{M}$ containing the rows $\mathbf{m}_i$ for $i \in I$, and similarly $\mathbf{M}^I$ is the submatrix of $\mathbf{M}$ containing the columns $\mathbf{m}^i$ for $i \in I$.

If S is a set and n and m are positive integers, then the set of all n-by-m matrices taking values from S is $S^{n\times m}$. The sets S commonly used for this purpose are the binary set $\{0, 1\}$, non-negative integers $\mathbb{Z}_{\geq 0}$, non-negative real numbers $\mathbb{R}_{\geq 0}$, and the set of real numbers $\mathbb{R}$. The notations $\mathbb{Z}_{\geq 0}$ and $\mathbb{R}_{\geq 0}$ are used instead of the (more common) $\mathbb{N}$ and $\mathbb{R}_+$ to clear the ambiguity with the latter notation, namely, whether 0 is in $\mathbb{N}$ and in $\mathbb{R}_+$. Two special matrices are used throughout this thesis: $\mathbf{I}_n$, the n-dimensional identity matrix and $\mathbf{J}_{n\times m}$, the n-by-m matrix full of 1s.

Let $\mathbf{A}$ and $\mathbf{B}$ be binary matrices of sizes n-by-k and k-by-m, respectively (i.e. $\mathbf{A} \in \{0,1\}^{n\times k}$ and $\mathbf{B} \in \{0,1\}^{k\times m}$). The *Boolean matrix product* of $\mathbf{A}$ and $\mathbf{B}$ is defined as normal matrix product, but with the addition defined as $1 + 1 = 1$. In other words, the algebra is done over the Boolean semi-ring $(0, 1, \vee, \wedge)$. The Boolean matrix product of $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A} \circ \mathbf{B}$. In the context of matrix multiplication (either Boolean or normal), the matrices $\mathbf{A}$ and $\mathbf{B}$ are referred to as *factor matrices*.

Sets of elements are set with normal capital letters while the elements are set with normal lower-case letters, as in $S = \{s_1, \ldots, s_n\}$. A collection of sets is denoted by $\mathcal{S} = \{S_1, \ldots, S_m\}$. The cardinality of a set (or collection) is denoted by $|S|$. If $\mathcal{S} = \{S_1, \ldots S_m\}$ is a collection, then

$$\cup\mathcal{S} = \bigcup_{S\in\mathcal{S}} S.$$

The *power set* of a set S is written as $\mathcal{P}(S)$. A *set system* is a pair $(U, \mathcal{S})$, where U is called a *universe* or *basis set* and $\mathcal{S}$ is a collection of subsets of U. Throughout this thesis the elements of sets (and sets in collections) are supposed to have some arbitrary ordering, and both sets and collections are assumed to be finite, unless otherwise stated or clear from the context. In general, sets are not assumed to contain multiple copies of same element (i.e. they are not *multi-sets*), but collections may contain same sets many times (or, as the sets are ordered, two sets containing exactly the same elements can still be considered to be different).

Some special sets will have their own notation, some of which are already defined (e.g. $\mathbb{R}$ and $\mathbb{Z}_{\geq 0}$). In general, the set $\{0, 1, 2, \ldots, n\}$ for any $n \in \mathbb{Z}_{\geq 0}$ will be denoted by $[n]$. Two shorthands, $\mathrm{poly}(n)$ and $\mathrm{polylog}(n)$, can also be considered as sets: the former is the set of all polynomials of n and the latter is the set of all polylogarithms of n (i.e. $p(\log n)$ for any polynomial $p(x)$). Alternatively, they can be interpreted as anonymous functions from the respective sets.

Set operator $\arg\max$ assumes a function and a set of elements from the domain of the function and it returns the element that maximizes the function over all elements in the input set. That is, if f is a function of nonnegative integers, $\arg\max_{n\in[N]} f(n) = \arg\max\{f(n) : n \in [N]\}$ is the integer $n \in \{0, \dots, N\}$ that maximizes $f(n)$ over all nonnegative integers less than $N + 1$. The counterpart to $\arg\max$ is $\arg\min$ that returns the element that minimizes the function.

Sets and set systems are closely related to binary vectors and matrices. Let $(U, \mathcal{S})$ be a set system with $U = \{u_1, \dots, u_n\}$. The *incidence vector* of a set $S \in \mathcal{S}$ is a binary n-dimensional column vector $\mathbf{v}$ such that $v_i = 1$ if and only if $u_i \in S$. The *incidence matrix* of $(U, \mathcal{S})$ is the n-by-m matrix $\mathbf{S}$ with $\mathbf{s}^j$ being the incidence vector of S_j for $1 \le j \le m$. If universe U is clear from context, it is omitted and thus $\mathbf{S}$ is said to be the incidence matrix of $\mathcal{S}$.

In addition to matrices and sets, also some functions are frequently used. The *indicator function* $\mathbf{1}(\cdot)$ is one of them. If P is a statement that is either true or false, then $\mathbf{1}(P) = 1$ if P is true, and $\mathbf{1}(P) = 0$ if P is false.

The *cost functions* of optimization problems are a specific class of functions. If Π is an optimization problem, I is an instance of Π, and S is a feasible solution of I, then $\mathsf{cost}_\Pi(I, S)$ denotes the cost S incurs for I; the exact definition of cost is, of course, problem-specific, but in general, cost functions are always nonnegative. When instance I is clear from the context, it can be omitted. The optimum cost of Π's instance I is denoted by $\mathsf{cost}^*_\Pi(I)$.

Two functions common in cost functions involved with matrices are the *Frobenius norm* and *sum of absolute values.* If $\mathbf{A} = (a_{ij})$ is an n-by-m matrix, the Frobenius norm $d_F(\mathbf{A})$ and sum of absolute values $d_1(\mathbf{A})$ are defined as

$$d_F(\mathbf{A}) = \|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2} \quad \text{and} \quad d_1(\mathbf{A}) = \sum_{i=1}^{n}\sum_{j=1}^{m} |a_{ij}| \,.$$

Neither of the functions places any restrictions over the values of $\mathbf{A}$. Note, however, that when $\mathbf{A} \in \{0,1\}^{n\times m}$, then $d_F(\mathbf{A})^2 = d_1(\mathbf{A})$. Thus, d_1 is common with binary matrices, while d_F is mostly used with other types of matrices.

2.1.2 A Word About Terminology

In data mining parlance, Boolean matrices are often referred to as *transaction databases*, their columns being objects and rows being transactions (or vice versa). Such terminology is not used in this thesis, but this is only a matter of convention: there is nothing that prevents one from considering the Boolean matrices of this thesis as transaction databases.

Every matrix decomposition has three concepts related to it. First of these is the formulation of the decomposition, that is, to what kind of matrices the decomposition applies (e.g. only to non-negative matrices or only to binary matrices), and what kind of factor matrices are feasible for the decomposition (e.g. nonnegative matrices or orthogonal matrices). Second concept is the concrete decomposition of some matrix $\mathbf{A}$. Third concept is the problem of finding a decomposition that admits the formulation, given some matrix $\mathbf{A}$. Which of these concepts is meant is usually clear from context, and thus for, say, Singular Value Decomposition (SVD), instead of always repeating 'SVD formulation' (or decomposition, or problem) simply 'SVD' is used. Yet, to distinguish between a decomposition and an algorithm for finding it, the names of algorithms are set in monospaced typewriter font, as in `SVD`. Finally, the term 'decomposition' (of a matrix) usually refers to an exact decomposition (of a matrix). The focus in this thesis, however, is on approximate decompositions, and therefore the term 'decomposition' usually refers to approximate decompositions; when an exact decomposition is meant, it is explicitly mentioned.

2.1.3 Decision and Optimization Problems

The problems considered in this thesis are optimization problems, specifically minimization problems. That is, the problems are of the form 'Given an instance I of Π, find a solution S of I such that S minimizes $\mathsf{cost}_\Pi(I, S)$.' To study the computational complexity of an optimization problem Π we need to consider the *decision version* of Π. If Π is a minimization problem, its decision version, denoted by d-Π, contains an extra variable t in its instance, and the problem is of the form 'Given an instance (I, t) of d-Π, *is* there a solution S of I such that $\mathsf{cost}_\Pi(I, S) \leq t$?' Unless otherwise specified, all decision versions of minimization problems studied in this thesis are formed similarly.

The decision versions of almost every problem presented in this thesis are NP-hard. Thus the *approximation factor* of the problem is an important feature of its complexity. Let Π be a minimization problem, and let $\mathfrak{A}$ be an algorithm for it so that if I is an instance of Π, then $\mathfrak{A}(I)$ is the solution of I produced by $\mathfrak{A}$. Define the approximation factor of $\mathfrak{A}$ for problem Π to be a function f such that for all I

$$\frac{\mathsf{cost}_\Pi(I, \mathfrak{A}(I))}{\mathsf{cost}^*_\Pi(I)} \le f(I). \tag{2.1}$$

It is possible that $\mathsf{cost}^*_\Pi(I) = 0$. For such instances, define $x/0 = \infty$ for $x > 0$ and $0 \cdot \infty = 0$, and thus $0/0 = 0$.

While the approximation factor is the standard measure for the quality of an approximation algorithm, it is by no means the only one. One important measure is the *absolute approximation guarantee*: an algorithm $\mathfrak{A}$ for problem Π has an absolute approximation guarantee of c (for some constant c) if

$$|\mathsf{cost}_\Pi(I, \mathfrak{A}(I)) - \mathsf{cost}^*_\Pi(I)| \le c \tag{2.2}$$

holds for all instances I of Π. The absolute approximation guarantee is usually defined to require a constant bound (as is done above), but we will also see a bound depending on the instance size.

The following terminology is used when the hardness of approximation is studied. When an optimization problem is NP-*hard to approximate to within a factor of* $f(n)$, it means that unless P = NP, there exists no polynomial-time approximation algorithm achieving approximation factor of $f(n)$; and when the problem is *quasi*-NP-*hard to approximate to within a factor of* $f(n)$, it means that unless NP $\subseteq$ DTIME$(n^{\mathrm{polylog}(n)})$, there exists no polynomial-time approximation algorithm achieving approximation factor of $f(n)$. Similar terminology is used in connection with absolute approximation guarantees.

2.1.4 Parameterized Complexity Theory

Parameterized complexity theory, originating to the work of Downey and Fellows [DF99], gives a framework to study computationally efficient ways of solving certain types of NP-hard problems. As the name suggests, the problems studied in parameterized complexity theory involve some parameter, the value of which can be independent of the instance size. A *parameterized problem instance* is a pair (I, k), where I is the original problem instance and $k \in \mathbb{Z}_{\geq 0}$ is the parameter. Parameterized problems are not optimization problems, but decision problems: the task is to determine whether certain condition involving I and k holds or not. However, many optimization problems have natural parameterizations, and thus parameterized complexity theory can give us useful information about the computational complexity of the problem.

The theory classifies problems based on whether they can be solved computationally efficiently if the parameter is assumed to be fixed. This is captured in the notion of *fixed-parameter tractability* (fpt for short).

Definition 2.1 ([FG06, Definition 1.4])**.** Let Π be a parameterized problem, i.e. its instances are of the form (I, k).

1. An algorithm is an *fpt-algorithm* for a parameterized problem Π if there is a computable function $f\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ and a polynomial p such that for every parameterized instance (I, k) of Π, the running time of the algorithm is at most
$$f(k)p(|I|).$$
2. A parameterized problem Π is *fixed-parameter tractable* if there is an fpt-algorithm that decides it.

The class of all fixed-parameter tractable problems is denoted by FPT.

The definition of fpt-algorithms does not restrict the function $f(k)$. This allows practically unfeasible problems to belong in class FPT. An example of such a problem is the Set Basis problem (SB) [GJ79, problem SP7], defined as follows.

Problem 2.1 (Set Basis, SB)**.** Given a set system $(U, \mathcal{C})$ and a positive integer k, is there a collection $\mathcal{B} \subseteq \mathcal{P}(U)$ of at most k sets ($|\mathcal{B}| \leq k$) such that for every set $C \in \mathcal{C}$ there is a subcollection $\mathcal{B}_C \subseteq \mathcal{B}$ with $\bigcup_{B \in \mathcal{B}_C} B = C$?

The proof of the following proposition, that SB is in FPT, simultaneously illustrates some techniques to obtain a fixed-parameter tractable algorithms and the possible infeasibility of such algorithms.

Proposition 2.1 ([DF99, Exercise 3.2.3]). *The* SB *problem is in class* FPT.

Proof. Let $n = |U|$ and $m = |\mathcal{C}|$. The idea is to show that we can bound n and m with a function depending only on k. We can then use an exhaustive search to find $\mathcal{B}$. Throughout the proof we must assume that neither U nor $\mathcal{C}$ contains duplicates, and that no two elements of U are contained in exactly the same sets $C \in \mathcal{C}$. This can be done without the loss of generality.

Notice first, that in a solution of SB, each set $C \in \mathcal{C}$ must be a union of sets in $\mathcal{B}$. There are at most 2^k such subcollections for any $\mathcal{B}$ with $|B| \leq k$, and hence $m \leq 2^k$.

But there are no more than $2^{|C|}$ different configurations on which sets C an element $u \in U$ belongs to. And because we assumed that each element has a different configuration, there can be no more than $2^{|C|} \leq 2^{2^k}$ elements in U.

Therefore, if $n > 2^{2^k}$ or $m > 2^k$, we know that there is no solution of the instance, and otherwise, there are at most

$$\binom{2^n}{k} \leq (2^n)^k \leq 2^{k2^{2^k}}$$

different collections $\mathcal{B}$ to try, and for each $\mathcal{B}$, at most

$$m2^k \leq 2^k 2^k = 2^{2k}$$

different subcollections $\mathcal{B}_C$ need to be tested. Thus, the function f is

$$f(k) = 2^{k2^{2^k}} 2^{2k} = 2^{k\left(2^{2^k}+k\right)}. \qquad \square$$

The above time complexity is, in fact, pessimistic, as, for example, the subcollections $\mathcal{B}_C$ can be constructed in a polynomial time (see Lemma 3.5), yielding

$$f(k) = 2^{k2^{2^k}} p(k),$$

where $p(k)$ is a polynomial depending on k.

While the above example unarguably shows the limitations of the theory of fixed-parameter tractability, some celebrated results – for example the $O(1.2738^k + kn)$-time algorithm for the Vertex Cover problem [CKX06] – have shown the strength of it.

As with standard complexity theory, reductions play an important role in parameterized complexity theory. The reductions used in parameterized complexity theory are called *fpt-reductions.*

Definition 2.2 ([FG06, Definition 2.1])**.** Let Π and Π' be parameterized problems with instances (I, k) and (I', k'), respectively. An *fpt-reduction* from Π to Π' is pair of mappings (R_I, R_k) from instances of Π to instances of Π' such that:

1. For all (I, k), the pair $(R_I(I), R_k(k))$ is a valid instance of Π'.
2. R_I and R_k are computable by an fpt-algorithm.
3. There is a computable function $g\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that $R_k(k) = k' \leq g(k)$ for all valid parameterized instances (I, k) of Π.

As could be expected, the class FPT is closed under fpt-reductions. The final piece of parameterized complexity theory we need is the notion of W-hierarchy (see [DF99, FG06]), a hierarchy of classes of problems not believed to be fixed-parameter tractable. The classes in W-hierarchy are denoted by W[1], W[2], and so on, the hierarchy being

$$\mathrm{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots .$$

We will not need an explicit definition of the classes W[t] (which can be found e.g. in [FG06]), but it will be sufficient to know problems complete for each class W[t] under fpt-reductions. To define such problems, we need the notion of *t-normalized* formulae, defined as follows.

Definition 2.3 ([DF95])**.** A propositional formula is *t-normalized* if it is of the form product-of-sums-of-products-of-... of literals with t alternations (i.e. $\bigwedge_i \bigvee_j \bigwedge_k \cdots x_{ijk\cdots}$).

Hence, 2-normalized formula is in the conjunctive normal form. The problems we need are the Weighted t-Normalized Satisfiability problems. For $t \geq 2$, Weighted t-Normalized Satisfiability is complete for W[t] [DF95].

Problem 2.2 (Weighted t-Normalized Satisfiability (WNS$_t$))**.** Given a propositional formula φ in a t-normalized form and a nonnegative integer k, does φ have a satisfying truth assignment of weight k, where the weight of a truth assignment is the number of variables set true?

Finally, we will need *Monotone Weighted t-Normalized Satisfiability* problems, defined as above, but requiring φ to be monotone, that is, it cannot have negated variables. For even t, Monotone WNS$_t$ is complete for W[t] [DF95], and as Monotone WNS$_t$ is clearly fpt-reducible to Monotone WNS$_{t+1}$, we have the following lemma.

Lemma 2.2. *For even t, Monotone* WNS$_{t+1}$ *is* W[t]*-hard.*

2.2 Related Work

This section introduces the prior work related to the themes discussed in this thesis. The discussion in this section is more general, and some of the more specific related works are discussed later in appropriate places.

2.2.1 Real-Valued Matrix Decompositions

Probably the best-known method to decompose a matrix is the Singular Value Decomposition (SVD) [GVL96, p. 70]. It decomposes a matrix $\mathbf{A}$ into the form $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, that is, $\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = \mathbf{I}$, and $\mathbf{\Sigma}$ is a diagonal matrix with nonnegative entries—the singular values of $\mathbf{A}$. The Singular Value Decomposition gives the *optimal* rank-k approximation of the matrix $\mathbf{A}$ with respect to the Frobenius norm.[1] Matrix $\mathbf{B}$ is the optimal rank-k approximation of $\mathbf{A}$ (with respect to the Frobenius norm) if the rank of $\mathbf{B}$ is k, and for any other rank-k matrix $\mathbf{C}$, $d_F(\mathbf{A} - \mathbf{C}) \geq d_F(\mathbf{A} - \mathbf{B})$. The optimal rank-$k$ approximation of $\mathbf{A}$ can be obtained from its Singular Value Decomposition simply by setting all but the k largest singular values to 0. This result

[1]In fact, the optimality of SVD is not limited to the Frobenius norm, but holds for any unitarily invariant norms (for definition of unitarily invariant norm and proof of the claim, see [Mir60]).

is sometimes referred to as the Eckart–Young Theorem after two mathematicians who proposed it in 1936 [EY36], although the result was established some twenty years earlier (see [Ste93]).

Computing the SVD is also relatively fast: it can be done in time $O(\min\{nm^2, n^2m\})$ for n-by-m matrices [GVL96, p. 254]. The methods often employed in practice, such as Lanczos methods [GVL96, pp. 470–507], are usually even faster. Nevertheless, for extremely large matrices this can still be too much. This has motivated the study of fast, approximate decompositions based on sampling the matrix. Work done in this field include the results of Frieze et al. [FKV04], Drineas et al. [DKM06a], and Achlioptas and McSherry [AM07].

If matrix $\mathbf{A}$ is nonnegative (e.g. because it is a result of measurements that can only yield nonnegative results), interpreting the results of SVD can be problematic, as the matrices $\mathbf{U}$ and $\mathbf{V}$ can contain negative values. This problem is addressed by Nonnegative Matrix Factorization (NMF), where the factor matrices are required to have only nonnegative values. Early formulations of the NMF problem include Paatero and Tapper's [PT94], where they call it 'positive matrix factorization', and Cohen and Rothblum's [CR93]. However, probably the most famous formulation is due to Lee and Seung [LS99]. Since their article, the problem has attained a lot of research and many algorithms are developed for it; for a recent survey, see [BBL$^+$07].

In addition to SVD and NMF, many other matrix decomposition methods have been proposed, most of which are based on probabilistic models. Such methods include multinomial Principal Component Analysis (MPCA) [Bun02], probabilistic Latent Semantic Indexing (PLSI) [Hof99], and Latent Dirichlet Allocation (LDA) [BNJ03]. There has been some research on expressing these decompositions in a unified way (see Buntine and Jakulin [BJ06] and Singh and Gordon [SG08]).

2.2.2 Decompositions for Binary-Valued Matrices

All of the aforementioned methods are designed for (possibly nonnegative) real- or integer-valued matrices. But the main focus of this thesis is in decomposing binary-valued matrices, and some previously proposed methods for that are given below.

Some decomposition formulations for binary-valued matrices can be found in Kim's classical text-book [Kim82]. But as Kim's focus is on the existence of the exact decompositions, the results are not always very useful in the context of this thesis.

Many decompositions designed for approximating a binary matrix are probabilistic. They include, for example, aspect Bernoulli models [BKF09], topic models [SBM03], and logistic PCA (LPCA) [SSU03]. Logistic PCA is perhaps the one that is most similar to the work done in this thesis. It works as follows. Given an n-by-m matrix (a_{ij}), and an integer k, LPCA finds n-by-k and k-by-m matrices $\mathbf{U}$ and $\mathbf{V}$. These matrices, however, are not binary. Instead, they are real-valued, and their product defines an element-wise probability distribution for binary n-by-m matrices: The probability $\Pr(a_{ij} = 1 \mid \Theta_{ij})$ that $a_{ij} = 1$ given the product $\mathbf{\Theta} = \mathbf{UV}$ is defined to be $\sigma(\Theta_{ij}) = (1 + \mathrm{e}^{-\Theta_{ij}})^{-1}$, that is, the sigmoid (or logistic) function of Θ_{ij}.

Semi-discrete decomposition [OP83] lies between SVD and the Boolean decompositions studied here. Like SVD, semi-discrete decomposition decomposes a given matrix into three matrices, $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$, but the values in $\mathbf{U}$ and $\mathbf{V}$ are restricted to -1, 0, and 1 only. Semi-discrete decomposition was originally designed for image compression [OP83], but has since been used, for example, in information retrieval [KO98].

Also hierarchical descriptions of binary data have been studied: the Proximus framework constructs a hierarchical clustering of rows of a given binary matrix [KGR05] and hierarchical tiles are probabilistic models hierarchically decomposing a binary matrix into almost monochromatic binary submatrices [GMS04].

Tiling transaction databases (i.e. binary matrices) is another line of related research [GGM04]. A tiling covers a given binary matrix with a small number of submatrices full of 1s. The main difference to Boolean decompositions is that no 0s can be covered in a feasible tiling. Methods have been developed for finding also large approximate tiles, for example fault-tolerant patterns [BPRB06] and conjunctive clusters [MRS04], but obtaining an accurate description of the whole dataset with a small number of approximate tiles has not been studied previously explicitly.

In co-clustering (or bi-clustering) the goal is to cluster simultaneously both dimensions of a matrix [Har72]. A co-cluster is thus a tuple (R, C), R giving the indices of rows and C giving the indices of columns. Decomposing a binary matrix into two matrices can be seen as a co-clustering of binary data where the clusters can overlap. The idea of co-clustering was originally proposed by Hartigan in 1972 [Har72], but it has gained a lot of attention recently, and many new methods have been proposed; see, for example, [BDG$^+$04, MO04, RF01].

The type of Boolean matrix factorization studied in this thesis have gained very little attention hitherto. Nevertheless, Bělohlávek and Vychodil [BV06] studied some properties of Boolean factorization via formal concepts. Genetic algorithms and neural networks have also been proposed to decompose a binary matrix using Boolean decomposition. Snášel et al. [SPK$^+$08, SKPH08] proposed and studied a genetic algorithm and a neural network concluding that, with very simple data, the latter performed better than the former, but with the cost of increased computational complexity.

2.2.3 Column and Column-Row Decompositions

The column and column-row matrix decompositions (respectively CX and CUR decompositions, for short) have been studied in computer science and in numerical linear algebra, although in the latter field, the terminology has often been somewhat different. The work in numerical linear algebra is usually related to the QR matrix decomposition [DMM07], where an n-by-m matrix $\mathbf{A}$ is represented as $\mathbf{A} = \mathbf{QR}$, with $\mathbf{Q} \in \mathbb{R}^{n \times n}$ being orthogonal and $\mathbf{R} \in \mathbb{R}^{n \times m}$ being upper triangular (see [GVL96] for more about QR decompositions). For example, Stewart [Ste99, Ste05] provides algorithms to obtain a sparse low-rank approximation of a large, sparse matrix. In practice, these approximations are obtained via CX and CUR decompositions.

In their seminal work, Gu and Eisenstat [GE96] proposed a factorization called *strong rank-revealing* QR *factorization*, and gave two algorithms to compute it. Their algorithms are efficient, taking time $O(mn^2)$ (with $m \geq n$) – the same time computing an SVD takes. For more information about the related work done in numerical linear algebra, see [DMM07] and references therein.

The relative error of the algorithms by Stewart [Ste99, Ste05] (or by Berry et al. [BPS05]) is not bounded. Gu and Eisenstat bound the smallest singular value of $\mathbf{C}$. In theoretical computer science, Frieze et al. [FKV04] proposed a probabilistic algorithm that makes only two passes over matrix $\mathbf{A}$ and provides a bound

$$\|\mathbf{A} - \mathbf{CX}\|_F \leq \|\mathbf{A} - \mathbf{A}_k^*\|_F + \varepsilon \|\mathbf{A}\|_F ,$$

with $\mathbf{A}_k^*$ being the best rank-k approximation of $\mathbf{A}$. The number of columns of $\mathbf{C}$ in Frieze et al.'s algorithm is polynomial to k, $1/\varepsilon$, and $1/\delta$, with $1-\delta$ being the success probability of the algorithm. Similar results followed, see, for example, [DKM06a], reducing the number of columns in $\mathbf{C}$ and analysing also the spectral norm. Related results were also obtained for CUR decompositions by Drineas et al. [DKM06b], giving bound

$$\|\mathbf{A} - \mathbf{CUR}\|_\xi \leq \|\mathbf{A} - \mathbf{A}_k\|_\xi + \varepsilon \|\mathbf{A}\|_F$$

for $\xi = 2, F$, and with the number of columns in $\mathbf{C}$ and number of rows in $\mathbf{R}$ depending polynomially on k, $1/\varepsilon$, and $1/\delta$.

The additive error was removed by Drineas et al. [DMM07] (see also [DMM06a] and [DMM06b]), by providing probabilistic algorithms that, for $\mathbf{A}' = \mathbf{CX}$ and $\mathbf{A}' = \mathbf{CUR}$, obtain an error bounded by

$$\|\mathbf{A} - \mathbf{A}'\|_F \leq (1+\varepsilon) \|\mathbf{A} - \mathbf{A}_k\|_F .$$

Here $\mathbf{A}_k$ is again the best rank-k approximation of $\mathbf{A}$, and $0 < \varepsilon \leq 1$. The number of columns in $\mathbf{C}$ and rows in $\mathbf{R}$ depend on k, $\log(1/\delta)$ and $1/\varepsilon$.

The best algorithm for CX decompositions where the number of columns in $\mathbf{C}$ is constant (and user-defined) was recently given by Boutsidis et al. [BMD08, BMD09], who provided a probabilistic algorithm that returns a k-column matrix $\mathbf{C}$ such that

$$\|\mathbf{A} - \mathbf{CX}\|_F \leq O(k\sqrt{\log k}) \|\mathbf{A} - \mathbf{A}_k\|_F$$

with probability at least 0.7.

CHAPTER 3

The Positive–Negative Partial Set Cover and Basis Usage Problems

Where we study a variant of the Set Cover problem and algorithms for it and see how it all relates to the rest of the thesis.

3.1 Introduction and Problem Definitions

For a thesis about matrix decompositions, starting with a special variant of the well-known Set Cover problem might seem strange. The reason is that what is said here about sets and collections applies to Boolean matrices and their factorizations via incidence matrices. Particularly, the results – namely, upper and lower bounds for its approximability – obtained here for the Positive–Negative Partial Set Cover problem yield results about the Basis Usage problem, which, in turn, appears frequently in subsequent chapters. Moreover, the results about the Basis Usage problem provide a stepping-stone for many other results.

Why not start with binary matrices (i.e. with the Basis Usage problem), if that is what we are going to use? The answer is twofold: it is easier to obtain the results via the intermediate step of studying Positive–Negative Partial Set Cover, and even if they are not much used in this thesis, these intermediate results have value and applications of their own.

That said, the main problem studied in this chapter, the Positive–Negative Partial Set Cover problem is defined as follows.

Positive–Negative Partial Set Cover Problem (±PSC)**.** Given a triple $(N, P, \mathcal{Q})$, where N and P are disjoint sets and $\mathcal{Q} \subseteq \mathcal{P}(N \cup P)$, find a subcollection $\mathcal{D} \subseteq \mathcal{Q}$ that minimizes

$$\mathsf{cost}_{\pm\mathrm{PSC}}(N, P, \mathcal{D}) = |P \setminus (\cup\mathcal{D})| + |N \cap (\cup\mathcal{D})| \,. \tag{3.1}$$

The elements in sets P and N are called positive and negative elements, respectively. Hence, the goal of the problem is to cover as much of the positive elements as possible and avoid covering the negative elements; the quantity (3.1) is the number of uncovered positive elements plus the number of covered negative ones. Denote the cardinalities of the sets of the instance as follows:

$$|P| = \pi, \quad |N| = \nu, \quad \text{and} \quad |\mathcal{Q}| = k.$$

Remark. The greedy algorithm achieves an $O(\log n)$ approximation factor for the classical Set Cover problem [Joh74, Lov75, Chv79], and this factor is optimal [LY94, RS97, Fei98, AMS06]. With ±PSC an analogous greedy algorithm would select the set $Q \in \mathcal{Q}$ with the highest ratio $|P \cap Q| \,/\, |N \cap Q|$ over the positive and negative elements that are not yet covered by some of the sets selected earlier. But this algorithm does not yield any useful approximation result. To see that, consider the following instance of ±PSC. There are 2π positive elements $p_1, \ldots, p_\pi, p'_1, \ldots, p'_\pi$ and $\pi + 2$ negative elements $n_1, \ldots, n_\pi, m$, and m'. Collection $\mathcal{Q}$ has two sets for each pair (p_i, p'_i) of positive elements: S_i and S'_i. Set S_i contains p_i, p'_i and n_i, while set $S'_i = \{p_i, p'_i, m, m'\}$. Clearly the optimal solution would be to select all sets S'_i yielding cost 2, but the greedy algorithm, selecting always the best-ratio set, selects all sets S_i, yielding cost π.

A problem related to ±PSC, and a crucial problem to study in this chapter, is the *Red–Blue Set Cover* problem, due to Carr et al. [CDKM00].

Problem 3.1 (Red–Blue Set Cover, RBSC)**.** Given a triple $(R, B, \mathcal{S})$, where R and B are disjoint sets and $\mathcal{S} \subseteq \mathcal{P}(R \cup B)$, find a subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that $\mathcal{C}$ covers B (i.e. $B \setminus \cup\mathcal{C} = \emptyset$) and minimizes

$$\mathsf{cost}_{\mathrm{RBSC}}(R, B, \mathcal{C}) = |R \cap (\cup\mathcal{C})| \,. \tag{3.2}$$

Analogously to ±PSC, elements of R are called red elements and elements of B are called blue elements. In RBSC the goal is to cover all blue elements while covering as little of the red elements as possible. The cardinalities of the sets are denoted as follows:

$$|R| = \rho, \quad |B| = \beta, \quad \text{and} \quad |S| = \sigma.$$

Remark. The RBSC problem is clearly related to the ±PSC problem, the former being more restricted case of the latter. In the subsequent sections we will see that these two problems are even more related to each other than what is *prima facie* obvious. The relation between RBSC and ±PSC is not an atypical one – indeed, one can say that ±PSC is a *prize-collecting* version of RBSC. In prize-collecting problems, one is asked to return a partial solution, instead of a complete one, and the problem instance is endowed with prizes for elements; the cost of the solution is the cost of the partial solution to the original problem plus the sum of the prizes of the elements not included in the partial solution. Examples of prize-collecting problems include prize-collecting travelling salesman [Bal89] – where the salesman does not have to visit all vertices, and the cost of the solution is the length of the path plus the sum of the prizes of the unvisited vertices – and prize-collecting Steiner tree [GW95] (defined in a natural way). These problems enrich their original versions, but this is not always true. For example, the prize-collecting set cover problem (defined in a natural way) is redundant in the sense that we can always reformulate an instance of it as an instance of normal weighted set cover problem by identifying the prize of an element as a singleton set with the weight of the prize.

The above problems, ±PSC and RBSC, are the two main problems considered in this chapter. However, outside this chapter they are not used directly. Instead, in Section 3.5 we will see how the Basis Usage problem, defined below, is linked to ±PSC. In the subsequent chapters, this chapter's results are applied via it.

Basis Usage Problem (BU)**.** Given matrices $\mathbf{A} \in \{0,1\}^{n\times m}$ and $\mathbf{B} \in \{0,1\}^{n\times k}$, find a matrix $\mathbf{X} \in \{0,1\}^{k\times m}$ such that $\mathbf{X}$ minimizes

$$\mathsf{cost}_{\text{BU}}(\mathbf{A},\mathbf{B},\mathbf{X}) = d_1(\mathbf{A} - \mathbf{B} \circ \mathbf{X}). \tag{3.3}$$

The Basis Usage problem asks for a half of a matrix decomposition: given the original matrix and the left factor matrix, what is the best possible right factor matrix. The matrix decomposition is not a typical one – factor matrices are binary-valued, as is the original one, and matrix multiplication is Boolean. This decomposition, the Boolean Matrix Factorization, is studied in Chapter 4, where we will also see some applications of this chapter's results.

3.2 Connections to Data Mining

While the main purpose of this chapter is to build the stepping-stones for the subsequent chapters, the problems studied here have connections to and applications in data mining on their own.

The original motivation to RBSC was, according to Carr et al. [CDKM00], a data mining question. The RBSC problem can also be seen as a more machine-learning oriented question of aggregating classifiers. Suppose we have a collection of partial (or incomplete) classifiers over a training data, that is, a set of positive (blue) and negative (red) elements. The classifiers are partial (or incomplete) in the sense that they do not classify all elements in the training data, but only a subset of them. They can also make errors, and we view the results of such classifier as a set of elements the classifier classifies as positive. This transforms naturally to an instance of RBSC. Our goal is to aggregate these partial classifiers to a single classifier that is complete, that is, it classifies all positive elements as positive, and makes minimal error on classifying the negative elements as positive. In other words, all blue points must be covered, and the number of red points covered should be minimized.

The ±PSC problem has a similar analogy to classifier aggregation. With the same input, the goal is to minimize the number of false negatives (positive points classified as negative) plus the number of false positives (negative points classified as positive).

In addition to classifier aggregation, ±PSC has other, perhaps more indirect applications in data mining. For example, Afrati et al. [AGM04] mentioned the negated version of ±PSC, that is, a problem with a goal to maximize the number of covered positive elements minus the number of covered negative elements. Afrati et al. noted the arbitrarily bad performance of the greedy algorithm for solving the problem (cf. the remark above). They also mentioned that being able to solve that problem could improve their algorithm;

in the light of this chapter's results it seems improbable to obtain improvements from that direction.

3.3 Previous Results

As mentioned before, RBSC is the problem upon which the results of this chapter are built. It was introduced by Carr et al. [CDKM00] who also gave two results about its hardness of approximation:

1. it is quasi-NP-hard to approximate RBSC to within a factor of $\Omega\left(2^{(4\log\sigma)^{1-\varepsilon}}\right)$ for any $\varepsilon > 0$; and

2. it is NP-hard to approximate RBSC to within a factor of $\Omega\left(2^{\log^{1-\delta}\beta}\right)$, with $\delta = (\log\log\beta)^{-c}$, for any constant $c < 1/2$.

The first result was independently proved by Elkin and Peleg [EP00], and the second result was based upon a result by Dinur and Safra [DS04].

Remark. The above results are somewhat unintuitive. How tight are the bounds? The lower bounds are, essentially, of the form $f(x) = 2^{\log^{1-\varepsilon} x}$, with the first result having a fixed $\varepsilon > 0$ and the second result having $\varepsilon = \varepsilon(x) \to 0$ as $x \to \infty$. The function $f(x)$, with fixed ε, is *superpolylogarithmic* and *subpolynomial*, that is, $f(x)$ grows asymptotically faster than any polylogarithm of x, but slower than any polynomial of x (with standard notation $f(x) = \omega(\text{polylog}(x))$ and $f(x) = o(\text{poly}(x))$). Hence, the approximation factor of any polynomial-time algorithm cannot have only a polylogarithmic dependency on both σ and β.

The best upper bound for RBSC is due to Peleg [Pel07], who recently presented a $2\sqrt{\sigma\log\beta}$-approximation algorithm for it. Peleg's upper bound is polylogarithmic with respect to β, but superpolylogarithmic with respect to σ. Also, notice that when $\sigma = \log\beta$, Peleg's algorithm achieves $\sqrt{8\log\beta}$ guarantee. Thus, in the family of instances of RBSC where $\sigma = \log\beta$, Peleg's algorithm achieves rather good approximation guarantees.

3.4 Computational Complexity of the ±PSC Problem

The results about the computational complexity of the ±PSC problem are given in the next subsection. The proofs of the results are mostly postponed to Sections 3.4.2, 3.4.3, and 3.4.4. The parameterized complexity of ±PSC is studied in Section 3.4.5.

3.4.1 Results

Before heading to the main result of this chapter, the approximability of ±PSC, let us start by considering special cases of RBSC and ±PSC.

In an earlier remark we saw that the greedy algorithm does not work well for the ±PSC problem, and it is easy to see that the same example holds also in the case of RBSC. In the light of that remark, it seems that much of the hardness of RBSC (and ±PSC) lies in the fact that red (or negative) elements can belong to multiple sets. And indeed, this intuition is correct. For if no red element can be contained in more than one set, we can identify the number of red elements in each set as the *weight* of the set to obtain an instance of the familiar weighted set cover problem. Thus, in such special cases, we know that RBSC is no harder than the weighted set cover (whereas, as we know, it is much harder in general cases). From another remark we know that ±PSC can be seen as prize-collecting RBSC, and that prize-collecting set cover is as hard as weighted set cover. Hence, we can conclude that in the special case where every negative element belongs in exactly one set, the ±PSC problem is as hard as the weighted set cover problem.

When negative elements are allowed to belong to multiple sets, the ±PSC problem becomes much harder. The main result of this chapter relates the upper and lower bounds for the ±PSC's approximability to the respective bounds for RBSC.

Theorem 3.1. *(*i*)* RBSC *can be approximated to within a factor of $f(\rho, \beta, \sigma)$ if* ±PSC *can be approximated to within a factor of $f(\rho, \beta/\rho_{\max}, \sigma)$, where $\rho_{\max}$ is the maximum number of red elements in any set of the* RBSC *instance. (*ii*)* ±PSC *can be approximated to within a factor of $g(\nu + \pi, \pi, k + \pi)$ if* RBSC *can be approximated to within a factor of $g(\nu, \pi, k)$.*

What this result shows is that we can sandwich the hardness of approximating ±PSC between the hardness of approximating RBSC with different instances. Hence, whatever results we have with respect to RBSC, the same results can be applied to ±PSC. Theorem 3.1 and the previous results provide the following two corollaries.

Corollary 3.2. *For any family of instances of* ±PSC *where* ν, π, *and* k *are polynomially related, and for any* $\varepsilon > 0$,

1. *it is quasi-*NP*-hard to approximate the* ±PSC *problem to within a factor of* $\Omega(2^{(4\log k)^{1-\varepsilon}})$*; and*
2. *it is* NP*-hard to approximate the* ±PSC *problem to within a factor of* $\Omega(2^{\log^{1-\varepsilon}\pi})$.

Corollary 3.3. *There exists a polynomial-time approximation algorithm for the* ±PSC *problem that achieves an approximation factor of* $2\sqrt{(k+\pi)\log\pi}$.

The first part of Corollary 3.2 follows from Theorem 3.1 by the result of Carr et al. [CDKM00], while the algorithm in Corollary 3.3 is the Peleg's algorithm [Pel07]. These are straightforward consequences of Theorem 3.1.

The second part of Corollary 3.2 requires more careful study. It follows from a result by Dinur and Safra [DS04] applied to RBSC: there exists a family of instances of RBSC where

$$\rho_{\max} = O\left(2^{\log^{1-(\log\log\beta)^{-c'}}\beta}(\log\log\beta)^{c'}\right)$$

for some constant $c' < 1/2$, and unless P = NP there are no polynomial-time approximation algorithms for it with an approximation factor of $2^{\log^{1-(\log\log\beta)^{-c}}\beta}$ for any constant $c < 1/2$. Thus, if we let $g_c(x) = 2^{\log^{1-(\log\log x)^{-c}}x}$ for all $c < 1/2$, then assuming that P $\neq$ NP, there exists no polynomial-time approximation algorithm to ±PSC achieving an approximation factor of

$$g_c\left(\frac{\pi}{O(g_{c'}(\pi)(\log\log\pi)^{c'})}\right). \tag{3.4}$$

It remains to bound the growth of (3.4) from below. This is done in the following lemma, proof of which is postponed to Appendix A.

Lemma 3.4. *Let g_c be defined as above. Then*

$$g_c\left(\frac{\pi}{O(g_{c'}(\pi)(\log\log\pi)^{c'})}\right) = \Omega(2^{\log^{1-\varepsilon}\pi}) \tag{3.5}$$

for all $c, c' < 1/2$ and $\varepsilon > 0$.

Theorem 3.1 is proved in Sections 3.4.3 and 3.4.4, while Section 3.4.5 studies the parameterized complexity of ±PSC. But before going further, let us study two special cases of RBSC and ±PSC in Section 3.4.2.

3.4.2 The Exact-RBSC and Exact-±PSC Problems

In this section we will study simple special cases of RBSC and ±PSC called exact-RBSC and exact-±PSC. These problems, as the name suggest, ask for a collection $\mathcal{C}$ (resp. $\mathcal{D}$) such that no red element is covered (resp. all positive and no negative elements are covered), or answer 'no' if no such collection exists.

Why are these problems interesting? Assume that it would be NP-hard to decide whether the answer for, say, exact-±PSC is 'no' or not. It would then follow, that no polynomial-time algorithm could achieve any polynomially computable approximation factor for ±PSC (unless P = NP, of course). To formalize this idea, consider the problem exact-Π asking, for an instance I, to return a solution S such that $\mathsf{cost}_\Pi(I, S) = 0$ or 'no' if no such S exists; problem Π is defined in an obvious way. Let the (decision version of the) exact-Π problem be NP-hard. The claim is, that unless P = NP, no polynomial-time algorithm can approximate Π to within any polynomially computable function $f(|I|)$. For a contradiction, assume that it is possible, and that $\mathfrak{A}$ achieves an approximation factor of $r(|I|)$. Let I be an instance of Π for which the solution of exact-Π is other than 'no', and let $\mathfrak{A}(I)$ be a solution of it. Thus, $\mathsf{cost}_\Pi(I, \mathfrak{A}(I))/\mathsf{cost}^*_\Pi(I) \leq r(|I|)$. But because the solution of exact-Π was not 'no', it must be that $\mathsf{cost}^*_\Pi(I) = 0$ and hence also $\mathsf{cost}_\Pi(I, \mathfrak{A}(I))$ must be 0, meaning that $\mathfrak{A}$ must always find an exact solution if one exists. A contradiction, as exact-Π was assumed to be NP-hard.

Luckily both exact-RBSC and exact-±PSC have simple polynomial-time algorithms.

Lemma 3.5. *There exist polynomial-time algorithms for* exact-RBSC *and* exact-±PSC.

Proof. The algorithms are similar: select all sets containing no red (resp. negative) elements, and if all blue (resp. positive) elements are covered, then – and only then – an optimal solution with zero cost is found. Otherwise return 'no'. □

It is to be understood that henceforth all instances of RBSC and ±PSC are such that the cost of their optimal solution is at least 1.

3.4.3 From RBSC to ±PSC

Consider an instance $(R, B, \mathcal{S})$ of RBSC. We will map this instance to an instance of ±PSC. Let each negative element n_i correspond to exactly one red element r_i. For each blue element b_i, create $\rho_{\max} = \max_{S \in \mathcal{S}} |R \cap S|$ positive elements p^i_j, for $j = 1, \ldots, \rho_{\max}$. For each set $S \in \mathcal{S}$ create a set $Q \in \mathcal{Q}$ such that $n_i \in Q$ if $r_i \in S$ and that, for $j = 1, \ldots, \rho_{\max}$, $p^i_j \in Q$ if $b_i \in S$.

Let $\mathcal{D}$ be a solution of this instance of ±PSC. If $\mathcal{D}$ covers all positive elements, then the corresponding subsets also cover all blue elements in RBSC, and $\mathcal{D}$ is a feasible solution of $(R, B, \mathcal{S})$. Moreover, $\mathsf{cost}_{\pm\text{PSC}}(\mathcal{D}) = \mathsf{cost}_{\text{RBSC}}(\mathcal{D})$, i.e. $\mathcal{D}$ induces same costs in both problems. If, on the other hand, there exists a positive element p^i_j not covered by $\mathcal{D}$, then there must be at least $\rho_{\max}$ positive elements p^i_j not covered by $\mathcal{D}$. Thus we can add any set S with $p^i_j \in S$ to $\mathcal{D}$ without increasing the cost of the solution, as we cannot cover more than $\rho_{\max}$ negative elements with any $S \in \mathcal{S}$. If $\mathcal{C}$ is the (possibly extended) solution to RBSC induced by $\mathcal{D}$, we see that $\mathsf{cost}_{\pm\text{PSC}}(\mathcal{D}) \geq \mathsf{cost}_{\text{RBSC}}(\mathcal{C})$.

Finally, it is clear that the optimal solution of a ±PSC instance will cover exactly the negative elements corresponding to the red elements covered by the optimal solution of RBSC, i.e. the costs of the optimal solutions are equal. Hence,

$$\frac{\mathsf{cost}_{\pm\text{PSC}}(P, N, \mathcal{Q}, \mathcal{D})}{\mathsf{cost}^*_{\pm\text{PSC}}(P, N, \mathcal{Q})} \geq \frac{\mathsf{cost}_{\text{RBSC}}(R, B, \mathcal{S}, \mathcal{C})}{\mathsf{cost}^*_{\text{RBSC}}(R, B, \mathcal{S})}.$$

Therefore, if we can approximate ±PSC to within a factor of $f(\rho, \beta/\rho_{\max}, \sigma)$, then we can approximate RBSC to within a factor of $f(\rho, \beta, \sigma)$. This concludes the proof of the first part of Theorem 3.1.

3.4.4 From ±PSC to RBSC

Consider an instance $(N, P, \mathcal{Q})$ of ±PSC. For each $n_i \in N$, let there be a red element $r_i^- \in R$, and for each $p_i \in P$, let there be a blue element $b_i \in B$ and a red element $r_i^+ \in R$. For each set $Q_j \in \mathcal{Q}$, let there be a set $S_j^+ \in \mathcal{S}$ and for each positive element $p_i \in P$, let there be a set $S_i^- \in \mathcal{S}$. Define these sets as

$$S_j^+ = \{r_k^- \mid n_k \in Q_j\} \cup \{b_k \mid p_k \in Q_j\} \quad \text{and}$$
$$S_i^- = \{r_i^+, b_i\}.$$

Let $\mathcal{C}$ be a solution of the thus created RBSC instance. Create $\mathcal{D}$, a solution of the ±PSC instance, by adding each Q_j to $\mathcal{D}$ if the corresponding set S_j^+ is in $\mathcal{C}$.

To show that this reduction preserves the approximability, we start by considering the cost of $\mathcal{D}$. First, let n_k be a negative element in $\cup\mathcal{D}$. That is, there is a set Q_j so that $n_k \in Q_j$ and $Q_j \in \mathcal{D}$. But this means that the corresponding set S_j^+ must be in $\mathcal{C}$, and therefore the red element r_k^- corresponding to n_k is in $\cup\mathcal{C}$.

Second, let p_k be a positive element that is not in $\cup\mathcal{D}$, so none of the sets Q_j that contain p_k are in $\mathcal{D}$. This means that none of the sets S_j^+ that contain b_k are in $\mathcal{C}$. But as b_k must be covered by $\mathcal{C}$, it follows that S_k^- is in $\mathcal{C}$, and so r_k^+ is in $\cup\mathcal{C}$. Hence $\mathsf{cost}_{\pm\text{PSC}}(\mathcal{D}) \le \mathsf{cost}_{\text{RBSC}}(\mathcal{C})$.

Consider then $\mathcal{D}^*$, the optimal solution of $(N, P, \mathcal{Q})$. We show that the cost of the optimal solution of the RBSC instance created from the ±PSC instance is at most that of $\mathcal{D}^*$. Create $\mathcal{C}$ so that S_j^+ is in $\mathcal{C}$ if $Q_j \in \mathcal{D}^*$. For all blue elements b_i not yet covered by $\mathcal{C}$, add S_i^- to $\mathcal{C}$. It is straightforward to see that $\mathsf{cost}^*_{\pm\text{PSC}} = \mathsf{cost}_{\pm\text{PSC}}(\mathcal{D}^*) = \mathsf{cost}_{\text{RBSC}}(\mathcal{C}) \ge \mathsf{cost}^*_{\text{RBSC}}$. Therefore

$$\frac{\mathsf{cost}_{\text{RBSC}}(\mathcal{C})}{\mathsf{cost}^*_{\text{RBSC}}} \ge \frac{\mathsf{cost}_{\pm\text{PSC}}(\mathcal{D})}{\mathsf{cost}^*_{\pm\text{PSC}}},$$

so that if we can approximate RBSC to within a factor of $g(\nu, \pi, k)$, then we can approximate ±PSC to within a factor of $g(\nu+\pi, \pi, k+\pi)$.

3.4.5 Parameterized Complexity

Denote the parameterized versions of ±PSC and RBSC by p-±PSC and p-RBSC. For both problems, the parameter we consider here is the cost of the solution. First we need to know the parameterized complexity of p-RBSC.

Lemma 3.6. *The p-RBSC problem is* W[2]*-hard.*

Proof. Carr et al. [CDKM00] showed a trivial reduction of RBSC from and to Monotone WNS$_3$ (Problem 2.2, which they called Minimum Monotone Satisfying Assignment of level 3): Each red element corresponds to a variable. The innermost conjunctions correspond to sets in $\mathcal{S}$; a variable appears in a conjunction if the element corresponding to it belongs to the set corresponding the conjunction. There is one disjunction of conjunctions for each blue element, and a conjunction (set) appears in a disjunction if the blue element appears in the set. Finally, the disjunctions of conjunctions are joined with conjunctions. An example of this reduction is given below. The reduction is clearly an fpt-reduction, and thus, together with Lemma 2.2, proves the claim. □

Example 3.1. Consider the following instance of RBSC: There are two blue elements, b_1 and b_2, five red elements, r_1, r_2, r_3, r_4, and r_5, and four sets $S_1 = \{r_1, b_1\}, S_2 = \{r_2, r_3, b_1\}, S_3 = \{r_3, r_4, b_2\}$, and $S_4 = \{r_5, b_2\}$. An optimal solution uses sets S_1 and S_4 with cost of 2.

To build a monotone 3-normalized formula we start by identifying the red elements as variables; thus we have 5 variables $x_1, \ldots, x_5$. Next we build 4 conjunctions of variables, corresponding to the sets: a variable x_i appears in conjunction C_j if $r_i \in S_j$. Hence the conjunctions are: $C_1 = x_1, C_2 = x_2 \wedge x_3, C_3 = x_3 \wedge x_4$, and $C_4 = x_5$. Then we build disjunctions of C_is, corresponding to blue elements: C_i is in disjunction D_j if $b_j \in S_i$. The disjunctions are: $D_1 = C_1 \vee C_2$ and $D_2 = C_3 \vee C_4$. Finally, the formula φ is the conjunction of D_1 and D_2, that is,

$$\begin{aligned}\varphi = D_1 \wedge D_2 &= (C_1 \vee C_2) \wedge (C_3 \vee C_4) \\ &= ((x_1) \vee (x_2 \wedge x_3)) \wedge ((x_3 \wedge x_4) \vee (x_5)).\end{aligned}$$

To satisfy φ, we need to satisfy both D_1 and D_2. To satisfy D_1, we need to satisfy either C_1 or C_2, and similarly for D_2. We can satisfy C_1 by assigning x_1 to true, and satisfy C_4 (to satisfy D_2) by assigning x_5 to true. This satisfies φ. Thus satisfying D_1 by satisfying C_1 corresponds to covering b_1 by S_1; assigning x_1 to true is the cost induced by this selection. ◇

The reductions from RBSC to ±PSC (Section 3.4.3) and from ±PSC to RBSC (Section 3.4.4) can be used as the R_I part of the respective fpt-reductions. For R_k we can use identity function in both cases. For function g from part 3 of Definition 2.2 we can also use the identity function. It remains to show that there is a solution to the original instance of RBSC (resp. ±PSC) with cost at most t if and only if there is a solution to the reduced instance of ±PSC (resp. RBSC) with a cost at most t. But in the reduction from RBSC to ±PSC the costs of the optimal solutions are equal, and in the reduction from ±PSC to RBSC the cost of the optimal solution to RBSC is at most the cost of the optimal solution to ±PSC. This proves that both reductions are indeed fpt-reductions, and gives rise to the following proposition.

Proposition 3.7. *When the parameter is the cost of the solution, the p-±PSC problem is equivalent to the p-RBSC problem under fpt-reductions; especially, the p-±PSC problem is* W[2]*-hard.*

The cost of the solution is a natural parameterization for the ±PSC problem. However, it is not the only possible parameterization. Another alternative would be to take k, the number of sets in $\mathcal{Q}$, as the parameter (yet another alternatives include π and ν). It may not look very intuitive to parameterize with respect to the instance size, but notice that the cardinality of $|P \cup N|$ must also be taken into account when computing the instance size for ±PSC. Indeed, we will shortly see that k can be a very intuitive parameterization when the problem is viewed from a different perspective.

Lemma 3.8. *The p-±PSC problem is in* FPT *when the parameter is* $|\mathcal{Q}| = k$.

Proof. There are only 2^k possible subcollections $\mathcal{D}$, and evaluating the cost of each of them takes only polynomial time. □

3.5 The ±PSC and BU Problems Are Equivalent

We can now prove the above-mentioned connection between ±PSC and BU. Consider first a special case of BU where the instance consists of an n-dimensional binary column vector $\mathbf{a}$ and a n-by-k Boolean matrix $\mathbf{B}$ (i.e. matrix $\mathbf{A}$ has only one column). Thus the task is to find a Boolean k-dimensional column vector $\mathbf{x}$ minimizing

$$d_1(\mathbf{a} - \mathbf{B} \circ \mathbf{x}) = \|\mathbf{a} - \mathbf{B} \circ \mathbf{x}\|_1 , \qquad (3.6)$$

where $\|\cdot\|_1$ is the Hamming distance. It is easy to see that this problem is exactly ±PSC rephrased using matrices: identify the matrix $\mathbf{B}$ as the incidence matrix of $\mathcal{Q}$ and set $a_i = 1$ if the ith element is positive, and $a_i = 0$ if it is negative (hence, $n = \pi + \nu$). Vector $\mathbf{x}$ defines the collection $\mathcal{D}$: the jth set of $\mathcal{Q}$ is in $\mathcal{D}$ if and only if $x_j = 1$. But this special case does not limit the generality of the BU problem too much. To see that, notice that in BU each column of $\mathbf{X}$ can be considered independently, and thus solving BU is equivalent to solving the above problem once for each column of $\mathbf{A}$. We can (and hereafter will) assume that n is polynomially related to m, and this repeated computing will only cause a polynomial slowdown. As for the approximation, approximating multiple independent instances is as hard or easy as approximating a single instance. Thus we have the following proposition.

Proposition 3.9. *The results about the computational complexity of* ±PSC *hold for* BU *up to polynomial factors; the same is also true for parameterized complexity given that the parameters are corresponding. The results about the approximability of* ±PSC *hold for* BU*.*

Care must be taken when one transforms the upper and lower bounds for the approximability of ±PSC to the respective bounds for BU, as some of the parameters that are natural for the former problem are not natural for the latter one. Specifically, the size of the collection $\mathcal{S}$ transforms to the number of columns in $\mathbf{B}$, and the number of positive elements in the instance, $|P|$, corresponds to the number of 1s in $\mathbf{A}$'s columns, and therefore, to the density of $\mathbf{A}$. Those are the two characteristics that determine the complexity of BU; there are no known results bounding the approximability of it using directly the number of rows in $\mathbf{A}$.

Notice that this result also shows a clear difference between normal linear algebra and Boolean arithmetic. If one replaces the Boolean matrix product in (3.6) with normal matrix product, removes the requirement that $\mathbf{x}$ is binary, and optimizes with respect to the L_2-norm, then the problem is solvable in polynomial time using standard least-squares methods, for example, via SVD [GVL96]. The problem stays in P even if we place linear constraints on the feasible set of $\mathbf{x}$s (e.g. we require $\mathbf{x}$ to be nonnegative), as it can be solved using convex quadratic programming [KTH79].

Finally, note that we can also solve BU exactly in time $O(2^k knm)$ using Lemma 3.8 and the independence of columns of $\mathbf{X}$: there are 2^k different column vectors $\mathbf{x}$, it takes $O(nk)$ time to compute (3.6) for one column $\mathbf{a}$ when $\mathbf{x}$ is fixed, and there are m columns in total.

3.6 Algorithms for the Problems

Any algorithm solving the RBSC problem can be used to solve the ±PSC problem (and vice versa), and any algorithm solving the ±PSC problem can be used to solve the BU problem, given the results we have obtained so far. But the BU problem is their main framework in this thesis, and hence the following discussion concentrates on that problem.

3.6.1 Peleg's Algorithm

Recall that given the reduction from Section 3.4.4 we can use Peleg's $2\sqrt{\sigma \log \beta}$-approximation algorithm for RBSC [Pel07] to obtain a $2\sqrt{(k+\pi)\log \pi}$-approximation algorithm for ±PSC (Corollary 3.3). Peleg's algorithm, `PelegRB`, works as follows: it discards sets that have too many red elements, lets the weights of the remaining sets to be the number of red elements each set contain, removes the red elements from the sets, and solves the resulting weighted set cover problem using the standard greedy algorithm. Peleg [Pel07] does not give any method to compute the correct threshold for the number of red elements used to discard the sets; instead, all values from 1 to ρ are tried.

Peleg's algorithm must be restarted to compute each column of $\mathbf{X}$. The parameter ρ corresponds to the number of 0s in the column of $\mathbf{A}$ corresponding to the column of $\mathbf{X}$ currently constructed. Thus, the time complexity of `PelegRB` is $O(mzk\mathrm{SC}(n,k))$, where z is the number of 0s in $\mathbf{A}$ and $\mathrm{SC}(n,k)$ is the time it takes to solve any instance of weighted Set Cover with n elements and k sets in the input.

The approximation factor of Peleg's algorithm, when used to solve an instance of ±PSC, depends on two characteristics of the instance: k, the number of sets in $\mathcal{Q}$, and π, the number of positive elements. As mentioned above, π corresponds to the number of 1s in $\mathbf{A}$'s columns in the framework of the BU problem. Hence, the denser the matrix, the worse the algorithm is expected to perform.

3.6.2 Iterative Algorithm

The next algorithm is based on the idea of iteratively updating a greedy solution. It is explained as an algorithm for BU – as that is its main usage in this thesis – but is equivalently a ±PSC algorithm. The algorithm, called `IterX`, does not have any proved approximation guarantees but, at least in the experiments done for this thesis (Section 3.7.2), it usually outperforms `PelegRB`.

The algorithm can be divided into two parts: greedy selection and iterative update using the same greedy selection. For each column $\mathbf{a}^j$ of $\mathbf{A}$, the greedy selection starts by considering the first column of $\mathbf{B}$, $\mathbf{b}^1$, and if $\mathbf{b}^1$ covers more 1s in $\mathbf{a}^j$ than it covers 0s, $\mathbf{b}^1$ is used to cover $\mathbf{a}^j$ (i.e. x_{1j} is set to 1). Then second column of $\mathbf{B}$ is considered, but this time only those elements of $\mathbf{a}^j$ are considered that were not covered by $\mathbf{b}^1$ (given that $\mathbf{b}^1$ was used to cover $\mathbf{a}^j$, that is). Again, if $\mathbf{b}^2$ covers more uncovered 1s than it covers uncovered 0s, x_{2j} is set to 1. This procedure is repeated for each column of $\mathbf{B}$, always considering only those 1s and 0s of $\mathbf{a}^j$ that were not covered by any of the previously-considered columns of $\mathbf{B}$.

The above procedure can be expressed using the following function cover:

$$\begin{aligned}\mathsf{cover}(\mathbf{A},\mathbf{B},\mathbf{X},w) = w\,&|\{(i,j) : a_{ij} = 1 \wedge (\mathbf{B}\circ\mathbf{X})_{ij} = 1\}| \\ &- |\{(i,j) : a_{ij} = 0 \wedge (\mathbf{B}\circ\mathbf{X})_{ij} = 1\}|\,.\end{aligned} \tag{3.7}$$

The procedure starts with $\mathbf{X}$ full of 0s and proceeds to update $\mathbf{X}$ row-by-row. Notice that, as the columns of $\mathbf{A}$ are independent, ith row of $\mathbf{X}$ can be computed at once (i.e. the algorithm can first compute $\mathbf{x}_1$, then $\mathbf{x}_2$, and so on).

The purpose of the variable w is to weight the covering of 1s against the covering of 0s. This can be used to achieve more intuitive

Algorithm 1 The `IterX` algorithm for BU and ±PSC.

```
Input: Matrices $\mathbf{A} \in \{0,1\}^{n\times m}$ and $\mathbf{B} \in \{0,1\}^{n\times k}$, weight parameter $w \in \mathbb{R}_{\geq 0}$.
Output: Matrix $\mathbf{X} \in \{0,1\}^{k\times m}$ approximately minimizing $d_1(\mathbf{A} - \mathbf{B} \circ \mathbf{X})$.
1: function IterX($\mathbf{A}, \mathbf{B}, w$)
2:     $\mathbf{X} \leftarrow (x_{ij})$ with $x_{ij} = 0$ for $1 \leq i \leq k$, $1 \leq j \leq m$
3:     repeat
4:         for $i = 1, \ldots, k$ do
5:             $\mathbf{x}_i \leftarrow \arg\max_{\mathbf{x}_i \in \{0,1\}^m} \mathsf{cover}(\mathbf{A}, \mathbf{B}, \mathbf{X}, w)$
6:         end for
7:     until $d_1(\mathbf{A} - \mathbf{B} \circ \mathbf{X})$ does not decrease
8:     return $\mathbf{X}$
9: end function
```

results with very sparse datasets, as we will see in Section 4.8.6, but for now we can assume that $w = 1$.

In the iterative part the algorithm proceeds as above, but this time the matrix $\mathbf{X}$ is not empty, but contains the initial version of $\mathbf{X}$. When considering row i of $\mathbf{X}$, it is first set to all-zero and then a $\mathbf{x}_i$ maximizing $\mathsf{cover}(\mathbf{A}, \mathbf{B}, \mathbf{X}, w)$ given all other rows of $\mathbf{X}$ is computed as above. The iteration is repeated until the reconstruction error does not decrease anymore.

The algorithm `IterX` is given in whole as Algorithm 1.

3.6.3 An Integer-Programming Formulation

Lu et al. [LVA08] present an integer-programming (IP) formulation for BU. Thus, any general-purpose IP-solver can be used to solve BU. Unfortunately, the IP formulation presented by Lu et al. contains an error (Haibing Lu, personal communication). The error, however, is not hard to fix (*idem*), and the correct version is presented below. The problem instance contains matrices $\mathbf{A} = (a_{ij})$ and $\mathbf{B}$. Index i runs from 1 to n, index j runs from 1 to m, and index t runs from 1 to k (number of columns in $\mathbf{B}$). The value of M is to be defined later. The correct IP formulation is the following:

$$
\begin{aligned}
&\text{minimize } \sum_{i=1}^{n}\sum_{j=1}^{m} u_{ij} \\
&\text{subject to} \\
&(\mathbf{BX})_{ij} + u_{ij} \geq 1 \quad \text{if } a_{ij} = 1 \\
&(\mathbf{BX})_{ij} - u'_{ij} = 0 \quad \text{if } a_{ij} = 0 \\
&Mu_{ij} - u'_{ij} \geq 0 \\
&u_{ij} - u'_{ij} \leq 0 \\
&u'_{ij} \in \mathbb{Z}_{\geq 0} \\
&u_{ij}, x_{tj} \in \{0, 1\}.
\end{aligned}
\tag{IP1}
$$

The formulation has three sets of variables, x_{ij}, u_{ij}, and u'_{ij}. The first two are binary, and the last is nonnegative. Variables u_{ij} (and, indirectly, u'_{ij}) are used to count the induced error.

The intuition of the IP formulation is the following. The first inequality makes sure that $(\mathbf{BX})_{ij} \geq 1$ for those i and j for which $a_{ij} = 1$. If this is not the case, error (i.e. u_{ij}) is increased by 1. Because $(\mathbf{B} \circ \mathbf{X})_{ij} = 0$ if and only if $(\mathbf{BX})_{ij} = 0$, this inequality counts the number of uncovered 1s.

The next equation is for covered 0s. Again the aim is to distinguish between $(\mathbf{BX})_{ij} = 0$ and $(\mathbf{BX})_{ij} \neq 0$, but this time the error is not directly related to the value of $(\mathbf{BX})_{ij}$. If $(\mathbf{BX})_{ij} \geq 1$ and $a_{ij} = 0$, then u'_{ij} is positive. If u'_{ij} is positive, the actual error u_{ij} is increased by 1, and if $u'_{ij} = 0$, then no error is induced and $u_{ij} = 0$. This relation between u_{ij}s and u'_{ij}s is described in third and fourth inequalities.

The third inequality guarantees that $u_{ij} = 1$ if $u'_{ij} \geq 1$. For that, the value of M needs to be at least the maximum value of u'_{ij}. As the maximum value of $(\mathbf{BX})_{ij} = k$, letting $M \geq k$ is sufficient. The purpose of the fourth inequality is to guarantee that when $u'_{ij} = 0$, then also $u_{ij} = 0$.

To be able to solve (IP1) efficiently would be very advantageous as that would mean one could solve BU optimally. Alas, it does not seem probable, in general, to be able to do that (given the results of this chapter), and a small experiment in Section 3.7.2 further strengthens this assumption.

3.7 Experimental Evaluation

The purpose of the experimental evaluation is to study the two algorithms, `IterX` and `PelegRB`. Recall that while the latter has a provable approximation accuracy, the former can, in principle, perform arbitrarily badly. The aim is also to study the effects that different characteristics of the data have on the performance of the algorithms.

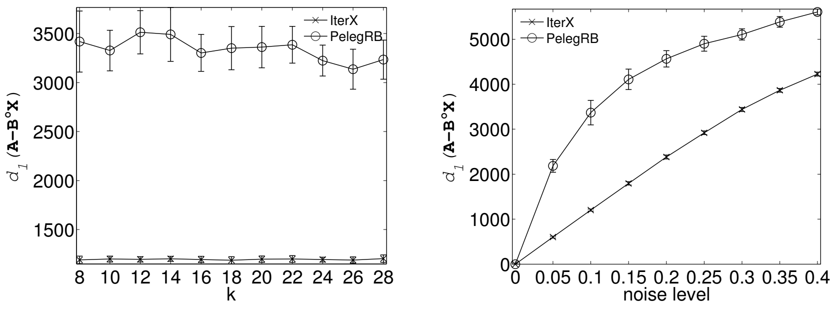
3.7.1 The Data Generation Process

To reliably achieve the stated goals, the experiments were performed using synthetic data, enabling total control over the parameters. The setting of the experiments was that of BU, that is, the problem instances contained two matrices, $\mathbf{A}$ and $\mathbf{B}$, and the task was to find $\mathbf{X}$ to minimize $\mathsf{cost}_{\text{BU}}(\mathbf{A}, \mathbf{B}, \mathbf{X})$. This setting is more valid for the subsequent chapters, and was thus used instead of the ±PSC setting.

The studied parameters were (*1*) the number of columns in $\mathbf{B}$; (*2*) the noise level; (*3*) the density of the columns in $\mathbf{B}$; and (*4*) the mean number of columns of $\mathbf{B}$ used to create each of the columns of the data. The first parameter controls the complexity of the data: the more columns in $\mathbf{B}$, the more variety there will be in $\mathbf{A}$'s columns. It is also a parameter in `PelegRB`'s approximation factor. The third and fourth parameters both change the overall density of the data, which was the other parameter in the said approximation factor.

All matrices $\mathbf{A}$ were of size 150-by-80. They were created as follows. First, a random matrix $\mathbf{B}$ was created, using the above parameters. Then, a random matrix $\mathbf{X}$ was created. The number of 1s in $\mathbf{X}$'s columns is, on expectation, the value of the fourth parameter above. A preliminary version of matrix $\mathbf{A}$ was constructed as $\mathbf{A} = \mathbf{B} \circ \mathbf{X}$. Finally, noise was introduced by changing the values of $\mathbf{A}$'s elements randomly, the probability of change being the defined noise level.

Each of the aforementioned parameters was varied in turn and 20 matrices were created for each parameter combination. The reported results are mean values over these 20 matrices. The default values for the parameters, used when they were not varied, were:



(a) (b)

Figure 3.1: Reconstruction errors of BU decompositions when (a) k, the number of columns in $\mathbf{B}$, varies; and (b) noise level varies. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.

(*1*) 16 columns in $\mathbf{B}$; (*2*) 10% of the bits were reverted due to noise; (*3*) the density of the columns of $\mathbf{B}$ was 10%; and (*4*) approximately 4 columns of $\mathbf{B}$ were combined to create a column of $\mathbf{A}$.

3.7.2 Results

Number of columns in B. The number of columns in $\mathbf{B}$, k, varied from 8 to 28 with steps of size 2. The total density of the resulting matrices was approximately 0.35.

The results of this experiment can be seen in Figure 3.1(a). The `IterX` algorithm is clearly superior to `PelegRB`, both in terms of error and deviation over different instances. Somewhat surprisingly, the overall trend of `PelegRB` seems to be towards better results when k increases, although the approximation factor of `PelegRB` depends on k. The trend is, however, very weak.

Noise level. The level of noise varied from 0 to 0.4 with steps of size 0.05. The results are presented in Figure 3.1(b). The results are intuitive as the cost induced by both algorithms increases together with the noise. Again, however, `IterX` is constantly better than `PelegRB`. When no noise is present, both algorithms return the exact solution. This was assumed, as when there is no noise, the problem is equivalent to exact-±PSC (see Section 3.4.2).

Introducing 5% of noise has dramatic effects to `PelegRB`'s performance, whereas `IterX`'s performance degenerates more smoothly.

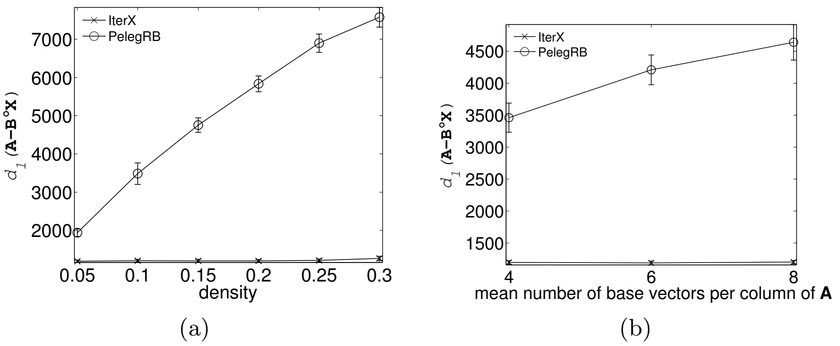


Figure 3.2: Reconstruction errors of BU decompositions when (a) density of **B**'s columns varies; and (b) the mean of **B**'s columns used to create a column of **A** (i.e. 1s in **X**'s columns) varies. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.

For both algorithms, the standard deviation is very small, being often negligible.

Density of B's columns. The mean density (number of 1s divided by the total number of elements) of **B**'s columns varied from 0.05 to 0.3. The total density of the resulting matrices varied from approximately 0.2 to approximately 0.7. The results can be seen in Figure 3.2(a).

Density is the other variable in `PelegRB`'s approximation factor, and it indeed has effects to the algorithm's performance. `IterX`, on the other hand, shows virtually no decrease of performance when the density increases.

Mean number of columns of B used for each column of A. The mean number of columns of **B** involved in the combination to create **A** had three possible values, 4, 6, and 8. The resulting matrices had an approximate total density between 0.35 and 0.55. The increased total density has, again, clear and adverse effects to `PelegRB`'s results, as can be seen in Figure 3.2(b). And, again, `IterX`'s performance is about the same in all experiments.

Solving BU optimally. There are two ways to solve BU optimally. One could either use the exhaustive search (see Section 3.5) or the IP formulation by Lu et al. [LVA08] (Section 3.6.3). To decide which method should be used, their empirical time complexity was studied. It is clear that the exhaustive search will become impractical for any larger values of k, but how about the IP formulation? The exhaustive search was implemented using C programming language and the IP solver used was the GNU Linear Programming Kit[1]. The data used was the same data that was used to study the effects of noise. The programs were executed on a dual-core 2.66GHz PC running Linux. All reported times are wall clock times.

The first test was executed with a data having no noise. Thus, the value of k was 16. The exhaustive search performed very well. With no noise, the time it took was 3.4 seconds, and with 5% of noise, it took 11.5 seconds. Solving the IP was considerably slower: with no noise, it took 118.6 seconds (almost 2 minutes), and with 5% of noise, it did not even finish after more than 4 *days*. The test was repeated with another matrix having same characteristics, and the results were same. The huge difference is explained by the fact that when there is an exact decomposition, the variables u_{ij} of (IP1) are all 0 even in the linear relaxation of (IP1), and one optimal solution of the relaxation is achieved when x_{ij}s are binary. Hence, an optimal solution to the linear relaxation is also an optimal solution to (IP1). This, of course, is not necessarily true when there is no exact decomposition present.

While this small experiment is by no means conclusive, it makes a strong point in favour of the exhaustive search – at least with moderate values of k. Thus, in the rest of this thesis, when the BU problem needs to be solved exactly, the exhaustive search is used.

3.7.3 Conclusions

Overall, the `IterX` algorithm seems to be superior to `PelegRB`, showing small variation over different instances and robustness to many characteristics of the data that have an adverse effect to `PelegRB`. Why cannot `PelegRB` do better? It is hard to identify any single reason, but one major factor seems to be the density of the matrix: `PelegRB`'s approximation guarantee depends on it, according to the experiments, for a good reason. The `IterX` algorithm will be the method of choice for the future experiments, although the lack of approximation guarantees for `IterX` means that also `PelegRB`'s performance will be studied in the same settings.

[1]http://www.gnu.org/software/glpk/

CHAPTER 4

The Boolean Matrix Factorization and Partition Problems

Where we shall study the BMF *and* BMP *problems, and see why they are interesting and how they relate to matrix ranks. The connections of* BMP *to well-known problems are revealed and algorithms for* BMF *are proposed. The algorithms are studied in empirical experiments.*

4.1 Problem Definitions

The Boolean Matrix Factorization problem complements the Basis Usage problem to a complete matrix decomposition: when BU asks for one factor matrix, given the other, the Boolean Matrix Factorization problem asks for both factor matrices. The problem is defined as follows.

Boolean Matrix Factorization Problem (BMF)**.** Given a binary matrix $\mathbf{A} \in \{0,1\}^{n\times m}$ and a positive integer k, find binary matrices $\mathbf{B} \in \{0,1\}^{n\times k}$ and $\mathbf{X} \in \{0,1\}^{k\times m}$ such that they minimize

$$\mathsf{cost}_{\text{BMF}}(\mathbf{A},\mathbf{B},\mathbf{X}) = d_1(\mathbf{A} - \mathbf{B}\circ\mathbf{X}). \tag{4.1}$$

The columns of the matrix $\mathbf{B}$ are called *basis vectors.* This is, of course, only a matter of convention, as there is nothing special about the columns of $\mathbf{B}$ compared to the rows of $\mathbf{X}$, and the roles of these two matrices could as well be exchanged.

As mentioned, the definition of BMF contains, in some sense, that of BU. Indeed, the BMF problem can be split into two subproblems: to find the left factor matrix, $\mathbf{B}$, and to find the right factor matrix, $\mathbf{X}$, given the left factor matrix. The latter is the BU problem, the former has been referred to as the Discrete Basis Problem [MMG+06, Mie06][1].

Discrete Basis Problem (DB)**.** Given a binary matrix $\mathbf{A} \in \{0,1\}^{n\times m}$ and a positive integer k, find a binary matrix $\mathbf{B} \in \{0,1\}^{n\times k}$ such that there exists a binary matrix $\mathbf{X} \in \{0,1\}^{k\times m}$ that, together with $\mathbf{B}$, minimizes

$$\mathsf{cost}_{\mathrm{DB}}(\mathbf{A},\mathbf{B},\mathbf{X}) = d_1(\mathbf{A} - \mathbf{B}\circ\mathbf{X}). \tag{4.2}$$

That is, a feasible solution of an instance of DB does not contain the matrix $\mathbf{X}$. Solving DB without solving BU (to find $\mathbf{X}$) is not very meaningful, and the focus in this thesis is on their combination, BMF. The BU problem is, as said, a subproblem of BMF, but it is not immediately clear that BMF is harder than – or even as hard as – BU. The complexity of BU is based on the fact that, for a given $\mathbf{A}$ and $\mathbf{B}$, it is hard to find a good $\mathbf{X}$; however, that does not imply that for a given $\mathbf{A}$ it would be hard to find $\mathbf{B}$ and $\mathbf{X}$. While not being a direct implication of the definitions, the BMF problem, indeed, is harder to approximate than the BU problem (see Section 4.4).

It should be noted here that the abbreviation BMF is also used for a related decomposition: the Binary Matrix Factorization [ZLDZ07]. The difference is that Binary Matrix Factorization uses normal matrix multiplication, not the Boolean one. These two decompositions should not be confused with each other.

We also study another version of BMF, called the Boolean Matrix Partitioning problem:

Boolean Matrix Partition Problem (BMP)**.** Given a binary matrix $\mathbf{A} \in \{0,1\}^{n\times m}$ and a positive integer k, find binary matrices $\mathbf{P}$ and $\mathbf{X}$ such that there is exactly one 1 in each row of $\mathbf{P}$ and matrices $\mathbf{P}$ and $\mathbf{X}$ minimize

$$\mathsf{cost}_{\mathrm{BMP}}(\mathbf{A},\mathbf{P},\mathbf{X}) = d_1(\mathbf{A} - \mathbf{P}\circ\mathbf{X}). \tag{4.3}$$

[1] [MMG+08], on the other hand, refers to the BMF problem as the DB problem.

The only difference between BMF and BMP is the extra constraint imposed on $\mathbf{P}$: there must be exactly one 1 in each row of $\mathbf{P}$. Matrices fulfilling this constraint are called *partition matrices* as they are incidence matrices of set systems that are partitions (see example below). Hence the name of the problem.

Example 4.1. Suppose we have a set of four elements, $S = \{s_1, \ldots, s_4\}$, and we partition it into partition $\mathcal{P}$ of two sets $P_1 = \{s_1, s_3\}$ and $P_2 = \{s_2, s_4\}$. The corresponding partition matrix $\mathbf{P}$ is then

$$\mathbf{P} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

Similarly, given $\mathbf{P}$ we can easily construct the partition $\mathcal{P}$. ◇

Why is BMP an interesting problem? Its decompositions do not have smaller reconstruction error (indeed, it is easy to see that $\mathsf{cost}^*_{\mathrm{BMF}}(\mathbf{A}) \leq \mathsf{cost}^*_{\mathrm{BMP}}(\mathbf{A})$ for all matrices $\mathbf{A}$), nor is there any direct reason why it should increase the interpretability of the results (but neither is there any direct reason why it should not). The BMP problem is introduced and studied because it turns out to be a much easier problem than BMF, and because it has an interesting relation to another well known problem, and this relation hopefully helps us better understand the computational aspects of BMF.

4.2 Boolean Decompositions as Data Mining Methods

Matrix decompositions, in general, are regularly applied in data mining, but why use Boolean decomposition? Why would the results of BMF be more appealing than, say, the results of SVD? For a simple motivation consider the following example.

Example 4.2. Consider three CS students, say, X, Y, and Z. Suppose that X is interested in the Systems specialization area, and that everybody in that area needs to study courses Operating Systems and Programming Languages. Student Z is interested in the Software specialization area, requiring Z to take courses Programming

Languages and Compilers. Student Y is interested in combining both areas, and thus needs to take all three courses (among others). These three students and their mandatory courses can be expressed as a binary matrix as follows:

$$\mathbf{A} = \underbrace{\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}}_{\text{courses}} \text{students} \ .$$

Solving BMF on **A** with $k = 2$ yields

$$\mathbf{B} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} .$$

This is a natural decomposition: columns of **B** represent the specialization areas and the rows of **X** tell which student has chosen which area. It is also exact, that is, $\mathbf{A} = \mathbf{B} \circ \mathbf{X}$. ◇

The crux of the Boolean decomposition lies in the following fact: using normal matrix multiplication we cannot achieve the previous example's result (see the next section). Then what makes the Boolean decomposition so natural in the above example? A set of courses taken by a student is indeed a set: it cannot contain duplicates. Thus a natural addition operation to the sets of courses is the set union operation, and this is exactly what the Boolean matrix multiplication does. Hence, when the data represents true presence/absence information, a Boolean decomposition is well-motivated. As discussed in the next section, it can also achieve better results than the methods based on normal matrix multiplication.

The relatively general framework of BMF decompositions has already been shown to apply to a more specific data-mining problem. After the initial publication of the problem [MMG$^+$06], Vaidya et al. [VAG07] showed how a version of the *role mining problem* can be reformulated as the BMF problem.

Finally, the idea of saving space deserves some attention. One motivation that is sometimes given for matrix decompositions is that when the inner dimension k is much smaller than n or m, then the factor matrices can be considered as a compressed form of the original matrix and storing only the factor matrices (either in disk, or in RAM) should take less space than storing the original matrix (see e.g. [LS99, BBL$^+$07, DKM06a, DKM06b]). This motivation, however, raises two questions: what are we going to do with the approximate representation of the matrix and whether the factor matrices indeed take less space than the original matrix. The first question is rarely addressed, although it seems to be very important. The decomposition of a matrix saves only some aspects of the data, namely those expressed by the factor matrices, and to yield any savings in the space, it must discard some other aspects. An accurate approximation, say, with respect to the Frobenius norm, is sometimes all we want, but there is no general reason why this should be true for *all* uses of the data. Therefore, the question of whether the aspects we have saved are those we are going to need is absolutely crucial – for we do not want to throw the baby out with the bath water!

That said, let us turn our attention to the second question, that is, whether the factor matrices actually take less space than the original one. This question is usually studied in the special case of sparse matrices: if the original matrix is sparse, can one guarantee that also the factor matrices will be sparse. For if the factor matrices of a sparse matrix can be dense (as is the case with SVD), it is questionable how much space one can save by keeping only the factor matrices. But sparsity is not the only characteristic of the data matrix one should consider. For if we know that the original matrix assumes values from some finite, fixed set, we can exploit this knowledge in order to store the matrix in smaller space (see [KO98, KO00] for similar ideas).

This brings us back to the binary matrices and Boolean decompositions. If our original matrix is binary and the factor matrices assume real values, it does not seem probable that we can save any space by using the decomposition instead of the original matrix. Thus, assuming we can settle the first question, it seems reasonable to require that binary matrices are decomposed into binary factor matrices, and better still, sparse binary matrices are decomposed into sparse binary factor matrices. These goals are achieved by a (good) BMF decomposition.

The factor matrices of BMF are binary by definition, but to see that they are sparse for sparse matrices requires some thinking. Consider a binary matrix $\mathbf{A}$ and its factor matrices $\mathbf{B}$ and $\mathbf{X}$. The product $\mathbf{B} \circ \mathbf{X}$ can be re-written as $\bigvee_{i=1}^{k} \mathbf{b}^i \mathbf{x}_i$, where $\vee$ denotes the Boolean sum. Now, if any of the matrices $\mathbf{b}^i \mathbf{x}_i$ in the sum are dense, so is the sum. And the number of 1s in $\mathbf{b}^i \mathbf{x}_i$ (for binary vectors $\mathbf{b}^i$ and $\mathbf{x}_i$) is the number of 1s in $\mathbf{b}^i$ times the number of 1s in $\mathbf{x}_i$. Hence, if the matrices $\mathbf{B}$ and $\mathbf{X}$ are dense, then so is their product, and if $\mathbf{A}$ was sparse, that product cannot be a good approximation of $\mathbf{A}$ (nothing, of course, prevents *bad* approximations of sparse matrices being dense).

Apart from previous paragraphs, the issue of saving storage space by storing only the factor matrices is not discussed in this thesis. Further discussion would require a specific use for which the decomposition could be used instead of the original matrix, more formal treatment of the vague terms like ‘sparse’ and ‘dense’, and an evaluation of different methods for storing the matrices. All this is considered to be out of the scope of this thesis.

4.3 Matrix Ranks

Matrix rank is a basic concept in linear algebra. It is usually defined to be the number of linearly independent columns (or rows) of the matrix, or, equivalently, as the dimension of the image of the linear map defined by the matrix. These definitions are not very intuitive in the context of this thesis: the matrices considered here are not well interpret as linear maps. Nevertheless, there are other definitions of matrix rank, equivalent in the case of real-valued matrices and better suited in our purposes.

4.3.1 Matrix Decompositions and Matrix Ranks

Recall the setting of Example 4.2. That is, $\mathbf{A}$ is a 3-by-3 binary matrix, where the rows represent the students X, Y, and Z, the columns represent the courses Operating Systems, Programming Languages, and Compilers, and $a_{ij} = 1$ denotes that student i has taken course j.

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} .$$

Using Boolean matrix decomposition, the matrix **A** can be expressed exactly as

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{B} \circ \mathbf{X}.$$

The same is not possible with any method based on normal matrix multiplication. Consider, for example, the rank-2 singular value decomposition of **A**:

$$\mathbf{U} = \begin{pmatrix} 0.50 & 0.71 \\ 0.71 & 0 \\ 0.50 & -0.71 \end{pmatrix} \mathbf{\Sigma} = \begin{pmatrix} 2.41 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{V} = \begin{pmatrix} 0.50 & 0.71 \\ 0.71 & 0 \\ 0.50 & -0.71 \end{pmatrix}.$$

To gain some intuition about the values in **U** and **V**, one can notice that $0.71 \approx 1/\sqrt{2}$.

Compare the column vectors of **B** and **U**: the former present a clear intuition about the data while the latter are arguably hard to interpret. The Boolean decomposition was exact, but the *rank* of matrix **A** is 3, and the approximation of **A** produced by SVD with rank-2 decomposition is

$$\mathbf{U\Sigma V}^T = \begin{pmatrix} 1.10 & 0.85 & 0.10 \\ 0.85 & 1.21 & 0.85 \\ 0.10 & 0.85 & 1.10 \end{pmatrix}.$$

By the optimality of SVD, this is the best that can be achieved by looking at real matrices of rank 2 and squared error.

Optimality for arbitrary matrices is not the whole story, however. For binary matrices, one can study different types of ranks. The *real rank* $\mathrm{rank}_{\mathbb{R}}(\mathbf{A})$ of a binary matrix **A** is simply the smallest value of k such that $\mathbf{A} = \mathbf{BX}$ with an n-by-k matrix **B**, a k-by-m matrix **X**, and using normal matrix multiplication. The *nonnegative rank* $\mathrm{rank}_{\mathbb{R}_{\geq 0}}(\mathbf{A})$ of **A** is similar to the real rank, but the factor matrices are restricted to have only nonnegative values. The *nonnegative integer rank* $\mathrm{rank}_{\mathbb{Z}_{\geq 0}}(\mathbf{A})$ of **A** further restricts the factor matrices **B** and **X** to only contain nonnegative integers. Finally, the *Boolean rank* $\mathrm{rank}_B(\mathbf{A})$ of **A** is the smallest k such that $\mathbf{A} = \mathbf{B} \circ \mathbf{X}$, where **B** is an n-by-k binary matrix, **X** is a k-by-m binary matrix, and the matrix multiplication is Boolean.

The Boolean rank is thus the smallest k such that there exists a BMF decomposition with zero error, the real rank is the smallest k such that there exists an SVD decomposition with zero error, and the nonnegative rank is the smallest k such that there exists an NMF decomposition with zero error. For BMP, it follows from the requirement of the matrix $\mathbf{X}$ being a partition, that the normal arithmetic could be used in the matrix multiplication: all of the summations involved in the matrix multiplication can have at most one non-zero term. Thus, the nonnegative integer rank is a lower bound for k such that we can solve BMP with zero error. It is, however, only a lower bound: the requirement that $\mathbf{X}$ is a partition is a sufficient, yet not necessary, condition to make sure that the product $\mathbf{BX}$ is a binary matrix.

The concepts of real and Boolean ranks discuss the exact representation of the matrix $\mathbf{A}$, but often only an approximate representation is sought. One could define the e-ranks $\text{rank}^e_{\mathbb{R}}(\mathbf{A})$, $\text{rank}^e_{\mathbb{R}_{\geq 0}}(\mathbf{A})$, $\text{rank}^e_{\mathbb{Z}_{\geq 0}}(\mathbf{A})$, and $\text{rank}^e_B(\mathbf{A})$ to be the least integer k such that there exists a rank-k matrix $\mathbf{B}$ for which $\|\mathbf{A} - \mathbf{B}\| \leq e$. For example, the $\text{rank}^e_B(\mathbf{A})$ of an n-by-m binary matrix $\mathbf{A}$ is the smallest k such that $d_1(\mathbf{A} - \mathbf{B} \circ \mathbf{X}) \leq e$ with $\mathbf{B} \in \{0,1\}^{n\times k}$ and $\mathbf{X} \in \{0,1\}^{k\times m}$.

4.3.2 Relations Between Ranks

There have been some studies with respect to the relations of the aforementioned ranks (see [MPR95, GP83]). In particular, the following inequalities hold for the nonnegative rank of a binary matrix $\mathbf{A}$ [GP83]:

$$\text{rank}_{\mathbb{R}}(\mathbf{A}) \leq \text{rank}_{\mathbb{R}_{\geq 0}}(\mathbf{A}) \quad \text{and} \tag{4.4}$$

$$\text{rank}_B(\mathbf{A}) \leq \text{rank}_{\mathbb{R}_{\geq 0}}(\mathbf{A}). \tag{4.5}$$

Similar inequalities hold also for the nonnegative integer rank of a binary matrix $\mathbf{A}$ [GP83]:

$$\text{rank}_{\mathbb{R}}(\mathbf{A}) \leq \text{rank}_{\mathbb{Z}_{\geq 0}}(\mathbf{A}) \quad \text{and} \tag{4.6}$$

$$\text{rank}_B(\mathbf{A}) \leq \text{rank}_{\mathbb{Z}_{\geq 0}}(\mathbf{A}). \tag{4.7}$$

Inequality (4.6) follows because both ranks use the same arithmetic, and (4.7) follows because the factor matrices are binary in both cases.

Between the real and Boolean ranks there are no clear relations. It can be shown that there are binary matrices $\mathbf{A}$ for which $\mathrm{rank}_{\mathbb{R}}(\mathbf{A}) < \mathrm{rank}_B(\mathbf{A})$ and vice versa [MPR95]. The complement of the identity matrix of size n-by-n is an example where $\mathrm{rank}_B(\mathbf{A}) = O(\log n)$, but $\mathrm{rank}_{\mathbb{R}}(\mathbf{A}) = n$ [MPR95]. This shows that while SVD can use the space of reals, BMF can, at least in some cases, take advantage of the properties of Boolean operations to achieve much smaller rank than SVD. Thus it is not a priori obvious that SVD will produce more concise representations than the Boolean methods.

Computing the real rank is easy (excluding the precision issues), and can be done, for example, using SVD: the real rank of a matrix is the number of its non-zero singular values [GVL96, p. 71]. Computing the Boolean rank, on the other hand, is NP-complete: identifying a binary matrix as an adjacency matrix of some bipartite graph G, the Boolean rank of that matrix is exactly the number of complete bipartite subgraphs needed to cover all edges of G [MPR95]. This problem, covering by complete bipartite subgraphs, is well known to be NP-complete [GJ79, problem GT18].

Approximating Boolean rank is also hard. It was shown by Simon [Sim90] to be as hard to approximate as the problem of partitioning a graph into cliques [GJ79, problem GT15] (equivalently, as hard to approximate as the minimum chromatic number). Yet, there exist some upper and lower bounds for the Boolean rank, and the relation between the Boolean and real ranks is known in some special cases; see, for example, [MPR95] and references therein. Finally, the problem "Given $\mathbf{A}$ and k, is $\mathrm{rank}_B(\mathbf{A}) \leq k$?" is fixed-parameter tractable with parameter k (see Section 2.1.4) [FG06].

In the case of e-ranks, inequality (4.5) does not hold, but inequality (4.4) does hold. For nonnegative integer rank, the upper bound property of course carries over in both cases, i.e.,

$$\mathrm{rank}^e_{\mathbb{R}}(\mathbf{A}) \leq \mathrm{rank}^e_{\mathbb{Z}_{\geq 0}}(\mathbf{A}) \quad \text{and} \tag{4.8}$$

$$\mathrm{rank}^e_B(\mathbf{A}) \leq \mathrm{rank}^e_{\mathbb{Z}_{\geq 0}}(\mathbf{A}). \tag{4.9}$$

In other words, knowing that one can solve BMP with parameter k and error e, one knows that the same error is attainable in BMF with some parameter $k' \leq k$, or with the same parameter, the error $e' \leq e$ can be achieved. Similar results also hold for the real-valued decompositions. Otherwise, even less seems to be known about e-ranks than about exact ranks.

4.4 Computational Complexity of the BMF Problem

As the BMF problem contains the BU problem, we might, at first sight, think that BMF must be at least as hard as BU. This turns out to be right, but the hardness of BMF does not seem to be a direct consequence of the hardness of BU. Recall that proving the hardness of BU requires selecting the matrix **B** adversely. This is not possible with BMF, because it is the algorithm that gets to select the matrix **B**. It is, however, probably possible to construct a reduction from BU to BMF proving some level of relation between the two (cf. Section 6.3).

Instead of having a reduction between BMF and BU, we will see a reduction between BMF and the Set Basis problem (SB), defined as Problem 2.1 in Section 2.1.4.

The BMF problem is an optimization problem: find the matrix decomposition into k basis vectors that minimizes the representation error. We formulate the decision version of it, the d-BMF problem, defined as explained in Section 2.1.3. The proof that d-BMF is NP-complete proceeds by showing that SB is a special case of d-BMF, and that d-BMF is in NP.

Theorem 4.1. *The d-BMF problem is NP-complete.*

Proof. Recall that an instance of SB is a set system $(U, \mathcal{C})$ and an integer k, and the question is, is there a collection $\mathcal{B}$ of k subsets of U such that each set of $\mathcal{C}$ can be represented as the union of some sets of $\mathcal{B}$. An instance of d-BMF contains a binary matrix **A** and nonnegative integers k and t. The question of d-BMF is, is there a BMF decomposition of **A** of size k that has error at most t.

We shall prove that for any instance of SB there is an equivalent instance of d-BMF with $t = 0$. Let $(U, \mathcal{C})$ be an instance of SB, and let **C** be the incidence matrix of $\mathcal{C}$. It is straight forward to see that SB now reduces to the problem asking 'Is there a binary matrix **B** such that each column of **C** can be represented as a Boolean combination of **B**'s columns?' That is, are there binary matrices **B** and **X** such that $\mathbf{C} = \mathbf{B} \circ \mathbf{X}$. This shows that d-BMF is NP-hard.

Finally, it immediately follows that d-BMF is in NP, because the sizes of **B** and **X** are polynomial in the size of **C**. □

Notice that in the above proof the hardness of d-BMF was solely due to the problem of finding at least one matrix of the decomposition (i.e. $\mathbf{B}$ in the above proof). Finding the other factor matrix when the other one is given and no error is allowed, that is, solving the exact-BU problem, is easy (straight forward consequence of Proposition 3.9, showing exact-BU equivalent to exact-±PSC, and Lemma 3.5).

The reduction from SB to BMF with $t = 0$ shows that (the decision version of) exact-BMF is NP-hard and hence implies the following inapproximability result (see Section 3.4.2).

Theorem 4.2. *The* BMF *problem cannot be approximated to within any polynomially-computable factor in polynomial time, unless* P = NP.

Note that solving SB is equivalent to computing the Boolean rank: the result of SB with parameter k is 'yes' if, and only if, the Boolean rank of the matrix is at most k.

The non-existence of an approximation factor to BMF seems to be – at least partly – due to its definition and the fact that its optimal solution can have zero error. Thus, asking how many times we need to multiply the optimal error to get our approximative answer might not be meaningful in the first place. An alternative would be to ask whether we can approximate BMF to within some *absolute* error. Unfortunately, that is also a hard problem. Nevertheless, before formulating the theorem, notice that we certainly can approximate BMF to within an absolute error of n^2 (if $\mathbf{A} \in \{0,1\}^{n\times n}$): no approximation, no matter how bad, can cause more error than that.

Theorem 4.3. *The* BMF *problem cannot be approximated to within any* constant *absolute approximation error in polynomial time, unless* P = NP.

Proof. For a contradiction, let us assume that $\mathfrak{A}$ is a polynomial-time algorithm for BMF and c is a constant such that for every n and m, for every $\mathbf{A} \in \{0,1\}^{n\times m}$, and for every $k \in \mathbb{Z}_{\geq 0}$, $k < \min\{n, m\}$,

$$\mathsf{cost}_{\mathrm{BMF}}(\mathbf{A}, \mathfrak{A}(\mathbf{A}, k)) \leq \mathsf{cost}^*_{\mathrm{BMF}}(\mathbf{A}, k) + c. \tag{4.10}$$

We construct a new instance of BMF, $\mathbf{A}'$, by copying $\mathbf{A}$ $c+1$ times, $\mathbf{A}' = \begin{pmatrix}\mathbf{A} & \mathbf{A} & \cdots & \mathbf{A}\end{pmatrix}$. Hence, $\mathbf{A}'$ has n rows and $(c+1)m$

columns. The parameter k is kept the same. The optimal solution of $(\mathbf{A}', k)$ decomposes $\mathbf{A}$ optimally, and copies this decomposition $c+1$ times, yielding

$$\mathsf{cost}^*(\mathbf{A}', k) = (c+1)\mathsf{cost}^*(\mathbf{A}, k). \tag{4.11}$$

To see that (4.11) indeed holds, notice first that copying the optimal decomposition of $\mathbf{A}$ gives the right-hand side of (4.11). On the other hand, if the optimal decomposition yields smaller error than that, then the average cost of approximating a copy of $\mathbf{A}$ must be smaller than $\mathsf{cost}^*(\mathbf{A}, k)$ – but this is a contradiction, and hence we have established (4.11).

Together (4.11) and (4.10) give us

$$\mathsf{cost}(\mathbf{A}', \mathfrak{A}(\mathbf{A}', k)) \leq (c+1)\mathsf{cost}^*(\mathbf{A}, k) + c. \tag{4.12}$$

As the value of k is intact, $\mathfrak{A}$ must, essentially, solve the original instance $(\mathbf{A}, k)$ $c+1$ times. To attain the bound (4.12), the average error caused by the repeated instances $(\mathbf{A}, k)$ cannot be more than the right-hand side of (4.12) divided by $c+1$. Hence, there must be at least one copy of $\mathbf{A}$, denoted by $\mathbf{A}_0$, such that

$$\begin{aligned}\mathsf{cost}(\mathbf{A}_0, \mathfrak{A}(\mathbf{A}_0, k)) &\leq \left\lfloor \frac{(c+1)\mathsf{cost}^*(\mathbf{A}, k) + c}{c+1} \right\rfloor \\ &= \mathsf{cost}^*(\mathbf{A}, k) + \left\lfloor \frac{c}{c+1} \right\rfloor \\ &= \mathsf{cost}^*(\mathbf{A}, k),\end{aligned} \tag{4.13}$$

which is a contradiction, as solving BMF optimally is an NP-hard problem (Theorem 4.1). □

It is, in fact, possible to prove a stronger version of Theorem 4.3 where c is no more a constant, but is allowed to depend on m, n or k. When c depends only on k and $c(k)$ is a polynomial on k, the above construction works as it is – the value of k is not changed. Notice, that $c(k)$ is polynomially bounded by n and m, as we can assume that $k < \min\{n, m\}$. This bound is important as the construction increases the instance size by $c(k) + 1$.

When c depends on n (or m), that is, $c = c(n)$, we have the aforementioned upper bound for the growth rate of $c(n)$: $c(n) = o(n^2)$. The claim we can establish using the previous construction bounds $c(n)$ even more, bounding it to $c(n) = \sqrt[4]{n}$.

Theorem 4.4. *There exist instances* $(\mathbf{A} \in \{0,1\}^{n\times m}, k)$ *of the* BMF *problem that cannot be approximated to within an additive error of* $c(n,m) = \max\{\sqrt[4]{n}, \sqrt[4]{m}\}$.

The proof of this theorem is an easy modification of the above proof, and is postponed to Appendix B.

4.5 Computational Complexity of the BMP Problem

Two basis vectors are said to *overlap* if they share a row having 1 (i.e. $\mathbf{b}^i$ and $\mathbf{b}^j$ overlap if there is a row k such that $b_{ik} = b_{jk} = 1$). The overlapping basis vectors seem to be a main source of difficulties in BMF. Hence, it is natural to ask whether the problem becomes easier if the basis vectors do not overlap. Such a variant of BMF is the Boolean Matrix Partitioning problem (BMP). To see that BMP can indeed be easier than BMF, consider the case where no error is allowed (i.e. exact-BMF and exact-BMP). The exact-BMF problem is NP-complete but exact-BMP can be solved in time linear in the number of ones in the input matrix since the basis vectors are exactly the maximal sets of identical rows.

We need to make a detour to study the computational complexity of the BMP problem. First, we see how to express BMP as a clustering problem: the Binary k-Median Problem (BKM). To that end, we show that BMP and BKM are equivalent problems, in the sense that all results regarding the computational complexity and approximation of one of these problems carry over to the other. Second, we study the computational complexity and hardness of approximation of BKM, obtaining simultaneously the same results for BMP.

Problem 4.1 (Binary k-Median Problem)**.** Given a set C of binary m-dimensional vectors with $|C| = n$ and a positive integer k, find a set $M = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ of binary m-dimensional vectors (the medians), and a partition $\mathcal{D} = \{D_1, \ldots, D_k\}$ of C such that M and $\mathcal{D}$ minimize

$$\mathsf{cost}_{\mathrm{BKM}}(C, M, \mathcal{D}) = \sum_{j=1}^{k} \sum_{\mathbf{c}\in D_j} \left\|\boldsymbol{\mu}_j - \mathbf{c}\right\|_1, \qquad (4.14)$$

where $\|\cdot\|_1$ is the Hamming norm.

Note that in the above definition, the vectors $\boldsymbol{\mu}_j$ are allowed to be arbitrary Boolean vectors, that is, M does not have to be a subset of C. If this is not the case (i.e. $M \subseteq C$ is required), we shall call the problem the (Binary) Graph k-Median problem (the intuition is that in the latter case we can consider the input as a weighted graph instead of a set of binary vectors). Both versions are studied here: Binary k-Median is easier to relate to BMP, while the algorithms are for (Binary) Graph k-Median. In what follows, we will first see that we can concentrate on BKM instead of BMP, and then prove results on BKM, yielding same results to BMP. Then we will see how to use algorithms for Graph k-Median to solve BKM (and BMP, also).

Remark. There exists unfortunate ambiguity concerning the problem name k-Median. For example, Arya et al. [AGK$^+$04] use k-Median to denote a problem where $M \subseteq C$ is required (Graph k-Median above), whereas for example Ackermann al. [ABS08] and Megiddo and Supowit [MS84] use k-Median (or p-Median) to denote the version that *does not* require the medians to be a subset of the input set. To complicate things even more, Chen [Che06], for example, uses the term Geometric k-Median to refer to a version of k-Median where the medians can be selected arbitrarily, but the metric must be Euclidean. The terminology in this thesis follows that of Ackermann et al.

The first task is to show that any computational complexity result for BKM is also a result for BMP (and vice versa). This is done by showing that the problems are equivalent in a sense described in the following theorem.

Theorem 4.5. *The Boolean Matrix Partitioning problem and the Binary k-Median Problem are equivalent problems, that is, there is a polynomially computable bijection $(\mathbf{A}, k, \mathbf{B}, \mathbf{X}) \mapsto (C, k', M, \mathcal{D})$ such that $\mathsf{cost}_{\text{BMP}}(\mathbf{A}, \mathbf{B}, \mathbf{X}) = \mathsf{cost}_{\text{BKM}}(C, M, \mathcal{D})$.*

Proof. The idea of the proof is to show that, with proper interpretation, the partition matrix $\mathbf{P}$ of BMP corresponds to the partition $\mathcal{D}$ of BKM, and that the medians in M correspond to the columns of $\mathbf{X}$.

For inputs, we can consider the rows of $\mathbf{A}$ (in BMP) to be the vectors in C and vice versa. The parameter k does not change. This is clearly one-to-one and onto.

Let (C, k) be an instance of BKM and let $(M, \mathcal{D})$ be a solution of it. Set M contains k m-dimensional vectors, which can be written as the rows of $\mathbf{X}$. Define the matrix $\mathbf{P} = (p_{ij})$ from partition $\mathcal{D} = \{D_1, \ldots, D_k\}$ by setting

$$p_{ij} = \begin{cases} 1 & \text{if } c_i \in D_j \\ 0 & \text{otherwise.} \end{cases}$$

Because $\mathcal{D}$ is a partition of C, matrix $\mathbf{P}$ has exactly one non-zero element in each row, and thus fulfils the definition of BMP. Using a similar approach we can create M and $\mathcal{D}$ from $\mathbf{X}$ and $\mathbf{P}$, respectively. Thus this mapping is also one-to-one and onto.

It remains to show that $\mathsf{cost}_{\mathrm{BMP}} = \mathsf{cost}_{\mathrm{BKM}}$. It holds that

$$\mathsf{cost}_{\mathrm{BMP}}(\mathbf{A}, \mathbf{P}, \mathbf{X}) = \sum_{i=1}^{n} \sum_{j=1}^{k} \|\mathbf{a}_i - \mathbf{x}_j\| \, \mathbf{1}(p_{ij} = 1)\,,$$

due to the structure of $\mathbf{P}$. As this is exactly $\mathsf{cost}_{\mathrm{BKM}}$, the claim follows. □

Remark. The above results builds also a connection between BMF and BKM, the latter being a restricted version of the former. The connection is not unique: for many clustering problems there is a matrix decomposition problem that generalizes them. This type of connections have only recently gained more attention in the literature (see [Li08] and references therein).

A polynomially computable, approximation-preserving reduction allows us to study the NP-hardness and approximability of BKM. The above theorem clearly furnishes us with such reduction for both directions (to and from BKM). The first result to prove with the reductions in hand is the NP-completeness of BKM(or, more precisely, that of d-(BKM)). The decision version of BKM, d-BKM, is defined as usual.

Lemma 4.6. *The decision version of the Binary k-Median Problem is* NP*-complete.*

For the proof, we need another problem, defined below.

Problem 4.2 ([MS84])**.** Let k and t be positive integers and let A be a set of n points from $\mathbb{R}^2$ such that if $(a_1, a_2), (a'_1, a'_2) \in A$ then

$$\{|a_i - a'_i| : i = 1, 2\} \subseteq [N], \tag{4.15}$$

where $N \leq 4n^4$. The question is: is there a set $B = \{\mathbf{b}_1, \dots, \mathbf{b}_k\}$, with $\mathbf{b}_i \in \mathbb{R}^2$ and a partition $\mathcal{E} = \{E_1, \dots, E_k\}$ of A such that

$$\sum_{j=1}^{k} \sum_{\mathbf{a} \in E_j} \|\mathbf{b}_j - \mathbf{a}\|_1 \leq t \tag{4.16}$$

holds?

Problem 4.2 is a variation of the k-Median problem, where the set of instances is restricted by (4.15). The problem was used by Megiddo and Supowit [MS84] to prove the NP-hardness of the k-median problem in $\mathbb{R}^2$. Even though Megiddo and Supowit do not explicitly state Problem 4.2, they prove that it is NP-complete.

Proof of Lemma 4.6. First notice that d-BKM is in NP.

To prove the NP-hardness, we reduce an instance (A, k, t) of Problem 4.2 to an instance (C, k, t) of the decision version of BKM in a way that the answer to (C, k, t) is 'yes' if and only if the answer to (A, k, t) is 'yes'.

Consider an instance of Problem 4.2, that is, a triplet (A, k, t). Without loss of generality, we may assume that the points in A are in the non-negative quadrant of the real plane. Furthermore, due to the restriction (4.15), we may assume that

$$A \subseteq [N] \times [N],$$

for $N = 4\,|A|^4$ (and thus, $A \subset \mathbb{Z}^2_{\geq 0}$).

Notice that the sum of the Hamming distances is minimized by the coordinate-wise median of the points in the sum. That is, for any solution $(B, \mathcal{E})$ minimizing (4.16), we may assume that $B \subseteq [N] \times [N]$.

For the reduction, we need to embed the points of A into a Boolean space. To obtain such an embedding, notice that

$$\{\|\mathbf{a} - \mathbf{a}'\|_1 : \mathbf{a}, \mathbf{a}' \in A\} \subseteq [2N].$$

We can use the standard technique of writing the coordinates of points of A in unary in order to embed A into a $2N$-dimensional Boolean space. This is done as follows. Let $\mathbf{a} = (x, y)$ be an element of A. We have assumed that $x, y \in [N]$. For the embedding, consider a $2N$-dimensional binary vector $\mathbf{b} \in \{0,1\}^{2N}$. Let the first x bits of $\mathbf{b}$ to be 1, followed by $N - x$ 0s. Then let the following y bits be again 1, and the remaining $N - y$ bits be 0. Repeating this procedure to all $\mathbf{a} \in A$ gives the desired embedding.

It is straight forward to see that this embedding does not change the Hamming distance between the points in A. Furthermore, since N is polynomially bounded in n, it can be computed in polynomial time with respect to n.

Finally, consider a solution $(M, \mathcal{D})$ of instance (C, k) so that (4.14) evaluates to at most t. Because points in C represent two integers written in unary, and because we can assume that the points in M are coordinate-wise medians of subsets of points in C, we see that also points in M represent two integers written in unary. Thus, using the reverse of the above reduction we can have a solution $(B, \mathcal{E})$ of the original instance (A, k) so that (4.16) evaluates to at most t. □

The following two corollaries are immediate consequences of Theorem 4.5 and Lemma 4.6.

Corollary 4.7. *The Boolean Matrix Partitioning problem is* NP-*hard.*

Corollary 4.8. *If it is possible to approximate* BKM *to within a constant factor in polynomial time, then it is possible to approximate* BMP *to within the same factor in polynomial time, and vice versa.*

The BKM problem can be approximated to within a factor of at most 10. To see this, recall that in the (Binary) Graph k-Median problem, set M is required to be a subset of set C. An answer to the Graph k-Median problem is clearly a valid answer to BKM, too, and – due to the triangle inequality – the error of an optimal answer to it is at most twice as large as the error of the optimal answer to BKM.

Thus, an approximation algorithm for the Graph k-Median problem with an approximation factor r is an approximation algorithm for BKM with an approximation factor $2r$. The claim now follows, because for example Arya et al. [AGK+04] have proposed an approximation algorithm for the Graph k-Median problem with an approximation ratio of at most 5 when applied to binary instances (see Section 4.7).

4.6 Algorithms for BMF

It seems natural, given the information we have, to approach the problem of solving BMF by dividing it into two subproblems: first, find the basis $\mathbf{B}$, and second, find a way to use it in the form of matrix $\mathbf{X}$. The same information tells us that this approach can lead to highly suboptimal behaviour as the matrix $\mathbf{B}$ selected can turn out to be a very hard instance for the BU problem.

Naturally we cannot, and will not, solve these two subproblems in complete isolation: to know which matrix $\mathbf{B}$ is good we need to have at least some level of information about how to use it. Nevertheless, this division is helpful in understanding the algorithms.

4.6.1 Finding the Basis

The BMF problem can be divided into two steps: first find the basis and then find a way to use it. Similarly, we can also divide the problem of finding the basis into two steps: first step is to find a set of *candidate vectors* from which the actual columns of $\mathbf{B}$ are selected in the second step.

Finding the candidates. The main method to construct the set of candidate vectors is based on the association accuracies between $\mathbf{A}$'s rows. The accuracy of an association between the ith and jth row of matrix $\mathbf{A}$ is defined as in association rule mining [AIS93], that is, $c(i \Rightarrow j, \mathbf{A}) = \langle \mathbf{a}_i, \mathbf{a}_j \rangle \,/\, \langle \mathbf{a}_i, \mathbf{a}_i \rangle$, where $\langle \cdot, \cdot \rangle$ is the vector inner product operation. An association between rows i and j is τ-strong if $c(i \Rightarrow j, \mathbf{A}) \geq \tau$.

The association accuracies are used in creating the candidate vectors via an *association accuracy matrix.* The association accuracy matrix $\mathbf{C}$ contains columns $\mathbf{c}^i$ having 1s in rows j such that $c(i \Rightarrow j, \mathbf{A}) \geq \tau$. Each column of $\mathbf{C}$ is considered as a candidate for becoming a column of $\mathbf{B}$. The threshold τ controls the level of accuracy required to include an attribute to the basis vector candidate, and it is assumed that τ is a parameter of the algorithm.

Why use association accuracies, that is, why consider the columns of the matrix $\mathbf{C}$ as candidate basis vectors? To see the intuition, consider a Boolean matrix $\mathbf{A}$ that has a decomposition $\mathbf{B} \circ \mathbf{X}$. Let $\mathbf{b}^p$ be the pth column of $\mathbf{B}$ with $b_{ip} = 1$ and $b_{jp} = 1$ for some i and j. Let q be such that $x_{pq} = 1$. Then we know that $a_{iq} = a_{jq} = 1$ for all such q. If no other column of $\mathbf{B}$ has 1 in positions i and j, then $c(i \Rightarrow j, \mathbf{A}) = c(j \Rightarrow i, \mathbf{A}) = 1$. Then, consider a more complex case when there is another column of $\mathbf{B}$, say r, that has $b_{jr} = 1$, but still $b_{ir} = 0$. In that case the rule $j \Rightarrow i$ no longer has accuracy 1 if there is a column q of $\mathbf{B}$ with $s_{pq} = 0$ and $s_{rq} = 1$. But $c(i \Rightarrow j, \mathbf{A})$ is still 1. Thus, in a case with no noise, the column $\mathbf{b}^p$ of $\mathbf{B}$ is a row of the matrix $\mathbf{A}$, given that there is i so that $b_{ip} = 1$ and $b_{iq} = 0$ for all $q \neq p$.

Example 4.3. Consider the following matrices $\mathbf{B}$, $\mathbf{X}$, and $\mathbf{B} \circ \mathbf{X} = \mathbf{A}$:

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad \mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

Now $b_{1,1} = b_{3,1} = 1$, but $b_{1,i} = b_{3,i} = 0$ for $i = 2, 3$. Hence, $c(1 \Rightarrow 3, \mathbf{A}) = c(3 \Rightarrow 1, \mathbf{A}) = 1$. The more complex case is presented in fourth row with $b_{4,1} = b_{4,2} = b_{4,3} = 1$. The accuracies for rule $4 \Rightarrow i$, where $i = 1, 2, 3, 5$, are $3/4$, $1/2$, $3/4$, and $1/2$, respectively. The accuracy for $1 \Rightarrow 4$ is, however, 1, as is the accuracy for $5 \Rightarrow 4$. In this case, selecting $\tau = 1$ gives the accuracy matrix

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

and selecting the first, second, and last column gives the correct $\mathbf{B}$. $\diamond$

Unfortunately the proposed technique does not handle more complex cases: if for all i such that $b_{ip} = 1$ there is a q such that also $b_{iq} = 1$, then we cannot find column $\mathbf{b}^p$ from $\mathbf{C}$. A concrete example of this is given by the complement of the n-by-n identity matrix, $\bar{\mathbf{I}}_n$. The Boolean rank of $\bar{\mathbf{I}}_n$ is $O(\log n)$ and real rank is n [MPR95] (see also Section 4.3). However, with $k < n$, no value of τ will give a perfect decomposition with that input, as the actual association accuracies in matrix $\mathbf{C}'$ are

$$\mathbf{C}' = \begin{pmatrix} 1 & f(n) & f(n) & \\ f(n) & 1 & f(n) & \cdots \\ f(n) & f(n) & 1 & \\ & \vdots & & \ddots \end{pmatrix}, \quad \text{with } f(n) = \frac{n-2}{n-1}.$$

The basis vector candidates in $\mathbf{C}$ are constructed from $\mathbf{C}'$ by rounding it from τ (i.e. $c_{ij} = \mathbf{1}(c'_{ij} \geq \tau)$) and thus $\mathbf{C}$ will either be full of 1s or an identity matrix, depending on the value of τ.

Constructing the matrix B. With the matrix $\mathbf{C}$ of candidate basis vectors in hand, we now need to select k of them to the columns of $\mathbf{B}$. A simple idea is to base the selection on a greedy procedure: always select the column of $\mathbf{C}$ that reduces the reconstruction error most (given all the columns selected in previous steps). The problem is, of course, how to compute the reconstruction error, especially, how to decide how a certain column of $\mathbf{C}$ should be used to minimize the reconstruction error. Knowing this would require us to solve the BU problem exactly. The solution is to use the cover function from Section 3.6.2 to produce an approximation of proper use of the columns.

The columns of $\mathbf{B}$ (i.e. the basis vectors) are selected from the matrix $\mathbf{C}$ and the rows of the matrix $\mathbf{X}$ (some version of which we need to know) are fixed in a greedy manner as follows. Initially $\mathbf{B}$ and $\mathbf{X}$ are empty matrices. The matrix $\mathbf{B}$ is updated in the iteration l by adding the lth column $\mathbf{b}^l$, and matrix $\mathbf{X}$ is updated by adding the lth row $\mathbf{x}_l$. The column $\mathbf{b}^l$ is a column $\mathbf{c}^i$ from $\mathbf{C}$ and the row $\mathbf{x}_l$ is an arbitrary m-dimensional binary row vector. The selection of $\mathbf{b}^l$ and $\mathbf{x}_l$ is done so as to maximize $\mathsf{cover}(\mathbf{A}, \mathbf{B}, \mathbf{X}, w)$ using the cover function defined in (3.7) (page 35). The value of function cover can be considered as the 'profit' of describing $\mathbf{A}$ using the product of matrices $\mathbf{B}$ and $\mathbf{X}$.

Algorithm 2 An algorithm for the BMF using association rules (`Asso`)

```
Input: A matrix $\mathbf{A} \in \{0,1\}^{n \times m}$ for data, a positive integer $k$, a threshold
    value $\tau \in ]0,1]$, and a real-valued weight $w$.
Output: Matrices $\mathbf{B} \in \{0,1\}^{n \times k}$ and $\mathbf{X} \in \{0,1\}^{k \times m}$ such that $\mathbf{B} \circ \mathbf{X}$
    approximates $\mathbf{A}$.
 1: function Asso($\mathbf{A}, k, \tau, w$)
 2:     for $i = 1, \ldots, n$ do    ▷ Construct the association matrix $\mathbf{C}$ column
    by column.
 3:         $\mathbf{c}^i \leftarrow \big(\mathbf{1}(c(i \Rightarrow j, \mathbf{A}) \geq \tau)\big)_{j=1}^{n}$
 4:     end for
 5:     $\mathbf{B} \leftarrow [\,], \mathbf{X} \leftarrow [\,]$                    ▷ $\mathbf{B}$ and $\mathbf{X}$ are empty matrices.
 6:     for $l = 1, \ldots, k$ do              ▷ Select the $k$ basis vectors from $\mathbf{C}$.
 7:         $(i, \mathbf{x}) \leftarrow \arg\max_{i \in \{1,\ldots,n\}, \mathbf{x} \in \{0,1\}^{1 \times m}} \mathsf{cover}(\mathbf{A}, [\,\mathbf{B}\ \mathbf{c}^i\,], \left[\begin{smallmatrix}\mathbf{X}\\ \mathbf{x}\end{smallmatrix}\right], w)$
 8:         $\mathbf{B} \leftarrow [\,\mathbf{B}\ \mathbf{c}^i\,], \mathbf{X} \leftarrow \left[\begin{smallmatrix}\mathbf{X}\\ \mathbf{x}\end{smallmatrix}\right]$
 9:     end for
10:     return $\mathbf{B}$ and $\mathbf{X}$
11: end function
```

The whole algorithm, referred to as `Asso` and combining the computation of association accuracies and the selection procedure, is presented in Algorithm 2.

The association matrix can be constructed in time $O(n^2m)$: computing the association accuracy $c(i \Rightarrow j, \mathbf{A})$ for fixed i and j can be done in time $O(m)$, and there are n^2 pairs of rows $(\mathbf{a}_i, \mathbf{a}_j)$ that need to be considered. Given the matrix $\mathbf{C}$, finding one column of $\mathbf{B}$ and one row of $\mathbf{X}$ can be done in time $O(n^2m)$: there are $O(n)$ columns of $\mathbf{C}$ to try, and for each column $\mathbf{c}$ of $\mathbf{C}$, we can compute the row $\mathbf{x}$ of $\mathbf{X}$ maximizing cover in time $O(nm)$ by setting $x_i = 1$ if and only if $\mathbf{c}$ covers more uncovered 1s than uncovered 0s in column i of $\mathbf{A}$ (cf. Section 3.6.2). Thus, Algorithm 2 has time complexity $O(kn^2m)$.

The algorithm has two parameters that control the quality of results: the threshold τ, and weight w. The straight forward way to set the parameters is to try several different possibilities and take the one that minimizes the reconstruction error. Alternatively the weight w can be used to express different valuations for covering 1s and 0s.

4.6.2 Variations on Finding the Basis

Though the `Asso` algorithm is very simple, it gives results that are comparable to those of more complex methods, as we will see in the next section. In certain occasions we know, however, that our data contains some structure we wish to take into account when computing the decomposition. This section presents a variation of the `Asso` algorithm for such situations.

The proposed variation of the `Asso` algorithm stems from the observation that for data that are easily clustered, the basis vectors should be different among the different clusters. To take advantage of this fact, one can first cluster the data and then solve BMF within each cluster. The twofold goal of this clustered decomposition is, on one hand, to find the features specific for each cluster, and on the other hand, to find the features specific for a group of clusters or the entire data.

The following approach achieves these two goals simultaneously. First, cluster the data and compose the set of candidate basis vectors from association rules within each cluster. This is different from computing the association rules over the whole data, as the association strengths can vary considerably. Second, select the final basis vectors from these candidates one by one. To select a basis vector, first select the cluster that has the most uncovered 1s, and consider the candidate basis vectors created for this cluster. Second, select the basis vector which maximizes the function cover within the selected cluster. Finally, use that basis vector to cover the rest of the data.

In general, the clustering method CLUSTER used in line 2 of Algorithm 3 is not fixed. It should be selected by the user according to her needs. Algorithm 3 finds the basis vector candidates for the clusters in the same way as in `Asso`.

Other variations are, of course, possible. One very simple idea is to transpose the input matrix. The optimal decomposition does not change, but for a heuristic such as `Asso` this can make some difference.

Algorithm 3 A BMF algorithm using clustering (`Asso+clust`).

Input: A matrix $\mathbf{A} \in \{0,1\}^{n\times m}$ for data, a positive integer p for the number of clusters, a positive integer $k \geq p$, a threshold value $\tau \in]0,1]$, and real-valued weight w.

Output: Matrices $\mathbf{B} \in \{0,1\}^{n\times k}$ and $\mathbf{X} \in \{0,1\}^{k\times m}$ such that $\mathbf{B} \circ \mathbf{X}$ approximates $\mathbf{A}$, and row indices for each cluster.

```
1: function AssoClust($\mathbf{A}, p, k, \tau, w$)
2:     $(I_1, \ldots, I_p) \leftarrow$ CLUSTER($\mathbf{A}, p$)    ▷ $I_h$s contain indices for $\mathbf{A}$'s columns.
3:     $\mathbf{A} \leftarrow [\,], \mathbf{X} \leftarrow [\,], \mathbf{B} \leftarrow [\,]$
4:     for $h = 1, \ldots, p$ do
5:         for $i = 1, \ldots, n$ do    ▷ Construct the basis vector candidate matrix $\mathbf{C}_{(h)}$ for each cluster $h$.
6:             $\mathbf{C}^i_{(h)} \leftarrow \big(\mathbf{1}\big(c(i \Rightarrow j, \mathbf{A}^{I_h}) \geq \tau\big)\big)_{j=1}^{n}$
7:         end for
8:         $\mathbf{C} \leftarrow [\, \mathbf{C}\ \mathbf{C}_{(h)}\,]$
9:     end for
10:    for $l = 1, \ldots, k$ do
11:        $h \leftarrow \arg\max_{h=1,\ldots,p} \big|\{(i,j) : (\mathbf{A}^{I_h})_{ij} = 1, (\mathbf{B} \circ \mathbf{X}^{I_h})_{ij} = 0\}\big|$
12:        $(\mathbf{c}^i, \mathbf{x}) \leftarrow \arg\max_{\mathbf{c}^i \in \mathbf{C}_{(h)}, \mathbf{x} \in \{0,1\}^m} \mathsf{cover}\left([\,\mathbf{B}\ \mathbf{c}^i\,], \left[\begin{smallmatrix}\mathbf{X}^{I_h}\\ \mathbf{x}\end{smallmatrix}\right], \mathbf{A}^{I_h}, w\right)$
13:        $\mathbf{B} \leftarrow [\,\mathbf{B}\ \mathbf{c}^i\,], \mathbf{X}^{I_h} \leftarrow \left[\begin{smallmatrix}\mathbf{X}^{I_h}\\ \mathbf{x}\end{smallmatrix}\right]$
14:        $J \leftarrow \{1, \ldots, n\} \setminus I_h$
15:        $\mathbf{x} \leftarrow \arg\max_{\mathbf{x}} \mathsf{cover}(\mathbf{B}, \left[\begin{smallmatrix}\mathbf{X}^{J}\\ \mathbf{x}\end{smallmatrix}\right], \mathbf{A}^J, w)$
16:        $\mathbf{X}^J \leftarrow \left[\begin{smallmatrix}\mathbf{X}^{J}\\ \mathbf{x}\end{smallmatrix}\right]$
17:    end for
18:    return $\mathbf{B}$, $\mathbf{X}$, and $(I_1, \ldots, I_p)$
19: end function
```

4.6.3 Using the Basis

The algorithms from the previous sections, `Asso` and `Asso+clust`, already return some way to use the basis $\mathbf{B}$ (i.e. some $\mathbf{X}$). While this is one possible solution, it is often advantageous (cf. Section 4.8) to try to improve that by using the methods for the BU problem from the previous chapter. The three methods considered here are to use the optimal, exponential-time search; to use the iterative update; and to use the Peleg's algorithm.

The first method is based on the exhaustive search. Recall from Section 3.5 that the BU problem can be solved exactly in time $O(2^k knm)$. Thus, if k is small – as it often is – and n and m are not too big, either, it could be plausible to solve $\mathbf{X}$ exactly. This method is henceforth called `Asso+opt`.

The second method is to update the matrix $\mathbf{X}$ returned by `Asso` using the `IterX` algorithm (Algorithm 1, page 36). The algorithm is called `Asso+IterX`.

Finally, it is possible to compute $\mathbf{X}$ using the `PelegRB` algorithm via the reductions form the previous chapter. But `PelegRB` cannot be seeded with a preliminary $\mathbf{X}$, and thus it has to be computed from scratch. Neither does `PelegRB` understand the weighting parameter w, which is why it must be ignored when calling the algorithm. Following the established naming convention, this algorithm is called `Asso+PelegRB`.

Naturally, any other algorithm to solve the BU problem could be used as well. Indeed, one of the advantages of splitting the BMF problem into subproblems is that the independent improvements in each subproblem have immediate effect on the BU problem. It must also be noticed that the variations from Section 4.6.2 could be paired with these methods, so that one could, for example, first compute `Asso+IterX` and then take the transpose of the data matrix and compute new `Asso+IterX` selecting the better of these two. For the sake of clearness that is not done in this thesis, though.

4.7 Algorithms for BMP

Recall that, due to Theorem 4.5, we can use any existing algorithm for BKM to solve BMP. Moreover, we can use any algorithm $\mathfrak{A}$ that approximates Graph k-Median, where the set of possible medians is restricted to the set of input points, to approximate BKM with approximation ratio at most twice that of $\mathfrak{A}$. Graph k-Median has been the subject of many papers in the course of years, and new results have been obtained recently.

Charikar and Guha [CG99] presented a 4-approximation algorithm for the Graph k-Median problem. This was improved by Arya et al. [AGK$^+$04], who proposed a local-search heuristic for the problem, obtaining a locality gap of 5 with single local swap, and more generally, $3 + 2/p + \varepsilon$ approximation ratio for polynomial number of local steps, each of which can be performed in time $n^{O(p)}$. The *locality gap* of a local-search heuristic is the approximation ratio guaranteed for the algorithm when the maximum number of local improvements is not limited. The maximum number of local improvements made by the algorithm is upper bounded by the ratio M/ε, where M is the maximum error possible for the input and ε is the smallest possible non-zero change in the problem's cost function. In general, this ratio can be exponential with respect to the instance size, but for example with BMP it is upper bounded by nm, the size of the matrix: the maximum error M cannot be more than that, and the minimum improvement ε is 1. It follows that when the algorithm of Arya et al. [AGK$^+$04] is applied to BMP, it has guaranteed approximation ratio of 10 that it achieves in polynomial time, as claimed above. The number of *local swaps* defines the size of the neighbourhood for the local search, and usually, as is the case with the algorithm of Arya et al., the size of the neighbourhood increases exponentially with the number of local swaps.

Building upon the results of Charikar and Guha, Chen [Che06] proposed a method based on core-sets that achieves $(10 + \varepsilon)$ approximation ratio with time $O(nk + k^7\varepsilon^{-4}\log^5 n)$, which is faster than the algorithm by Arya et al. [AGK$^+$04]. Most recently, Ackermann, Blömer, and Sohler [ABS08] provided a universal k-median clustering algorithm that works with those (not necessarily metric) dissimilarity measures that satisfy certain properties. Their algorithm does not restrict the medians to the set of input points, applying directly to BMP. The algorithm provides $(1 + \varepsilon)$ approximation in time $O\left(n2^{(k/\varepsilon)^{O(1)}}\right)$. Notice, however, that their algorithm assumes fixed-dimensional space, and thus its applicability to BMP is questionable from that point of view.

Other recent results include Meyerson et al.'s algorithm [MOP04] with running time independent of the number of input points, and Chrobak et al.'s [CKY06] simple reverse greedy algorithm with $O(\log n)$ approximation.

4.8 Experimental Evaluation

The main goal for the experimental evaluation was to verify that the algorithms work as expected, that is, they find accurate and meaningful decompositions. Synthetic data was used to study the accuracy of the algorithms, and the effects various characteristics of the data have on them. Accuracy with real-world data was studied, but there also the meaningfulness and interpretability of the results was considered. Real-world data was also used to confirm that the behaviour of the algorithms observed with synthetic data was also present with real-world data, instead of being a result of the data generation process.

4.8.1 Algorithms Used

The seven algorithms used in these experiments contained four variations of the `Asso` algorithm, a BMP algorithm and two benchmark algorithms. It should be noted that, to the best of the author's knowledge, this is the first work to address the algorithmic issues for finding approximate Boolean decompositions, and the algorithms from Section 4.6 are the only known algorithms for the problem. Thus, they cannot be directly compared to other algorithms. Instead, results from SVD and NMF algorithms are presented as benchmarks, those being two popular matrix factorization methods. Their results are not directly comparable to the results of `Asso` and its variants, but it could be argued that not being much worse than an SVD or NMF algorithm is already an achievement from a method that is bounded to binary-valued factor matrices. The SVD implementation used was Matlab's build-in command, and the NMF algorithm was based on the alternating least squares approach (see [BBL$^+$07]). The algorithm for SVD is denoted by `SVD`, and the algorithm for NMF is denoted by `NMF`.

The variations of `Asso` differed in the way they selected the matrix $\mathbf{X}$, and the purpose of the experiments was to study the effects different BU algorithms had on the overall performance. In addition to plain `Asso` (with no post-processing to improve $\mathbf{X}$), `Asso+IterX`, `Asso+PelegRB`, and `Asso+opt` were used, the last method using the optimal, $O(2^k knm)$-time exhaustive search.

The BMP problem is not a central problem of this thesis, and given the previous results, solving it is equivalent to solving a k-Medians clustering with binary input. Nevertheless, some experiments were performed to compare the results to the results of BMF algorithms. For the algorithm, the local-search heuristic of Arya et al. [AGK$^+$04] with single local swap was selected.

4.8.2 Measuring the Error

Selecting a correct error measure for comparisons between real-valued and Boolean methods is not a straight forward task. The real-valued methods, for example, `SVD` and `NMF`, try to minimize the squared reconstruction error (i.e. the Frobenius norm), which squeezes the element-wise errors less than 1, but increases those above 1. With the sum-of-absolute-differences distance, the contribution of the small errors is increased, if at the same time that of big errors is decreased. As an example, if `SVD` approximates an element of a binary matrix by saying it is 0.5, the contribution of this to the (squared) Frobenius error is only 0.25, while with the sum-of-absolute-errors the error is 0.5. Furthermore, in the context of approximating a binary matrix, the value 0.5 is equivalent to saying ‘I don’t know’, and is, arguably, almost as bad as an error of 1.

To facilitate the comparison, two approaches are used in this thesis. The first approach is to use the Frobenius norm. This is the natural approach to all real-valued methods, but slightly less natural to binary-valued ones. While minimizing the sum-of-absolute-differences, in case of binary matrices, is equivalent to minimizing the Frobenius, the fact that the smallest possible error is 1 prevents the binary-valued methods on benefiting from the aforementioned properties of the Frobenius norm.

The second approach uses the sum-of-absolute-differences distance. To overcome the problem with real-valued methods, the rounded results of `SVD` and `NMF` are computed as follows: first, the real-valued factor matrices are multiplied together using normal matrix multiplication. The resulting matrix (a_{ij}) is then rounded to matrix (a'_{ij}) by setting $a'_{ij} = 0$ if $a_{ij} < 0.5$ and $a'_{ij} = 1$ otherwise. The SVD and NMF algorithms with rounding are referred to as $\mathtt{SVD}_{0/1}$ and $\mathtt{NMF}_{0/1}$, respectively.

Both of these approaches can be seen to benefit the real-valued methods, albeit in different situations. While a totally fair method would definitely be better, the lack of such methods justifies using the other comparison methods which are fair in the sense that at least they are not presenting too optimal view of the proposed algorithms.

Notice that BMF is a different problem than SVD or NMF. Thus, a straight forward comparison between these methods and their results is not possible, and all conclusions based on the comparisons should be taken with a grain of salt.

4.8.3 Synthetic Data

The data and parameters used were the same as in Section 3.7, except this time the algorithms were only given the matrix $\mathbf{A}$, and they had to find both matrices $\mathbf{B}$ and $\mathbf{X}$. The four parameters whose effects were studied were the same, too: (*1*) the number of columns in $\mathbf{B}$; (*2*) the noise level; (*3*) the density of the columns in $\mathbf{B}$; and (*4*) the mean number of columns of $\mathbf{B}$ used to create each of the columns of the data. For details of the data generation process, see Section 3.7.1.

Only algorithms for BMF and benchmark algorithms were used (i.e. no BMP algorithms were used). The results are presented below.

Number of columns in B. The number k of columns in $\mathbf{B}$ varied from 8 to 28 with steps of size 2. The total density of the resulting matrices was approximately 0.35.

The results are presented in Figure 4.1. Notice first that `SVD` and `NMF` have reconstruction errors below the errors of the other methods. This means that `Asso` and its variations were not fully able to take advantage of the Boolean algebra. Increasing k slightly increases the error of `Asso` and its variations; on the other hand, `SVD` (and $\mathtt{SVD}_{0/1}$) seems to improve with larger values of k (this was actually expected, as the matrices had full real rank).

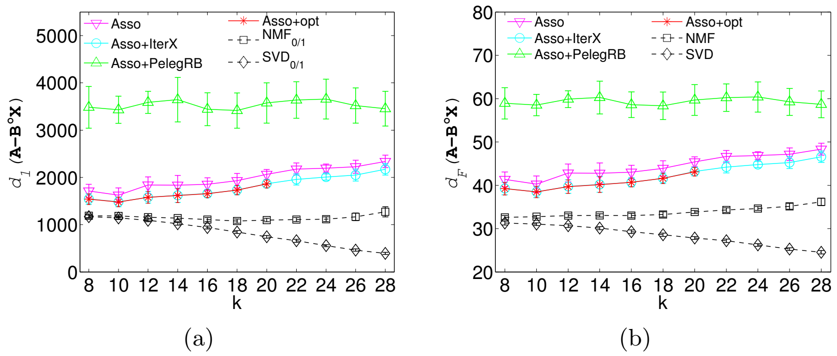


Figure 4.1: Reconstruction errors of BMF decomposition when k, the number of columns in $\mathbf{B}$, varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is normal for `SVD` and `NMF`, and Boolean for the other methods.

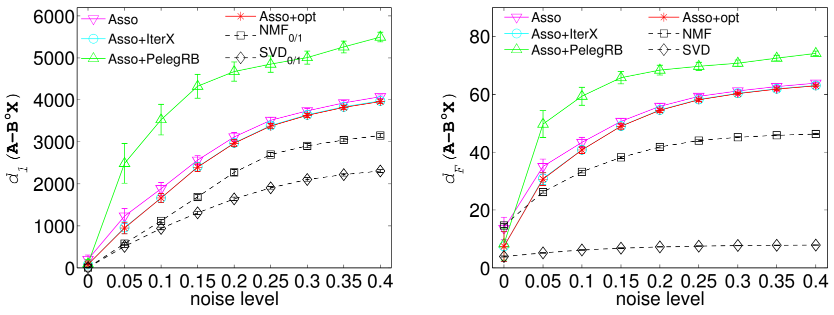
For the proposed methods, `Asso+IterX` is always as good as `Asso+opt`, that is, optimal with the given $\mathbf{B}$, and even the plain `Asso` is very close to those other two. `Asso+PelegRB` is clearly the worst in both cases (recall that `PelegRB` does not use the already-computed $\mathbf{X}$ from `Asso`).

The difference between sum-of-absolute-differences (Figure 4.1(a)) and Frobenius distance (Figure 4.1(b)) is very small: only the relative performance of `SVD` improves considerably against the other methods when Frobenius norm is used.

Noise level. The level of noise varied from 0 to 0.4 with steps of size 0.05. The results are presented in Figure 4.2. All algorithms follow essentially the same, intuitive trend. Again, `SVD` (or `SVD`$_{0/1}$) is the best, but with zero noise, all BMF algorithms are better than `NMF` with Frobenius distance (Figure 4.2(b)). Also, with zero noise, `Asso+PelegRB` is better than plain `Asso`, but after that it is worst of all methods with both error measures.

Similar to previous experiments, `Asso+IterX` is always as good as `Asso+opt`, and – except in the zero noise case – plain `Asso` is very comparable to these two. The variance of all methods was very small with all values of noise greater than 0.05.

Density of columns of B. The mean density (number of 1s divided by the total number of elements) of $\mathbf{B}$'s columns varied from 0.05 to 0.3. The total density of the resulting matrices varied from



(a) (b)

Figure 4.2: Reconstruction errors of BMF decomposition when the level of noise varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is normal for `SVD` and `NMF`, and Boolean for the other methods.

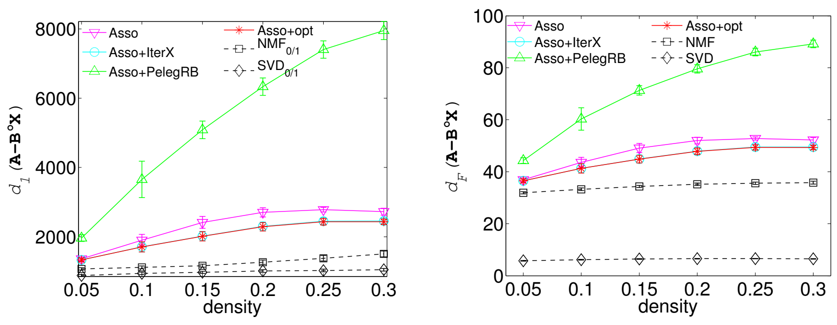
approximately 0.2 to approximately 0.7. The results can be seen in Figure 4.3.

Density does not seem to have any effects on `SVD` or `NMF`. The results of `Asso`, `Asso+IterX`, and `Asso+opt` decline slightly as density increases to 0.2, but then they seem to improve slightly. As was expected, the results of `Asso+PelegRB` get worse as density increases. The variance is negligible in almost all cases.

Mean number of columns of $\mathbf{B}$ used for each column of $\mathbf{A}$. The mean number of columns of $\mathbf{B}$ involved in the Boolean combination to create columns of $\mathbf{A}$ (i.e. mean number of 1s in columns of $\mathbf{X}$) had three possible values, 4, 6, and 8. The resulting matrices had an approximate total density between 0.35 and 0.55. The results, seen in Figure 4.4, follow those of `IterX` and `PelegRB` in a similar experiment (see Figure 3.2(b)): the quality of `Asso+PelegRB`'s results decreases, while other methods are almost immune to it having virtually no variance.

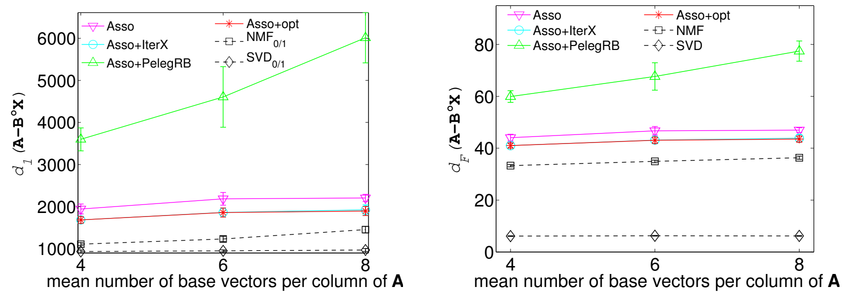
4.8.4 Real-World Data Sets

Several real-world data sets were used to study the performance of the algorithms. A detailed description of each data set, its source, and possible preprocessing steps taken is given below and some of the characteristics of the data sets are summarized in Table 4.1.



(a) (b)

Figure 4.3: Reconstruction errors of BMF decomposition when the density varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is normal for `SVD` and `NMF`, and Boolean for the other methods.



(a) (b)

Figure 4.4: Reconstruction errors of BMF decomposition when the mean number of **B**'s columns used for each column of **A** varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is normal for `SVD` and `NMF`, and Boolean for the other methods.

Table 4.1: Some characteristics of the real-world data sets. Average numbers of 1s are rounded to the nearest integer.

Data	Rows	Columns	Density (%)	Average # of 1s	
				per row	per column
Abstracts	4 894	12 841	0.9	115	44
DBLP	19	6 980	12.95	904	2
Dialect	1 334	506	16.14	82	215
20Newsgroups	5 163	19 997	0.89	177	46
NOW	124	139	11.48	16	14

The Abstracts data. This data[2] is a collection of project abstracts submitted for funding by National Science Foundation of the USA. The data is represented in a bag-of-words format, and a sample of the data, instead of the full data, is considered here. The data was preprocessed by first stemming the words using Porter stemmer[3] [Por80]. Then, all words occurring in less than 10 or in more than 999 sampled abstracts were removed. Only the presence/absence information was saved, omitting the word counts, to achieve a binary matrix. The resulting data has $12,841$ abstracts and 4894 words.

The DBLP data. This data is collected from the DBLP article database[4], and it contains conference–author information. More precisely, it contains 19 hand-picked conferences and 6980 authors. The conferences, namely WWW, SIGMOD, VLDB, ICDE, KDD, SDM, PKDD, ICDM, EDBT, PODS, SODA, FOCS, STOC, STACS, ICML, ECML, COLT, UAI, and ICDT, were selected to present various, yet slightly overlapping fields of computer science, and the authors are all of the authors with in total at least 2 publications in the selected conferences. The resulting matrix is very flat and wide.

The Dialect data. This data contains 1334 features of spoken dialect collected from 506 Finnish municipalities [EW97, EW00]. Each municipality is additionally characterized by two spatial coordinates, but these were only used in plotting the results. The suggested division of Finnish dialects is given in Figure 4.5.

[2]http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html
[3]http://tartarus.org/~martin/PorterStemmer/perl.txt
[4]http://www.informatik.uni-trier.de/~ley/db/

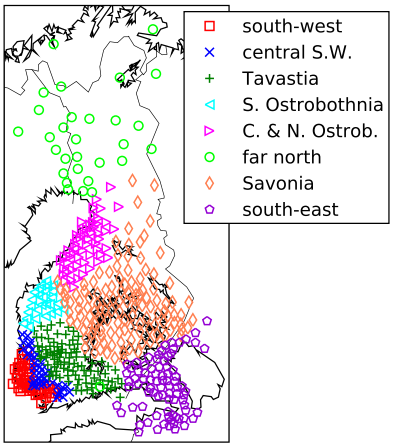


Figure 4.5: The division of Finnish dialects [EW97, EW00].

The 20Newsgroups data. This data[5] is a bag-of-words representation of almost 20,000 Usenet news posts spread (almost) evenly in 20 newsgroups. As with the Abstracts data, the data was first stemmed using Porter Stemmer, and all words appearing in less than 36 or in more than 999 posts were removed. This data set was only used in this chapter; for subsequent chapters, a different preprocessing was applied to yield a smaller data set.

The NOW data. This data[6] contains information about species' fossils found in specific palaeontological sites in Europe [F$^+$03]. The data is preprocessed following Fortelius et al. [FGJM06].

4.8.5 Randomization of the Data

A simple reconstruction error, even if it is small, does not necessary guarantee that the data contains the latent structure we were trying to find. It is possible that the results were obtained by chance. The standard method to distinguish these cases is to measure the *statistical significance* of the results against the null hypothesis, which – broadly speaking – is 'results were obtained by chance'.

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

[6]NOW public release 030717, available from `http://www.helsinki.fi/science/now/`.

The general (randomization) framework for testing the significance is the following. First, generate multiple instances of (random) data that satisfy the null hypothesis. Then, apply the algorithm whose results are studied to the generated data and measure the error. If, say, 99% of the results obtained with the random data are worse than the results with the original data, the null hypothesis can be abandoned and the original results are considered statistically significant (at significance level 99%) (see, e.g. [GMMT07]).

There are two problems on applying this approach: how to define the null hypothesis and how to generate the data agreeing with it. In general, one would like the null hypothesis to be complex in the sense that it rules out any simpler explanation of the data than the one sought. Naturally, also the original data must agree with the null hypothesis for the test to be meaningful. An example of a typical null hypothesis is 'the results are due to the density of the data matrix'. But density is a very low-level characteristic, and being able to rule out that null hypothesis does not guarantee that there is not some more complex, yet uninteresting, feature of the data that explains the good results.

On the other hand, the data-generation process is not usually known, complex null hypotheses are hard to define, or the testing method is required to be applicable to different kinds of data. All these limit the possible complexity of the null hypothesis. Moreover, there is very important practical limitation: generating truly random data satisfying complex null hypotheses can be very hard. A null hypothesis which cannot be tested is not very useful. Simpler null hypotheses do not usually have this problem, for example generating random binary matrices with (approximately) same density is easy.

Thus, a compromise between the complexity of the null hypothesis and the hardness of generating random data satisfying it is sought, and here the compromise comes in the form of *marginal sums* of the binary matrix. The marginal sums of an n-by-m binary matrix $\mathbf{A} = (a_{ij})$ are given by a pair of vectors, $(\mathbf{c}, \mathbf{r})$. Vector $\mathbf{c}$ contains the sums over $\mathbf{A}$'s columns, that is, $c_j = \sum_{i=1}^{n} a_{ij}$, and vector $\mathbf{r}$ contains the sums over $\mathbf{A}$'s rows (i.e. $r_i = \sum_{j=1}^{m} a_{ij}$). Given a pair $(\mathbf{c}, \mathbf{r})$, define $\mathcal{M}(\mathbf{c}, \mathbf{r})$ to be the set of all n-by-m binary matrices such that $(\mathbf{c}, \mathbf{r})$ gives their marginal sums. The task of generating a uniform sample from this set can be done by performing *swap randomizations* (see, e.g. [CC03]). To generate the swap randomizations, an MCMC method by Gionis et al. [GMMT07] was used. As with any Markov chain, it requires enough iterations to be fully mixed. In all experiments the number of iterations was $10000nm$ for n-by-m data matrix. The chain was re-initialized for each random sample.

Hence the procedure of checking the statistical significance of the results is as follows. First, a *seed matrix* is created. Then, a predefined number of random matrices are generated, initializing the Markov chain every time with the seed matrix. The purpose of this back-and-forth process is to make sure that the original matrix is reachable from the same seed matrix as the random matrices with the same number of iterations. Then the `Asso` algorithm (or some other algorithm) is executed on the original and each of the random matrices. Finally, the reconstruction errors are compared, and if the reconstruction error of the original matrix is smaller than the error of 99% of the random matrices, the result is considered significant.

The results for randomizations are presented below, after the reconstruction errors.

4.8.6 Results on Real-World Data

The purpose of the experiments with the real-world data was, first, to verify that the algorithms can achieve comparable results also with real-world data, and second, to verify that the results returned by the algorithms are intuitive and reveal interesting information about the data.

Due to its poor performance with synthetic data, `Asso+PelegRB` was not used with real-world data. Also `Asso+opt` was excluded from the list of algorithms – not because of its poor performance, but because of its practically infeasible running time with larger data sets.

The results are reported as follows. First, the reconstruction errors for BMF algorithms and benchmark algorithms (i.e. `NMF` and `SVD` and their variations) are given. Then, these results are compared against the results obtained from randomized data sets, following experiments and discussion about the interpretability of the results by BMF algorithms. Finally, some examples about the quality and interpretability of the results obtained using the local-search heuristic of Arya et al. [AGK$^+$04] for the BMP problem are studied.

Table 4.2: Reconstruction errors from real-world data sets, d_1 distance.

Data	k	`Asso`	`Asso+IterX`	$\texttt{NMF}_{0/1}$	$\texttt{SVD}_{0/1}$
Abstracts	5	559 436	559 436	563 248	563 336
	10	554 569	554 569	561 476	561 503
	15	549 780	549 780	558 703	559 011
DBLP	5	8 676	8 676	8 623	8 384
	10	4 503	4 503	4 121	3 863
	15	1 519	1 519	1 087	988
Dialect	5	58 683	55 057	43 194	40 846
	10	48 967	41 649	32 409	28 136
	15	45 231	35 825	26 343	22 379
20Newsgroups	5	902 585	902 582	905 003	904 293
	10	896 389	896 388	897 896	896 984
	15	891 564	891 540	894 237	892 147
NOW	5	1 573	1 569	1 421	1 379
	10	1 407	1 396	1 168	1 020
	15	1 278	1 272	971	760

Reconstruction errors. The reconstruction errors for real world data are given in Tables 4.2 (for d_1 distance) and 4.3 (for d_F distance).

With d_F distance, `SVD` is definitely the best method with `NMF` being close to it. `Asso` and `Asso+IterX` are much worse. One must remember, however, that with Boolean decomposition the smallest error one can make in each element is 1. This has the tendency of increasing the squared error, especially when compared to those methods that can take advantage of the properties of squaring, that is, that errors less than 1 are decreased even further.

When d_1 distance is used, results change slightly. Again, $\texttt{SVD}_{0/1}$ and $\texttt{NMF}_{0/1}$ are often better than `Asso` or `Asso+IterX`, although this time the difference is not so striking. But with Abstracts and 20Newsgroups, `Asso` and `Asso+IterX` are better than $\texttt{NMF}_{0/1}$ or even $\texttt{SVD}_{0/1}$ (in fact, with Abstracts and $k = 15$, $\texttt{NMF}_{0/1}$ is better than $\texttt{SVD}_{0/1}$). As the results change dramatically with d_F distance, this suggests that `NMF` and `SVD` err in many elements of Abstracts and 20Newsgroups by slightly more than 0.5, yielding larger rounded d_1 error, but still keeping the Frobenius error small.

Comparing `Asso` and `Asso+IterX` to real-valued decompositions arguably has its problems, but comparing them to each other is

Table 4.3: Reconstruction errors from real-world data sets, d_F distance.

Data	k	Asso	Asso+IterX	NMF	SVD
Abstracts	5	747.95	747.95	727.48	726.60
	10	744.69	744.69	721.25	719.28
	15	741.47	741.47	717.23	713.95
DBLP	5	93.15	93.15	81.18	81.01
	10	67.10	67.10	58.69	58.27
	15	38.97	38.97	33.67	33.11
Dialect	5	242.25	234.64	179.93	177.72
	10	221.28	204.08	157.60	152.09
	15	212.68	189.27	146.16	137.53
20Newsgroups	5	950.04	950.04	920.27	919.12
	10	946.78	946.78	913.13	910.62
	15	944.23	944.21	908.25	904.37
NOW	5	39.66	39.61	32.23	31.93
	10	37.51	37.36	29.65	28.46
	15	35.75	35.67	27.90	25.84

straight forward. We know that Asso+IterX will always do at least as well as Asso, but whether it will give any added accuracy to the decompositions is still an open question. The synthetic experiments suggested that it should. And so it seems to do, but only in remarkably few cases, and even then usually very little. From Table 4.2 one can see that IterX aids Asso considerably only with Dialect data. There are some improvements also with 20Newsgroups and NOW, but they are basically insignificant, and with Abstracts and DBLP the Asso algorithm has already reached the local optimum. It thus seems that improving $\mathbf{X}$ with additional methods (at least, with IterX) is not usually necessary, although with some cases it can provide considerable improvements.

Randomization results. Randomized versions of data sets were used to assess the significance of the results, as described in Section 4.8.5. In total, 100 random versions of each data set, excluding 20Newsgroups, were created. The 20Newsgroups data was excluded due to its size: running experiments on 100 copies of it would have taken too much time. The algorithms used were Asso, Asso+IterX, $\mathtt{NMF}_{0/1}$, and $\mathtt{SVD}_{0/1}$, and the measure of error was the d_1 distance. For Abstracts data, only Asso and $\mathtt{SVD}_{0/1}$ were used: the results of Asso+IterX were exactly the same as those of Asso

Table 4.4: Percentage of random data sets that gave smaller d_1 error than the original data. 100 random data sets were used. Missing results are denoted by —.

Data	k	Asso	Asso+IterX	$\texttt{NMF}_{0/1}$	$\texttt{SVD}_{0/1}$
Abstracts	5	2	—	—	0
	10	1	—	—	0
	15	0	—	—	0
DBLP	5	2	2	2	2
	10	1	1	1	1
	15	0	0	0	0
Dialect	5	2	2	2	2
	10	1	1	1	1
	15	0	0	0	0
NOW	5	1	0	1	1
	10	0	0	0	0
	15	0	0	0	0

with that data, and $\texttt{NMF}_{0/1}$ was unable to finish in reasonable time (roughly one week). The results are presented in Table 4.4.

In short, all results can be deemed significant with confidence level 95%, and all results with $k \geq 10$ are significant with confidence level 99%. In other words, all four algorithms, Asso, Asso+IterX, $\texttt{NMF}_{0/1}$, and $\texttt{SVD}_{0/1}$, were able to find such structures in the data sets that are highly unlikely to be a consequence of the marginal sums of the data matrices.

Intuitiveness of the results. The Abstracts, 20Newsgroups, and Dialect data sets were used to examine the interpretability Asso's results. As the terms in the first two data sets were stemmed, so were the words in the respective results. To increase the readability, the words are returned to their original singular form whenever that form is clear.

Intuitiveness and accuracy do not come hand in hand, and the parameters giving lowest reconstruction error are not necessary those giving the most intuitive results. The parameter w, used in function cover to weight the different types of errors, is a good example of this. To achieve a good reconstruction error, setting $w = 1$ is usually required (and this is what is done above), but especially with very sparse datasets some other value, typically $w > 1$, can give more intuitive results.

For Abstracts, the parameters used were $\tau = 0.3$ and $w = 6$. The rows of the data represent words, and thus the columns of $\mathbf{B}$ represent sets of words. Studying the intuitiveness and interpretability of these results is done by studying the columns of $\mathbf{B}$, and seeing if the words in sets form a meaningful concept. Examples of these sets of words are given below. For some sets, only a subset of words is presented, and this is indicated by an ellipsis. The sets are:

1. fund, NSF, year
2. cell, gene, molecular, protein, ...
3. gopher, Internet, network, world, wide, web, ...
4. behaviour, effect, estim, impact, measure, model, overestimate, predict, test, ...
5. course, education, enrol, faculty, institute, school, student, undergraduate, ...
6. abstract, don, set.

Many of the sets found consisted of words typical to a specific field of science; examples 2 and 3 are of this type. Some of them were of more general, for example, words used in science in general (example 4), or words referring to education (example 5). Due to the nature of the data, words related to money and funding were also found (example 1). Despite the fact that most of the resulting sets of words were very natural, also less natural ones were found; example 6 illustrates this behaviour.

For 20Newsgroups, no specific pair of parameters was better than others, as most of the results were very intuitive. The examples given here are obtained setting $\tau = 0.4$ and $w = 4$, leading to somewhat large sets of words. Examples are:

1. disk, FAQ, FTP, newsgroup, server, Sun, UNIX, Usenet, ...
2. Bible, Christ, church, faith, Jesus, Lord, Paul, scripture, ...
3. playoff, team, win
4. agent, BATF, cult, FBI, fire, Koresh, Waco, ...

5. Arab, Israel, Jew, land, Palestinian, war, ...
6. American, announce, campaign, crime, federal, fund, office, policy, president, ...
7. Armenia, Azerbaijan, border, defend, military, Soviet, war, ...
8. earth, orbit, space.

First example is clearly computer-related, second contains religious words, specific for Christianity, and third is about sports. Finding such basis vectors could hardly be regarded as a surprise, as the 20Newsgroups data contains many newsgroups in each of these subjects.

To understand the fourth example, one needs to have some knowledge of the recent history of the United States. All words are related to the so-called Waco siege in 1993, where FBI and BATF raided a religious group led by David Koresh in Waco, Texas.

Examples 5 and 7 are partly overlapping; word 'war' appears in both of them. Yet, the wars are different ones. Example 5 clearly points to Israel–Palestinian conflicts, while example 7 is about Armenia and Azerbaijan, neighbours and former Soviet republics.

Example 6 includes words usually seen in political contexts while the last example is from a space-related newsgroup.

In summary, `Asso` was able to find decompositions that were intuitive and informative. For example, there were no prior information telling that the Waco siege was discussed in some newsgroup. Yet, it came out very strongly in the results, indicating that it had been a 'hot topic' for some time, at least. Among the informative and intuitive results there were also less intuitive and less informative ones. Careful selection of the algorithm's parameters helps reducing the number of unintuitive results, but they cannot be totally avoided. This should not be a problem, as there are always meaningful and interesting basis vectors present in the answers, too.

For Dialect data, the balancing variable w was set to 1. The linguistic studies divide Finnish into 8 main dialects, namely south-western dialects, central south-western dialects, Tavastian dialects, South Ostrobothnian dialects, Central and North Ostrobothnian dialects, the dialects of the far north, Savonian dialects, and south-eastern dialects [Itk65]. While this division has been rather stable,

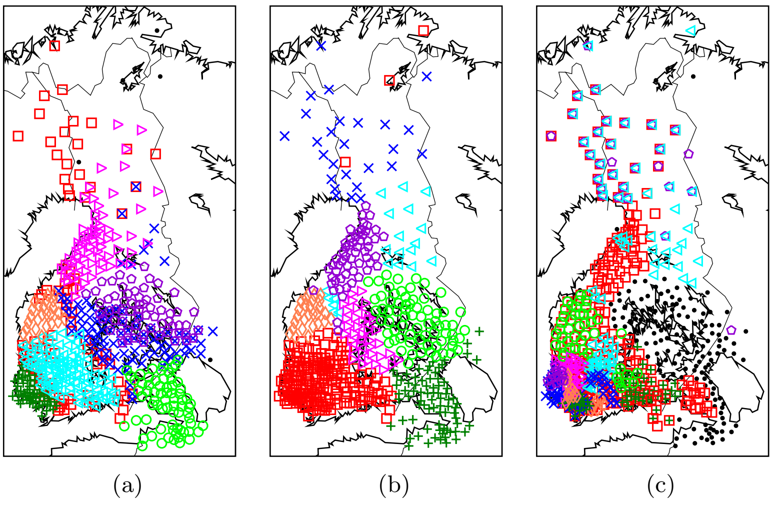


Figure 4.6: (a) The division of municipalities reported by `Asso+IterX` with $k = 8$. (b) The eight clusters used in `Asso+clust`. (c) The division of municipalities reported by `Asso+clust` with $k = 16$. Black dots represent municipalities that are not reported to belong to any dialect.

some minor changes have taken place during the past half a century (cf. [Itk65, p. 31] and [SYL94, p. 19]). The dialect regions, following Itkonen [Itk65], are presented in Figure 4.5.

When decomposing this data with `Asso+IterX`, the columns represented the municipalities and k was set to 8. That is, the goal was to find eight sets of dialectical features in $\mathbf{B}$ such that these sets would reveal which dialect was spoken in which municipality. One could infer that dialect i was spoken in municipality j if $x_{ij} = 1$. The division of dialects obtained by this method is presented in Figure 4.6(a). There are certain differences between the results of `Asso+IterX` and the proposed division (e.g. splitting the Savonian dialects into two), meaning that `Asso+IterX` was not able to fully agree with the linguistics' model. However, its results certainly follow the main trends of the known dialect borders, and it would be interesting to further study the algorithm's results to see if they contain some interesting linguistic results.

The division between dialects is not a strict one, and the exact borders between two dialects are usually more or less arbitrary. Nevertheless, the `Asso+clust` algorithm was used with this data.

Recall that `Asso+clust` works in two phases: at first phase it clusters the data, and at the second phase it construct the basis vectors. For the first phase, the k-Means algorithm from SOM Toolbox[7] was used. Its results for eight clusters are presented in Figure 4.6(b). We can see that the south-western dialects and Tavastian dialect are all combined into one dialect, while Savonian dialect is divided into three. With such a starting point, it is not surprising that `Asso+clust` cannot produce the original ground truth. `Asso+clust` was run with $k = 16$, that is, on average 2 basis vectors per cluster. It does not have to divide the basis vectors equally, though – and indeed, that was not done. The dialect boundaries returned by `Asso+clust` can be seen in Figure 4.6(c): no set of linguistic features (i.e. basis vector) contains the features on Savonian dialect. Instead, the huge cluster of south-western and Tavastian dialects is split into many small, partly overlapping, regions. This splitting is somewhat intuitive, as the big cluster contains very heterogeneous dialects that presumably share very few features, and while the results in Figure 4.6(c) do not resemble those in Figure 4.5, they do show that the `Asso+clust` algorithm can provide interesting information about the data, and something that cannot be obtained with pure clustering methods.

Results for BMP. The algorithm used for the BMP problem was Arya et al.'s local-search algorithm [AGK$^+$04] (henceforth referred to as `AryaLocal`). The results presented here serve as examples of the differences between BMF and BMP, and do not provide a throughout comparison between the respective algorithms.

Only two domains are considered, namely, 20Newsgroups and Dialect, and the reconstruction errors obtained by `AryaLocal` are compared against those obtained by `Asso+IterX`. Results are presented in Table 4.5.

We can see that, with $k = 5$, the `AryaLocal` algorithm finds a slightly better decomposition for the Dialect data than `Asso+IterX`. Yet, with the Dialect data and higher values of k, `Asso+IterX` performs considerably better than `AryaLocal`. This was expected, for example, on the basis of inequality (4.9). Moreover, with the 20Newsgroups data, the `AryaLocal` algorithm is clearly inferior to the `Asso+IterX` algorithm.

[7] `http://www.cis.hut.fi/projects/somtoolbox/`

Table 4.5: Reconstruction errors from real-world data sets, d_1 distance.

Data	k	`AryaLocal`	`Asso+IterX`
Dialect	5	55 029	55 057
	10	42 840	41 649
	15	36 060	35 825
20Newsgroups	5	1 053 752	902 582
	10	1 042 428	896 388
	15	1 035 561	891 540

Also the intuitiveness of `AryaLocal`'s results with 20Newsgroups data left a lot to hope for: most of the words in a single partition did not have any meaningful relation. An example of partition with some meaning in it was a partition containing words 'George', 'Bush', 'Jimmy', 'Carter', 'president', 'Arab', 'Jew', and 'Israel', but this was the best the algorithm could do.

The poor performance of `AryaLocal` with 20Newsgroups data was expected: after all, the idea of partitioning all terms in documents does not fit well with the fact that many terms are used in many documents, and even other terms are homonyms, meaning that they have a different meaning in different contexts.

A data set that is seemingly better suited for the context of BMP is the Dialect data; this idea is emphasized by the relatively good performance of `AryaLocal` with Dialect data in Table 4.5. The dialects can be considered to be partly overlapping, but traditionally the boundaries have been strict and some dialect has been used in every municipality. Thus, the municipalities could be partitioned according to which dialect is used in which municipality. This was done using `AryaLocal` with $k = 7$ and $k = 8$, and results are presented in Figure 4.7.

The results show that, indeed, `AryaLocal` is able to find meaningful partition of the municipalities. Comparing to the ground truth (Figure 4.5), one can see that the major differences are the introduction of a new partition in central South Finland, combination of central south-western dialect with the Tavastian one, and, in case of $k = 7$, combination of far north and Northern Ostrobothnian dialects. With $k = 8$, the results are otherwise similar, except the division of the northern partition into two.

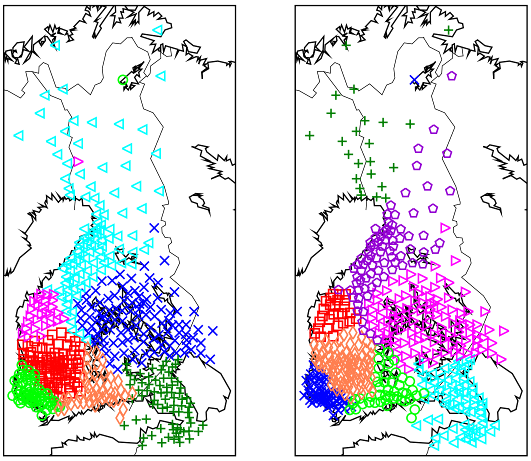


Figure 4.7: The partition of municipalities to different dialect regions by `AryaLocal` with (a) $k = 7$ and (b) $k = 8$.

In short, `AryaLocal`'s results were mostly what was expected: in reconstruction errors, it was not as good as `Asso+IterX`, given the greater freedom the latter has in constructing the decomposition, and from the intuitiveness point of view, it performed well only with data that assumed some partitioned structure.

CHAPTER 5

The Nonnegative Column and Column-Row Decomposition Problems

Where the idea of CX *and* CUR *decompositions is motivated (with and without the nonnegativity constraint) and the convex cone interpretation of the nonnegative* CX *decomposition is discussed. Algorithms for both problems are proposed and compared against prior algorithms for* CX *and* CUR.

5.1 Problem Definitions

The problems considered in this chapter are no longer restricted to the Boolean world, but use the space of nonnegative real numbers, $\mathbb{R}_{\geq 0}$. Nevertheless, the goal is still to create an interpretable matrix decomposition. In the decompositions studied in this chapter, the input matrix is assumed to be in $\mathbb{R}_{\geq 0}^{n \times m}$, the inner dimension of factor matrices, k, is given as an input, and the factor matrices are also required to be nonnegative. As an additional requirement, the left-hand factor matrix is required to be a subset of the input matrix's columns. Also, in the other decomposition discussed in this chapter where three factor matrices are used, the rightmost factor matrix is required to be a subset of the input matrix's rows. These decompositions are called NCX and NCUR, respectively, and they are defined below.

Nonnegative Column Decomposition Problem (NCX)**.** Given a nonnegative matrix $\mathbf{A} \in \mathbb{R}_{\geq 0}^{n \times m}$ and a positive integer k, find k columns of $\mathbf{A}$, constituting the matrix $\mathbf{C}$, and a nonnegative matrix $\mathbf{X} \in \mathbb{R}_{\geq 0}^{k \times m}$ such that $\mathbf{C}$ and $\mathbf{X}$ minimize

$$\mathsf{cost}_{\text{NCX}}(\mathbf{A}, \mathbf{C}, \mathbf{X}) = d_F(\mathbf{A} - \mathbf{C}\mathbf{X}). \tag{5.1}$$

The NCUR problem, using three factor matrices, is defined as follows.

Nonnegative Column-Row Decompositiom Problem (NCUR)**.** Given a nonnegative matrix $\mathbf{A} \in \mathbb{R}_{\geq 0}^{n \times m}$ and positive integers k_c and k_r, find k_c columns of $\mathbf{A}$, constituting the matrix $\mathbf{C}$, k_r rows of $\mathbf{A}$, constituting the matrix $\mathbf{R}$, and a nonnegative matrix $\mathbf{U} \in \mathbb{R}_{\geq 0}^{k_c \times k_r}$ such that $\mathbf{C}$, $\mathbf{U}$, and $\mathbf{R}$ minimize

$$\mathsf{cost}_{\text{NCUR}}(\mathbf{A}, \mathbf{C}, \mathbf{U}, \mathbf{R}) = d_F(\mathbf{A} - \mathbf{C}\mathbf{U}\mathbf{R}). \tag{5.2}$$

In the general counterparts of NCX and NCUR, called CX and CUR problems, the input matrix can contain arbitrary real values, as can matrices $\mathbf{X}$ and $\mathbf{U}$, too.

The NCX problem can be divided into two subproblems, first asking for the matrix $\mathbf{C}$ minimizing the reconstruction error, and the second asking for the matrix $\mathbf{X}$ given some $\mathbf{C}$. The latter problem is well-known constrained least-squares fitting problem (see Section 5.4.1; however, cf. this to connections between the BU problem and Boolean CX decompositions in Chapter 6). The former problem is known as the Column Subset Selection (CSS) problem [BMD09], or, if the nonnegativity constraint is applied as in NCX, as Nonnegative Column Subset Selection (NCSS) problem. Notice that in NCSS only matrix $\mathbf{C}$ is asked, and if $\mathbf{A}$ is nonnegative, so is any submatrix of it; thus, the nonnegativity constraint is only implicitly assumed when considering how good a matrix $\mathbf{C}$ is.

The computational complexity of CSS is unknown [BMD09], the same being the case with NCSS. Indeed, it is not known whether the nonnegativity constraint changes anything, but it seems possible it does not.

5.1.1 Alternative Interpretation of CX and NCX

The CX and NCX problems have alternative interpretations using subspaces and projections, much akin to the interpretations of SVD and PCA. Considering the columns of matrices as points in n-dimensional space, the goal of the CX problem is to select k of these columns such that when we project the matrix $\mathbf{A}$ to the subspace spanned by these columns, the projected matrix is as close to the original one as possible (in terms of the Frobenius distance, that is).

The nonnegativity requirement of $\mathbf{X}$ in NCX means that we cannot consider the whole subspace spanned by $\mathbf{C}$. Instead, we consider the *convex cone* generated by the columns selected in $\mathbf{C}$. A (pointed) convex cone is a set C of vectors such that if $\mathbf{v}, \mathbf{w} \in C$, then $\alpha\mathbf{v} + \beta\mathbf{w} \in C$ for all $\alpha, \beta \in \mathbb{R}_{\geq 0}$. A convex cone C *generated* by the k columns of a matrix $\mathbf{C}$ is the smallest convex cone such that $\mathbf{c}^i \in C$ for $1 \leq i \leq k$. The goal of the NCX problem is to find $\mathbf{C}$ such that, when $\mathbf{A}$ is projected to the convex cone generated by $\mathbf{C}$, the projected matrix is as close to the original as possible (again, in the terms of the Frobenius distance).

In both problems, the ith column of $\mathbf{A}$ is projected to $\mathbf{C}\mathbf{x}^i$ which is either a point of a subspace (CX) or of a convex cone (NCX).

Example 5.1. Consider a matrix $\mathbf{A}$ defined as

$$\mathbf{A} = \begin{pmatrix} 0.6 & 0.9 & 0.6 & 0.4 & 0.7 \\ 1.0 & 0.7 & 0.9 & 1.0 & 0.9 \\ 0.6 & 0.5 & 0.2 & 0.4 & 1.0 \end{pmatrix}.$$

For an NCX decomposition with $k = 3$ we could select the first three columns. These three columns, interpreted as points in 3D space, generate a convex cone $C = \{\alpha\mathbf{a}^1 + \beta\mathbf{a}^2 + \gamma\mathbf{a}^3 : \alpha, \beta, \gamma \in \mathbb{R}_{\geq 0}\}$. The squared reconstruction error $d_F(\mathbf{A} - \mathbf{C}\mathbf{X})^2$ equals to the sum of the squared distances from the columns of $\mathbf{A}$ not in $\mathbf{C}$ to the nearest points in this cone, that is,

$$d_F(\mathbf{A} - \mathbf{C}\mathbf{X})^2 = \sum_{\mathbf{a}^i \notin \mathbf{C}} \min_{\mathbf{v} \in C} \|\mathbf{a}^i - \mathbf{v}\|_2^2.$$

For example, the point of C closest to the fourth column of $\mathbf{A}$ is $0.6\mathbf{a}^1 + 0.3\mathbf{a}^3$.

The convex cone generated by the first three columns of $\mathbf{A}$ is presented in Figure 5.1. Red circles are the generating vectors, and the blue circles are the remaining two columns. The black lines show where in the cone the blue points are projected to. $\diamond$

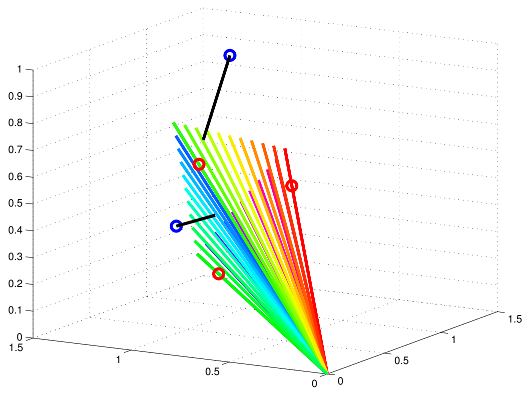


Figure 5.1: An example of interpreting NCX as a convex cone. Red points are the generating vectors, blue points are the other vectors of the matrix, and black lines show where in the cone the blue points are projected to.

5.2 Motivation as Data-Mining Techniques

Interpretability is the key term when NCX and NCUR decompositions are considered as data-mining techniques. The standard CX and CUR decompositions can be used as data mining techniques, and the interpretability of the results is one of the main reasons to do so. The usual problem with matrix decompositions in data mining is that the factor matrices can turn out to be hard to interpret. Boolean Matrix Factorization addressed this problem by requiring the factor matrices to be binary, much in the same lines as NMF requires the factor matrices to be nonnegative. The CX and CUR decompositions do not restrict the values in the factor matrices as such. Instead, the interpretability is achieved via the requirement of using the actual columns and, in CUR, rows of the original matrix. If the columns (and rows) of the data matrix had some interpretation, and they usually had, then so have the columns of $\mathbf{C}$ (and rows of $\mathbf{R}$). In a sense, the CX decomposition can be seen as a feature selection method.

The NCX and NCUR decompositions combine the approaches taken by NMF and CX decompositions by assuming nonnegative input and requiring the matrix $\mathbf{X}$ to be nonnegative. The matrix $\mathbf{C}$ is, of course, nonnegative if $\mathbf{A}$ is.

Finally, it should be noted that a variation of the CX decomposition has been used in bioinformatics to reduce the redundancy in a set of genetic markers [PDL$^+$08].

5.3 Properties of NCX and NCUR Decompositions

It is easy to see that the following basic properties hold for NCX and NCUR. An NCX decomposition of $\mathbf{A}$ with parameter k is at-most-rank-k approximation of $\mathbf{A}$, and $\mathsf{cost}^*_{\text{NCX}}(\mathbf{A}, k) \geq \mathsf{cost}^*_{\text{SVD}}(\mathbf{A}, k)$; the same holds for NCUR with k replaced by $\min\{k_c, k_r\}$. The error of NCX decomposition is also bounded below by a nonnegative decomposition of $\mathbf{A}$, i.e. $\mathsf{cost}^*_{\text{NCX}}(\mathbf{A}, k) \geq \mathsf{cost}^*_{\text{NMF}}(\mathbf{A}, k)$, and the same holds for NCUR, again by replacing k with $\min\{k_c, k_r\}$ (but notice that it is $\mathsf{cost}^*_{\text{NMF}}$; the lower-bound property does not have to hold for any specific NMF algorithm, unless it is optimal). Yet another lower bound is given by CX (and CUR) decompositions, and again in the form that the optimal CX (or CUR) decomposition costs at most the cost of optimal NCX (or NCUR) algorithm. The optimal cost of NCX decreases when the number of columns in $\mathbf{C}$ increases, but again, this does not need to hold for specific algorithms for NCX (although we would certainly like it to hold).

These properties are simple and straight forward. A more interesting question is the computational complexity of the problems. Unfortunately, virtually nothing is known in this respect. The research has hitherto concentrated on the CX problem (or on the CSS problem; these two are essentially the same from computational complexity's point of view), but not even an NP-hardness result for it is known [BMD09]. There is, however, some evidence towards the hardness of the problem, as a recent result shows that finding a submatrix $\mathbf{C}$ of k columns of $\mathbf{A}$ such that the *volume* of the parallelepiped spanned by the columns of $\mathbf{C}$ is maximized is NP-hard [ÇMI07].

One remarkable feature of the CX problem is that it does not have any algorithm that has provable approximation guarantee with respect to the optimal CX decomposition, but several algorithms have been proposed that have guarantees with respect to the SVD of the same matrix (e.g. [BMD09]).

Whether a hardness result for CX would imply anything about the hardness of NCX is another open issue. Indeed, there is no obvious relation between these two problems (other than the lower-bound property), and finding one could perhaps help in understanding both of the problems.

What we do know is the hardness of the problem where, given a matrix $\mathbf{A}$ and a matrix $\mathbf{C}$, a (possibly nonnegative) matrix $\mathbf{X}$ is sought, and the cost to minimize is $\|\mathbf{A} - \mathbf{CX}\|_F$. This problem, which is a clear analogy of the BU problem, is in P with and without the nonnegativity constraint. Without the nonnegativity constraint it can be solved optimally using the Moore–Penrose matrix pseudoinverse of $\mathbf{C}$, C^+ [GVL96, p. 257]: $\mathbf{X} = \mathbf{C}^+\mathbf{A}$. With the nonnegativity constraint, the problem is still solvable in polynomial time as a convex quadratic program (CQP) [KTH79], and also general convex optimization methods or quasi-Newton methods could be used. As a result, we can find optimal CX and NCX decompositions in time $O(m^k \operatorname{poly}(n, m))$ by enumerating all possible k-subsets of the columns of $\mathbf{A}$. Note, however, that this does not mean that the CX (NCX) problem is in FPT: for that, the function exponential in k must not depend on n or m.

5.4 Nonnegative Column Subset Selection Algorithms

The possible intractability of the NCX problem lies in the NCSS problem, and thus the algorithms given here concentrate on solving it. To that end, one could take any existing algorithm for the CSS problem and then apply the nonnegativity constraint to matrix $\mathbf{X}$ (for existing algorithms, see, e.g. [BPS05, DMM06b, BMD09]) Instead of taking this approach, two new algorithms are proposed: `LocNCX` and `ALS`.

The structure of the NCSS problem (and the CSS problem) lends itself readily to a local-search heuristic, which `LocNCX` is. In general, a local-search heuristic (or greedy local-search heuristic) works by studying the *neighbourhood* of the current solution and moving to the neighbouring solution improving the solution the most. The definition of the neighbourhood is problem- and algorithm-dependent, but it usually consists of all solutions achievable from the current

Algorithm 4 A generic local-search heuristic.

Input: An instance I of a minimization problem Π.
Output: Solution S of I.

```
1: function Local($I$)
2:     $S \leftarrow$ random solution of $I$
3:     repeat
4:         $\mathcal{N} \leftarrow$ neighbourhood of $S$
5:         $S_{\text{best}} \leftarrow S$
6:         $S \leftarrow \arg\min_{S \in \mathcal{N}} \mathsf{cost}_\Pi(I, S)$
7:     until $\mathsf{cost}_\Pi(I, S) \geq \mathsf{cost}_\Pi(I, S_{\text{best}})$
8:     return $S_{\text{best}}$
9: end function
```

one with only a small change; in `LocNCX` the neighbourhood of $\mathbf{C}$ consists of all matrices $\mathbf{C}'$ with exactly one column being changed to a column of $\mathbf{A}$ not in $\mathbf{C}$. The algorithm continues until it reaches a local optimum, that is, until no neighbouring solution is better than the current solution.

Local-search algorithms need some initial solution to start with. Usually, and particularly in this thesis, the initial solution is randomly selected, but in some cases careful combination of randomness and determinism can yield better results (cf. [AV07]).

Local-search algorithms can make exponentially many iterations in the worst case if they approach a local optimum with exponentially small steps, but in practice that seldom happens. Nevertheless, it is possible to avoid this by requiring that the algorithm stops after the best possible update is less than a certain threshold. Naturally, this reduces the quality of the answer, and the trade-off is controlled by a parameter that, together with the input size, determines the threshold (for a concrete example, see [AGK$^+$04]). Another option would be to use a constraint, preferably a user-defined one, for the maximum number of iterations allowed. A generic local-search algorithm for minimization problems is given as Algorithm 4.

The `LocNCX` algorithm is a local-search heuristic with the neighbourhood structure, as mentioned, being those matrices $\mathbf{C}'$ having exactly one column different from $\mathbf{C}$. Thus, there are at most $k(n-k)$ neighbours for each matrix $\mathbf{C}$ to consider. An important detail on `LocNCX` is the computation of the cost induced by a matrix $\mathbf{C}$ – computing it requires computing the corresponding matrix $\mathbf{X}$.

Algorithm 5 The ALS algorithm for NCX.

```
Input: An $n$-by-$m$ nonnegative matrix $\mathbf{A}$ and an integer $k$.
Output: Matrices $\mathbf{C}$ and $\mathbf{X}$ that approximate $\mathbf{A}$.
1: $J = k$ random column indices from $\{1, \dots, m\}$
2: $\tilde{\mathbf{C}} = \mathbf{A}^J$
3: while err decreases and max iterations are not reached do
4:     $(\mathbf{X}, \text{err}) = \texttt{SolveX}(\mathbf{A}, \tilde{\mathbf{C}})$
5:     $(\tilde{\mathbf{C}}, \text{err}) = \texttt{Solve}\tilde{\texttt{C}}(\mathbf{A}, \mathbf{X})$
6: end while
7: $\mathbf{C} = \texttt{MatchColumns}(\tilde{\mathbf{C}}, \mathbf{A})$
8: $(\mathbf{X}, \text{err}) = \texttt{SolveX}(\mathbf{A}, \mathbf{C})$
```

For subsequent references, the subroutine to compute $\mathbf{X}$ for a given $\mathbf{C}$ is called `SolveX`. For discussion about `SolveX`, see Section 5.4.1.

Notice also that the `LocNCX` algorithm is not bounded to the NCX problem, but it can also be used as an alternative solution to the standard CX problem simply by not requiring $\mathbf{X}$ to be nonnegative. As we will see in Section 5.6.4, using `LocNCX` for the standard CX problem can yield better results than the previously-proposed algorithms.

The second algorithm for the NCX problem comes with the name `ALS` and is inspired by the alternating least squares method used in the NMF algorithms [BBL$^+$07]. A sketch of the algorithm is given in Algorithm 5. The idea is the following: first pick a random set of k columns from the original matrix to construct a matrix $\tilde{\mathbf{C}}$. Then use this $\tilde{\mathbf{C}}$ to compute the nonnegative matrix $\mathbf{X}$ using the routine `SolveX`. In turn, compute a new nonnegative matrix $\tilde{\mathbf{C}}$ using the current $\mathbf{X}$ in an analogous way of computing $\mathbf{X}$ given $\tilde{\mathbf{C}}$. The new $\tilde{\mathbf{C}}$ does not have to contain columns of $\mathbf{A}$ (hence $\tilde{\mathbf{C}}$, not $\mathbf{C}$), and is computed by routine `Solve`$\tilde{\texttt{C}}$. Continue by recomputing $\mathbf{X}$ given the new $\tilde{\mathbf{C}}$, and so on.

The loop terminates either when the alternating process gives no more improvement in the error function, or when a certain number of iterations has been reached. Alternatively, stopping criteria can be changed so that the **while** loop terminates when the improvement in the error goes below some threshold. The user can supply the algorithm with stopping criteria best suited to her needs.

After the **while** loop, matrix $\tilde{\mathbf{C}}$ might not – and in general does not – contain columns of $\mathbf{A}$. Thus, an additional post-processing step is needed, and this is what happens on line 7 of Algorithm 5. The task is to find k distinct columns of $\mathbf{A}$ that are as close as possible to the columns of $\tilde{\mathbf{C}}$. This task is handled by the routine `MatchColumns` as follows: A weighted complete bipartite graph is constructed, the columns of $\mathbf{A}$ being the vertices in the left-hand side and the columns of $\tilde{\mathbf{C}}$ being the vertices in the right-hand side. The weight of the edge between columns $\mathbf{a}^i$ and $\tilde{\mathbf{c}}^j$ is the cosine similarity between them, that is, $\langle \mathbf{a}^i, \tilde{\mathbf{c}}^j \rangle / (\|\mathbf{a}^i\| \|\tilde{\mathbf{c}}^j\|)$, but also other similarity measures could be used. The columns of $\mathbf{C}$ are those matched with the columns of $\tilde{\mathbf{C}}$ in the minimum-weight matching of the bipartite graph. An optimal matching can be found in polynomial time using the Hungarian method [PS82, pp. 221–224]. Finally, the last row recomputes $\mathbf{X}$ for the final $\mathbf{C}$.

5.4.1 Solving $\mathbf{X}$ and $\tilde{\mathbf{C}}$

The methods `SolveX` and `Solve`$\tilde{\texttt{C}}$ require solving the constrained least-squares fitting problem: given $\mathbf{A}$ and $\mathbf{C}$, minimize $\|\mathbf{A} - \mathbf{C}\mathbf{X}\|_F^2$ such that $x_{ij} \geq 0$. As mentioned above, this is doable in polynomial time using various methods.

Unfortunately, these methods can be too slow to be used in a local-search algorithm such as `LocNCX`, and thus in this thesis a very simple approach is taken instead. First, an approximation of $\mathbf{X}$, called $\mathbf{X}'$, is computed using the pseudoinverse: $\mathbf{X}' = \mathbf{C}^+\mathbf{A}$. Second, the final matrix $\mathbf{X}$ is constructed from $\mathbf{X}'$ by projecting all negative elements of the latter to 0. That is,

$$x_{ij} = \begin{cases} x'_{ij} & \text{if } x'_{ij} \geq 0 \\ 0 & \text{if } x'_{ij} < 0. \end{cases}$$

This *projection-based* approach is in fact a common practice employed in many NMF algorithms [BBL+07], where similar constrained least-squares fitting problems are encountered.

The projection approach is very simple, and can be improved with some simple ideas. For example, one could project all values of $\mathbf{X}'$ that are smaller than some positive ε to 0. A careful selection of ε can indeed yield a smaller reconstruction error, but not without introducing a new parameter. Thus, aiming to simplicity, this idea is not used in the algorithms.

It should be noted that `LocNCX`, equipped with the aforementioned projection method, is not strictly a local-search heuristic as described in Algorithm 4, as the selected neighbour is not necessary optimal over all neighbours. The same is, of course, true for all similar algorithms using possibly suboptimal neighbour selection.

5.4.2 Convergence of the Algorithms

The `LocNCX` algorithm will eventually converge to some stationary point since there is only limited, albeit exponential, number of different matrices $\mathbf{C}$. Yet, this stationary point is not guaranteed to be a local optimum. The convergence of `ALS` is a more complicated issue. Would there not be the requirement for the columns to be from the original matrix, the results reported in [BBL$^+$07] would directly apply here. That is, the **while** loop in line 3 of the `ALS` algorithm converges to a local optimum provided that the optimal solution is computed in every step. But the last step of mapping the columns of $\tilde{\mathbf{C}}$ to the closest columns of the original matrix $\mathbf{A}$ (line 7) prevents any claims about the optimality of the obtained solution.

5.4.3 Time Complexity

The time complexity of the algorithms is dictated by the time complexity of `SolveX` and `Solve`$\tilde{\texttt{C}}$, both requiring the computation of a Moore–Penrose pseudoinverse for an n-by-k (or k-by-m) matrix, and a multiplication of two matrices of sizes n-by-k and k-by-m. Using SVD for the pseudoinverse, this is doable in time $O(nk^2)$ (assuming $n = m$ and $k < n$) [GVL96, p. 254]. That is also the time needed for a single iteration of the `ALS` algorithm. A single iteration of `LocNCX` takes time $O((n-k)nk^3)$. Assuming that the number of iterations is constant, these are the resulting time complexities, albeit with possibly large hidden constants.

5.5 Algorithms for NCUR Decompositions

The simplest approach to find CUR (and NCUR) decompositions is to first find the matrix $\mathbf{C}$ using any algorithm for the CX (resp.

NCX) decomposition, and then find the matrix $\mathbf{R}$ by applying the same algorithm on the transpose of the input matrix. Finally, the mixing matrix $\mathbf{U}$ needs to be solved. For CUR this is done using the pseudoinverse as $\mathbf{U} = \mathbf{C}^+\mathbf{A}\mathbf{R}^+$; for NCUR the projection approach can be used.

The transpose-based approach is actually used for solving CUR (e.g. [BPS05]), but an alternative approach, explained here, is to apply the local-search heuristic to the CUR (or NCUR) decomposition. The algorithm resembles `LocNCX`, the difference being in the neighbourhood structure. In this algorithm, `LocNCUR`, the neighbourhood of $(\mathbf{C}, \mathbf{R})$ is the set of all pairs $(\mathbf{C}', \mathbf{R}')$ where either $\mathbf{C}' = \mathbf{C}$ or $\mathbf{R}' = \mathbf{R}$ and in the other matrix exactly one column (in $\mathbf{C}'$) or row (in $\mathbf{R}'$) is different to $\mathbf{C}$ or $\mathbf{R}$. Again, the projection method is used to compute $\mathbf{U}$ and thus the cost of pair $(\mathbf{C}, \mathbf{R})$.

5.6 Experimental Evaluation

The purposes of the experiments in this section are the same as in the other experimental evaluations in this thesis, namely, to study the effects of different data characteristics; to compare the proposed methods to other ones; to validate the numerical results obtained with synthetic data with real-world data; and to study the interpretability of the results obtained from real-world data. But unlike the other chapters, this chapter deals with nonnegative, real-valued matrices, not binary ones, and this makes the experiments slightly different. Additionally, a small experiment is done in order to study the usability of `LocNCX` as an algorithm for the CX decomposition.

5.6.1 Algorithms Used

In total six different NCX and CX algorithms (and two variations) were used in these experiments, plus eight additional NCUR and CUR algorithms. The NCX (and NCUR) decompositions have not been studied previously, and thus the proposed algorithms cannot be compared against other NCX (and NCUR) algorithms. Instead, two algorithms solving the standard CX decomposition are used, namely `2Step` and `SPQR`. The former, by Boutsidis, Mahoney, and Drineas [BMD08, BMD09], is a new algorithm for CX decomposition that, with a high probability, approximates the best *rank-k Singular Value Decomposition* to within a factor of $O(k\sqrt{\log k})$ with respect to the Frobenius norm. Notice that this is a very good result, as a CX decomposition of k columns is a very restricted type of rank-k decomposition, whereas SVD provides the optimal rank-k decomposition. The latter algorithm, `SPQR`, is due to Stewart et al. [Ste99, Ste05, BPS05]. Unlike `2Step`, it does not have any guaranteed approximation properties.

These algorithms take very different approaches to the problem: `SPQR` selects columns in $\mathbf{C}$ in a greedy way, such that with the same data its results for k and $k-1$ differ only in the last column of $\mathbf{C}$. The `2Step` algorithm, on the other hand, works in two steps: first, it samples some number $c > k$ columns from $\mathbf{A}$ based on carefully selected probabilities. In the second step, it employs some deterministic algorithm to select the final k columns from the c sampled ones. Thus, its results with k and $k-1$ columns (and, indeed, between two runs with same value of k) can be completely different.

A variation of both `2Step` and `SPQR` were also used. They are referred to as $\texttt{2Step}_{\geq 0}$ and $\texttt{SPQR}_{\geq 0}$, and they differ from the original versions in that their decompositions are forced to be nonnegative. Specifically, the matrices $\mathbf{X}$ returned by `2Step` and `SPQR` were projected to the nonnegative orthant (i.e. all coordinates were projected to nonnegative values) in $\texttt{2Step}_{\geq 0}$ and $\texttt{SPQR}_{\geq 0}$.

The other two algorithms against which the proposed algorithms were compared were `NMF` and `SVD`. Notice that both present, at least in theory, a lower bound for NCX (and NCUR) decompositions.

The details for `NMF` and `SVD` are same as in Section 4.8.1. For `2Step` and `SPQR`, the implementations were provided by the respective authors. The deterministic subroutine in `2Step` was `SPQR`, it selected $c = \Theta(k \log k)$ columns in the first step, and the algorithm was re-started 40 times, as suggested in [BMD09].

When studying the applicability of `LocNCX` to solve the CX problem, the algorithm is denoted by `LocCX`; it differs from `LocNCX` only by not projecting the computed $\mathbf{X}$ to the nonnegative orthant.

The `LocNCX`, `LocCX`, and `ALS` algorithms were re-started 13 times to overcome the adverse effects of a bad initial selection of $\mathbf{C}$.

5.6.2 Synthetic NCX Data

The purposes of studying the synthetic NCX data were same as the purposes of on synthetic BMF data, that is, to study the effects of (*1*) the number of columns in $\mathbf{C}$; (*2*) the noise level; (*3*) the density of the columns in $\mathbf{C}$; and (*4*) the mean number of columns of $\mathbf{C}$ used to create each of the columns of the data.

Data generation process. The data was generated by varying each of the four parameters one-by-one. Twenty matrices were created for each combination of parameters. The default values for parameters used when the parameter was not varied were $k = 16$, 10% of noise, density of 0.1 and 4 columns of $\mathbf{C}$ per each column of the data.

The data generation process was different from that of previous chapters. First, a nonnegative random matrix $\mathbf{C}$ was generated with each element being sampled uniformly at random from $]0, 1[$. To model the sparsity of real data, $d\%$ of the elements were then set to 0, selecting the values at random.

A random matrix $\mathbf{X}$ was created again by sampling the elements from $]0, 1[$. Random elements in $\mathbf{X}$ were set to 0 in a way that, on expectation, there were a prescribed number of non-zero entries in each column of $\mathbf{X}$. A matrix $\tilde{\mathbf{A}}$ was created as $\tilde{\mathbf{A}} = \mathbf{CY}$ with $\mathbf{Y} = (\mathbf{I}_k\ \mathbf{X})$, so that the columns of $\mathbf{C}$ are the first k columns of $\tilde{\mathbf{A}}$.

Noise was added by selecting, uniformly at random, a required percentage of elements of $\tilde{\mathbf{A}}$ and perturbing these elements' values with Gaussian noise with mean 0 and standard deviation 0.5. If this yielded negative values in the resulting matrix, they were projected to 0 to obtain the final matrix $\mathbf{A}$.

Number of columns in C. The number of columns in $\mathbf{C}$, k, varied from 8 to 28 with steps of size 2. The total density of the resulting matrices varied between 0.34 and 0.4.

The results of these experiments are presented in Figure 5.2. All NCX (and CX) methods are very close to each other at all points. They also follow the trend of `NMF` and `SVD` closely, although the latter two are clearly the best methods. Forcing the results of `2Step` and `SPQR` to the nonnegative orthant does not seem to have strong effects on their results' quality, hinting that they already find such columns in $\mathbf{C}$ that are best used as nonnegative combinations to present the remaining columns.

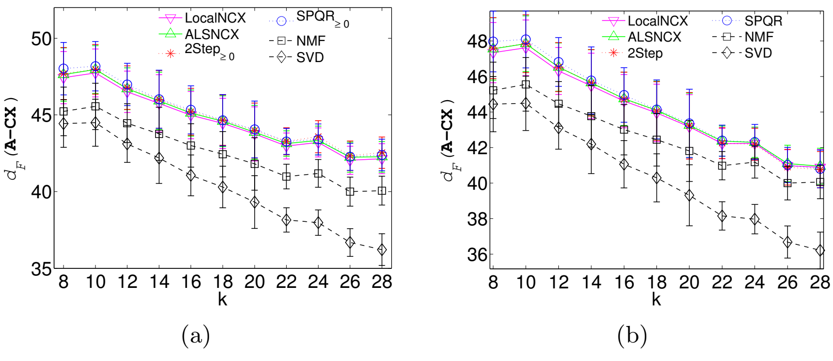


Figure 5.2: Reconstruction errors of NCX decomposition when k, the number of columns in $\mathbf{C}$, varies. The matrix $\mathbf{X}$ returned by `2Step` and `SPQR` is projected to nonnegative orthant in (a) and left as it in (b). Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.

On the other hand, the good performance of `ALS` and `LocNCX` show that the ability of using negative real values does not add any extra power to the other methods.

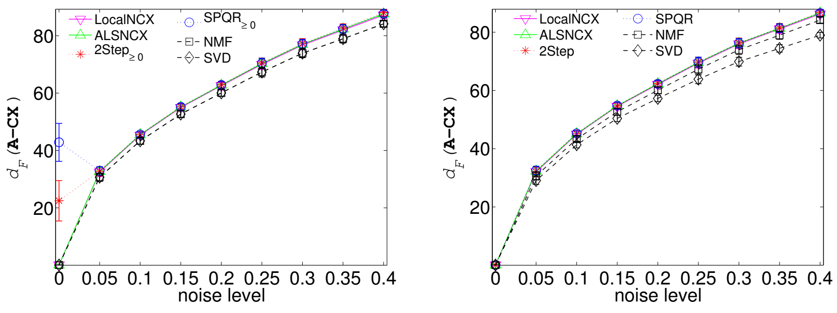
Noise level. The level of noise varied from 0 to 0.4 with steps of size 0.05. The resulting matrices' density varied between 0.35 and 0.45. The results are presented in Figure 5.3.

All algorithms' error increase with increasing noise, as is expected, and the difference between the best (`SVD`) and the worst stays constant and is very small. Nevertheless, `2Step`$_{\geq 0}$ and `SPQR`$_{\geq 0}$ show strange behaviour with data having no noise.

Density of columns of C. The mean density (number of nonzero elements divided by the total number of elements) of columns of $\mathbf{C}$ varied from 0.05 to 0.3. The total density of the resulting matrices varied from approximately 0.2 to approximately 0.75. The results can be seen in Figure 5.4.

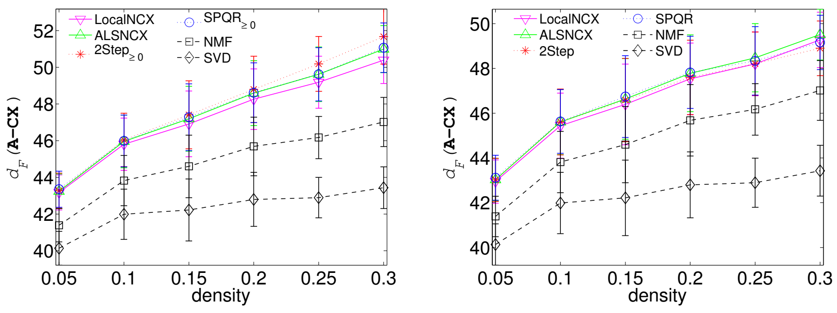
The increasing density seems to increase the error with all algorithms. Otherwise the results follow the previous ones: `SVD` is the best, `NMF` the second, and all other algorithms are basically indistinguishable above it. The small variations between the ordering of these algorithms are all well within the standard deviation, and thus are not very significant.

Mean number of columns of C used for each column of A. The mean number of columns of $\mathbf{C}$ involved in the nonnegative



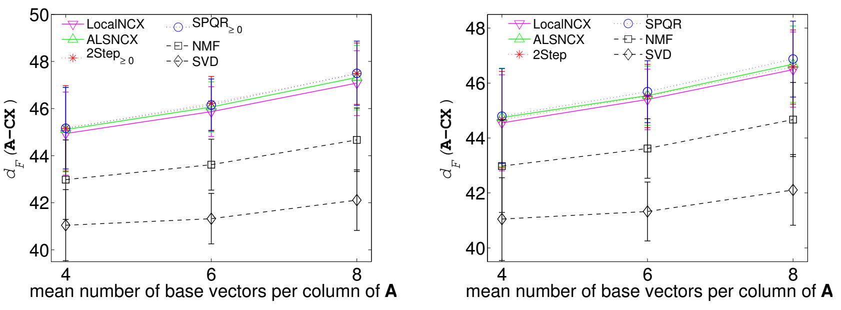
(a) (b)

Figure 5.3: Reconstruction errors of NCX decomposition when the level of noise varies. The matrix $\mathbf{X}$ returned by `2Step` and `SPQR` is projected to the nonnegative orthant in (a) and left as it is in (b). Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.



(a) (b)

Figure 5.4: Reconstruction errors of NCX decomposition when the density of $\mathbf{C}$'s columns varies. The matrix $\mathbf{X}$ returned by `2Step` and `SPQR` is projected to the nonnegative orthant in (a) and left as it is in (b). Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.



(a) (b)

Figure 5.5: Reconstruction errors of NCX decomposition when the mean number of **C**'s columns used for each column of **A** varies. The matrix **X** returned by `2Step` and `SPQR` is projected to the nonnegative orthant in (a) and left as it is in (b). Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.

linear combination used to create each column of **A** (i.e. the number of positive coefficients) had three possible values, 4, 6, and 8. The resulting matrices had an approximate total density between 0.4 and 0.55. The results, from Figure 5.5, do not bring anything new to the discussion: they follow trends similar to the previous ones.

5.6.3 Synthetic NCUR Data

The purposes of studying the synthetic NCUR data were similar to those of synthetic NCX data (and hence, to those of synthetic BMF data): they were used to study the effects of (*1*) the number of columns in **C** and the number of rows in **R**; (*2*) the noise level; and (*3*) the density of the columns of **C** and rows of **R**. The fourth characteristic in the synthetic NCX and BMF data, the mean number of columns of **C** used to create each of the columns of the data, does not have a counterpart in the NCUR data generation process.

Data generation process. The data was generated by varying each of the three parameters one-by-one. Twenty matrices were created for each combination of parameters, using the following default values for parameters when they were not varied: = 16, 10% of noise, and density of 0.1. To ensure that the generated matrices, before the introduction of noise, do have an exact NCUR decomposition, only a special type of decomposition was used. Parameters n and m were set to 150 and 80, respectively. The number of columns in $\mathbf{C}$ was equal to the number of rows in matrix $\mathbf{R}$, both denoted by k, and the structure of the matrices was

$$\mathbf{C} = \begin{pmatrix} \mathbf{I}_k \\ \mathbf{X}_{n\times k} \end{pmatrix} \quad \text{and} \quad \mathbf{R} = \begin{pmatrix} \mathbf{I}_k & \mathbf{X'}_{k\times m} \end{pmatrix},$$

where $\mathbf{X}_{n\times k}$ and $\mathbf{X'}_{k\times m}$ are n-by-k and k-by-m random matrices with independently and identically distributed entries from $]0,1[$. To model the sparsity, predetermined number of the entries (selected uniformly at random) were set to 0. In the data generation process, the mixing matrix $\mathbf{U}$ was a k-by-k identity matrix $\mathbf{I}_k$, and the generated matrix was $\mathbf{A} = \mathbf{CR}$. The first k rows of $\mathbf{A}$ were those of $\mathbf{R}$, and the first k columns of $\mathbf{A}$ were those of $\mathbf{C}$. Noise was introduced by perturbing a required percentage of the elements of $\mathbf{A}$ with Gaussian noise with mean 0 and standard deviation 0.5; if the perturbation resulted in negative-valued elements, they were projected to 0.

Number of columns in C and rows in R. The first parameter, k, varied from 8 to 28 with steps of size 2. The total density of the resulting matrices varied between 0.06 and 0.2.

The results of this experiment are presented in Figure 5.6. `SVD` and `NMF` are the best. Over all methods returning an NCUR decomposition (Figure 5.6(a)), `LocNCUR` is the best. Also $\texttt{ALS}^T$ and $\texttt{LocNCX}^T$ perform better than $\texttt{2Step}^T_{\geq 0}$ or $\texttt{SPQR}^T_{\geq 0}$.

When the nonnegativity constraint is removed from $\texttt{2Step}^T$ and $\texttt{SPQR}^T$, they start to perform better, being slightly better than `LocNCUR` with higher values of k (Figure 5.6(b)). In general, the results of all CUR and NCUR algorithms are comparable to each other, all following the same trends. None of the algorithms is able to attain a reconstruction error close to that of NMF or SVD, hinting an inherent complexity in finding a good NCUR (or CUR) decomposition.

Noise level. The level of noise varied from 0 to 0.4 with steps of size 0.05. The resulting matrices' density varied between 0.1 and 0.16. The results, presented in Figure 5.7 are similar to those in Figure 5.6: `SVD` and `NMF` are the best and, with $\texttt{2Step}^T$ and $\texttt{SPQR}^T$ forced to nonnegative $\mathbf{U}$, `LocNCUR`, $\texttt{ALS}^T$, and $\texttt{LocNCX}^T$ are slightly

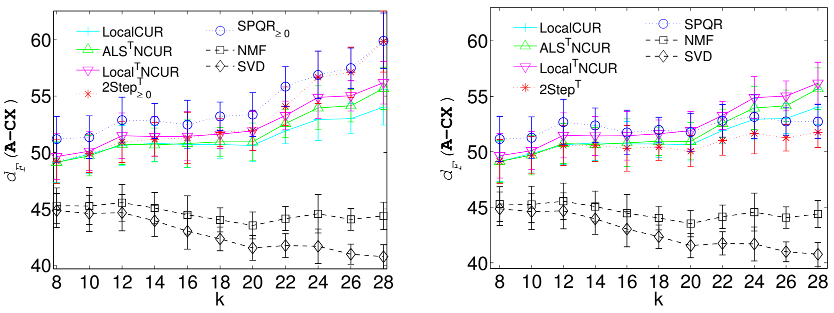


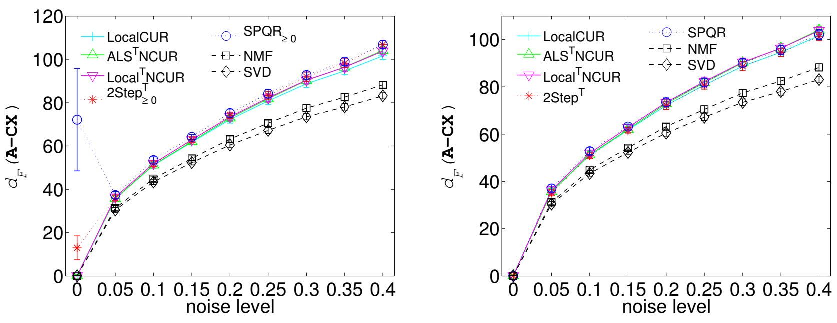
Figure 5.6: Reconstruction errors of NCUR decomposition when the number of columns in $\mathbf{C}$ and the number of rows in $\mathbf{R}$, k, varies. The matrix $\mathbf{U}$ returned by 2Step^T and SPQR^T is projected to the nonnegative orthant in (a), and left as it is in (b). Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.

better. With 2Step^T and SPQR allowed to return arbitrary-valued $\mathbf{U}$, the differences between the methods is indistinguishable.

Density of columns in C and rows in R. The mean density of columns of $\mathbf{C}$ and rows of $\mathbf{R}$ varied from 0.05 to 0.3, yielding matrices with total density between 0.03 and 0.6. This experiment presents the biggest difference between the methods designed for NCUR decompositions (namely, LocNCUR, LocNCX^T, and ALS^T) and methods designed for general CUR decomposition (2Step^T and SPQR^T). When the results of 2Step^T and SPQR^T are forced to nonnegative values, their results are notably inferior to those of LocNCUR, LocNCX^T, and ALS^T with higher levels of density, as can be seen from Figure 5.8(a). On the other hand, when they are allowed to return an arbitrary-valued matrix $\mathbf{U}$, only LocNCUR can present comparable results (Figure 5.8(b)).

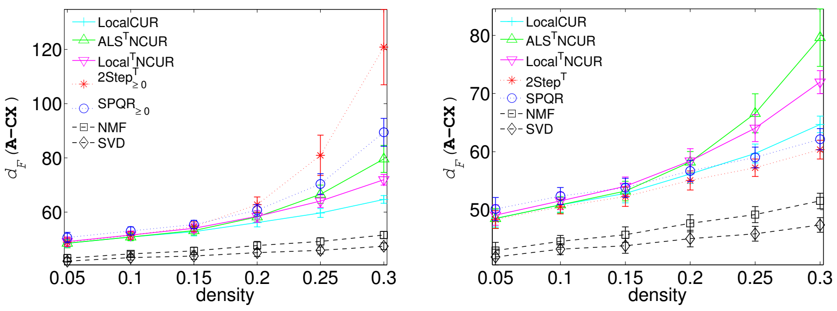
5.6.4 Solving the CX Decomposition

A small experiment was conducted in order to verify whether the local search is applicable also to the standard CX problem, and how well LocNCX would do without the nonnegativity constraint. To this end, two set of matrices admitting CX decomposition were created in a process described below.



(a) (b)

Figure 5.7: Reconstruction errors of NCUR decomposition when the level of noise varies. The matrix $\mathbf{U}$ returned by $\texttt{2Step}^T$ and $\texttt{SPQR}^T$ is projected to the nonnegative orthant in (a), and left as it is in (b). Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.



(a) (b)

Figure 5.8: Reconstruction errors of NCUR decomposition when the density parameter varies. The values of x-axes refer to the parameter used in generating the data, not to the actual density of the data. The matrix $\mathbf{U}$ returned by $\texttt{2Step}^T$ and $\texttt{SPQR}^T$ is projected to the nonnegative orthant in (a), and left as it is in (b). Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation.

Data generation process. All matrices had 150 rows and 92 columns, 80 of which are random linear combinations of the remaining 12.

First set of matrices is based on a uniform distribution. First, a random 100-by-12 matrix $\mathbf{C}$ was created by sampling its values uniformly at random from $]0,1[$. Then, a random 12-by-92 matrix $\mathbf{X}$ was created by setting the leftmost 12-by-12 submatrix as an identity matrix, and sampling the values for the rest uniformly at random from $]0,1[$. These two matrices were multiplied together to obtain a matrix $\tilde{\mathbf{A}}$.

The second set of matrices was constructed as the first one, but when in the first one the sampling was done from the uniform distribution, in the second one it was done from the normal distribution with mean 0 and variance 1.

Noise was added to the obtained matrices as follows. A predefined portion of matrix elements were selected uniformly at random. These elements were perturbed using Gaussian noise with 0 mean and standard deviation 0.5. The portion of elements perturbed varied from 0 to 0.3 with steps of 0.05. Forty matrices were created in both sets for each level of noise, resulting in a total of 560 matrices.

Experiments and results. The `LocCX` algorithm (i.e. `LocNCX` without the nonnegativity constraint) was compared against `2Step` and against a result by a third algorithm that had the matrix $\mathbf{C}$ used in creating the data as an input and solved $\mathbf{X}$ optimally. Notice, however, that this algorithm is not necessarily optimal: with the introduction of the noise, it is not clear that the columns from the original matrix $\mathbf{C}$ still constitute the optimal factor matrix for the data.

Figure 5.9 presents the results. In the figure, the dotted lines represent maxima and minima of reconstruction error and the solid line represents the mean. With data generated using uniform distribution (Figure 5.9(a)), all methods perform similarly. Upon close examination, one can see that with higher levels of noise `LocCX` and `2Step` perform better than the method of selecting the original $\mathbf{C}$ (i.e. there are better columns to be selected than those originally used), and that between `LocCX` and `2Step` the former is constantly, albeit narrowly, better than the latter.

The difference between `LocCX` and `2Step` is more visible in Figure 5.9(b), where the data was created using normal distribution:

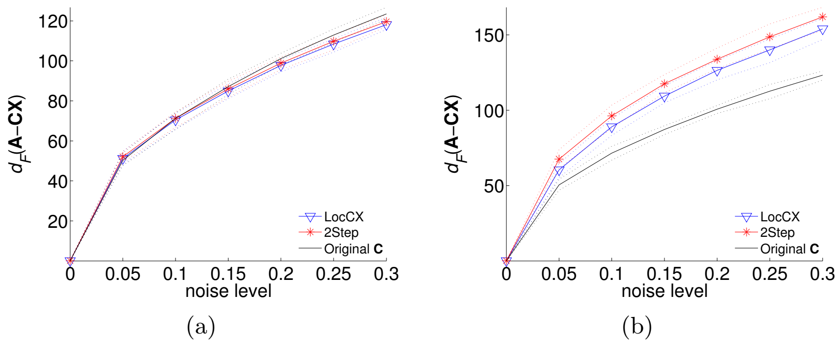


Figure 5.9: Reconstruction errors of CX decomposition. Dotted lines represent maxima and minima of reconstruction error and solid line the mean over 40 random matrices created with identical parameters. (a) Matrices created using uniform distribution. (b) Matrices created using normal distribution.

the worst result of `LocCX` is always better than the average result of `2Step`. In contrast to the uniform distribution, selecting the original matrix **C** proves to be very advantageous, being clearly superior to `LocCX` and `2Step`. This shows that neither of the algorithms is able to find optimal CX decomposition. Yet, the difference between `LocCX` and the method using the original **C** stays approximately constant after the noise level has reached 0.1, the reconstruction error of the former being roughly 6/5 of the reconstruction error of the latter. Thus, this experiment does not rule out the possibility that `LocCX` could have a provable, constant approximation guarantee.

5.6.5 Real-World Data

The purposes of experiments done with the real-world data were twofold. On one hand, to verify that the good results obtained with synthetic data carry over to real-world data sets, and on the other hand, to make sure that the interpretability of the results, a main motivation behind NCX and NCUR decompositions, is indeed good.

Data sets used. The data sets used were similar to those in Section 4.8.4. Nevertheless, there are a few differences. Abstracts data was not used in these experiments; DBLP data contained also information about how many times an author has published in a conference, not just the published/not published information used with BMF; Dialect and NOW data sets were exactly the same. The 20Newsgroups data set was replaced with a smaller subset, called 4News, having 87 news from 4 Usenet groups, namely, `sci.crypt`, `sci.med`, `sci.space`, and `soc.religion.christian`. The data was preprocessed[1] using Andrew McCallum's Bow Toolkit[2] with stemming and stop word removal, and taking the 100 most frequent words. Thus the data matrix is 100-by-348, and it contains the frequencies of the terms per document.

Reconstruction errors. The results for NCX and CX decompositions with real-world data are given in Table 5.1. Perhaps the most important result is the good performance of `LocNCX`: it consistently presents results that are equivalent, or even better, than the results of `2Step` or `SPQR`. That is, `LocNCX` is able to find a *nonnegative* CX (NCX) decomposition that is better than the best CX decomposition found by the other algorithms. This is important for two reasons. On one hand, it shows that `LocNCX` can find good NCX decompositions, and on the other hand, it shows the concept of NCX decompositions being meaningful. `LocNCX` and `ALS` are almost on par, with the notable exception of DBLP data with $k = 15$ where `ALS` is considerably worse.

In Table 5.1, `2Step` and `SPQR` were allowed to use negative values in matrix $\mathbf{X}$. It is not surprising that when their results are forced to be nonnegative, via the same projection method used in `LocNCX` and `ALS`, the error is increased. Results for such experiments are reported in Table 5.2. Comparing to Table 5.1, we see that the change in error can be negligible (as with 4News and $k = 5$) or tremendous (DBLP and $k = 15$). Yet, `LocNCX` is always better than $\texttt{2Step}_{\geq 0}$ and $\texttt{SPQR}_{\geq 0}$, giving it best overall performance. The big change in $\texttt{2Step}_{\geq 0}$'s results, compared to `2Step`'s results, with Dialect data hints that the CX and NCX decompositions of the data are very different; on the other hand, the CX and NCX decompositions of 4News data are probably very similar, given the good performance of $\texttt{2Step}_{\geq 0}$ and $\texttt{SPQR}_{\geq 0}$.

Similar experiments were conducted with NCUR and CUR decompositions, but the DBLP data was left out due to the shape

[1]The author is grateful to Ata Kabán for preprocessing the data, and to Ella Bingham for providing the preprocessed data.

[2]`http://www.cs.cmu.edu/~mccallum/bow/`

Table 5.1: Reconstruction errors of NCX and CX decompositions with real-world data.

Data	k	`LocNCX`	`ALS`	`2Step`	`SPQR`	`NMF`	`SVD`
4News	5	134.96	135.62	134.87	134.87	128.16	127.64
	10	104.89	105.39	103.48	103.93	99.45	98.19
	15	93.37	92.91	87.70	89.16	85.97	83.00
DBLP	5	258.22	259.14	258.68	291.59	253.75	251.92
	10	165.92	173.00	168.44	170.58	158.97	157.30
	15	72.99	131.95	76.22	95.33	71.16	70.65
Dialect	5	201.40	201.46	205.67	228.46	179.93	177.72
	10	175.53	175.91	181.32	202.03	157.60	152.09
	15	159.88	160.88	165.46	184.30	146.16	137.53
NOW	5	35.07	35.08	36.11	36.12	32.23	31.93
	10	32.39	33.10	32.32	32.60	29.65	28.46
	15	30.50	31.19	30.03	29.90	27.91	25.84

Table 5.2: Reconstruction errors of NCX decompositions with real-world data. Results of `2Step`$_{\geq 0}$ and `SPQR`$_{\geq 0}$ are forced to be non-negative.

Data	k	`LocNCX`	`ALS`	`2Step`$_{\geq 0}$	`SPQR`$_{\geq 0}$	`NMF`	`SVD`
4News	5	134.96	135.62	134.96	134.96	128.16	127.64
	10	104.89	105.39	105.29	106.86	99.45	98.19
	15	93.37	92.91	96.43	96.24	85.97	83.00
DBLP	5	258.22	259.14	314.65	293.98	253.75	251.92
	10	165.92	173.00	222.98	240.33	158.97	157.30
	15	72.99	131.95	316.64	427.55	71.16	70.65
Dialect	5	201.68	201.73	205.84	228.54	179.93	177.72
	10	176.43	176.95	182.45	203.08	157.60	152.09
	15	162.83	163.71	168.22	186.50	146.16	137.53
NOW	5	35.07	35.08	36.29	36.58	32.23	31.93
	10	32.39	33.10	32.89	34.23	29.65	28.46
	15	30.50	31.19	31.97	33.46	27.91	25.84

of the matrix (having only 19 rows). The results are reported in Table 5.3. Over the proposed NCUR algorithms, LocNCUR gives the best overall performance, being next to $\mathtt{LocNCX}^T$ only once, and always better than $\mathtt{ALS}^T$. $\mathtt{LocNCX}^T$ is better than $\mathtt{ALS}^T$, although the differences are almost negligible. The CUR decompositions reported by $\mathtt{2Step}^T$ are better than those of $\mathtt{SPQR}^T$, and also better than NCUR decompositions by LocNCUR in 4News data. In Dialect and NOW, excluding cases with $k = 15$, LocNCUR's NCUR decomposition yield smaller error than $\mathtt{2Step}^T$'s CUR decomposition, hinting that the NCUR decompositions seems natural to these data sets.

When only NCUR decompositions were asked, the results (in Table 5.4) followed those with NCX decompositions (in Table 5.2): with 4News, $\mathtt{2Step}^T_{\geq 0}$ and $\mathtt{SPQR}^T_{\geq 0}$ perform almost as well as $\mathtt{2Step}^T$ and $\mathtt{SPQR}^T$, but with Dialect, the error of the former methods increases heavily. Overall, the best method for NCUR decompositions seems to be LocNCUR. This is not surprising, as it is the only method that is designed specifically for this problem.

Interpretability of the results. The NCX decomposition was used with the DBLP dataset (recall that it has only 19 rows). The results, with $k = 6$, give an example of intuitive column selection. The columns correspond to authors, and a priori, one would hope to have authors that represent the various fields of Computer Science covered by the selected conferences. As can be seen from Table 5.5, this is indeed the case with both LocNCX and ALS, although their results are different.

In ALS's results Umesh V. Vazirani is selected to represent theoretical computer scientists that publish in FOCS and STOC. In LocNCX's results, Vazirani's role is given to Leonid A. Levin. Equivalently, John Shawe-Taylor (in LocNCX's results) represents a group of machine learners, and Hector Garcia-Molina (in ALS) and Divesh Srivastava (in LocNCX) a group of data management researchers.

Another example, this time for NCUR decomposition, is given by the Dialect data. The experiment is somewhat similar to that done with BMF algorithms, except that in case of NCUR decompositions, the results contain actual municipalities (all sharing some feature) and features (all sharing some municipality). The first experiments are conducted with $\mathtt{ALS}^T$. In Figure 5.10(a), $\mathbf{C}$ has 7 columns and $\mathbf{R}$ has 7 rows; in Figure 5.10(b) they have 8 columns and rows, respectively. In both cases, the spread of selected features (rows

Table 5.3: Reconstruction errors of NCUR and CUR decompositions with real-world data.

Data	k	`LocNCUR`	`LocNCX`^{T}	`ALS`^{T}	`2Step`^{T}	`SPQR`^{T}	`NMF`	`SVD`
4News	5	142.48	140.68	146.83	140.11	151.30	128.16	127.64
	10	117.56	118.69	119.42	113.56	116.76	99.45	98.19
	15	109.63	115.67	116.99	93.96	99.06	85.97	83.00
Dialect	5	221.77	221.84	225.29	225.52	247.09	179.93	177.72
	10	199.29	197.64	207.05	201.63	226.12	157.60	152.09
	15	191.94	189.58	201.17	186.48	209.24	146.16	137.53
NOW	5	37.43	37.43	37.95	38.28	38.97	32.23	31.93
	10	35.18	35.22	36.39	35.85	36.59	29.65	28.46
	15	33.95	34.41	35.32	33.66	34.37	27.91	25.84

Table 5.4: Reconstruction errors of NCUR decompositions with real-world data. The results of $\texttt{2Step}^{T}_{\geq 0}$ and $\texttt{SPQR}^{T}_{\geq 0}$ are forced to be nonnegative.

Data	k	`LocNCUR`	$\texttt{LocNCX}^{T}$	$\texttt{ALS}^{T}$	$\texttt{2Step}^{T}_{\geq 0}$	$\texttt{SPQR}^{T}_{\geq 0}$	`NMF`	`SVD`
4News	5	142.48	140.68	146.83	140.68	152.14	128.16	127.64
	10	117.56	118.69	119.42	126.30	131.10	99.45	98.19
	15	109.63	115.67	116.99	159.65	156.70	85.97	83.00
Dialect	5	221.77	221.84	225.29	294.54	278.07	179.93	177.72
	10	199.29	197.64	207.05	388.28	401.95	157.60	152.09
	15	191.94	189.58	201.17	543.55	568.72	146.16	137.53
NOW	5	37.43	37.43	37.95	38.28	39.27	32.23	31.93
	10	35.18	35.22	36.39	37.26	39.79	29.65	28.46
	15	33.95	34.41	35.32	41.88	46.75	27.91	25.84

Table 5.5: Results for DBLP data using `ALS` and `LocNCX` with $k = 6$.

`ALS`	`LocNCX`
Naoki Abe	Divesh Srivastava
Craig Boutilier	Leonid A. Levin
Uli Wagner	Souhila Kaci
Umesh V. Vazirani	Xintao Wu
Hector Garcia-Molina	Uli Wagner
Dirk Van Gucht	John Shawe-Taylor

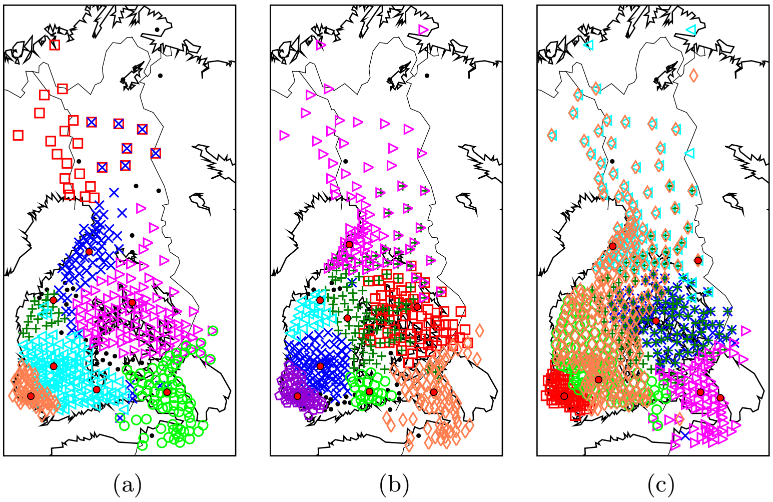


Figure 5.10: Spread of selected features (rows of **R**) and the selected municipalities (columns of **C**) as reported by (a) `ALS`^{T} with $k = 7$, (b) `ALS`^{T} with $k = 8$, and (c) `LocNCUR` with $k = 7$.

of **R**) is represented by open-faced (non-solid) symbols, while the selected municipalities (columns of **C**) are plotted as solid dots.

With $k = 7$, `ALS`^{T}'s results are a good approximation of the ground truth (Figure 4.5 on page 75): as there are only 7 features, central south-western dialects have been merged to neighbouring dialects. There is also some overlapping, as few Ostrobothnian municipalities have both Tavastian and South Ostrobothnian dialects, and some northern municipalities have dialects of far north and North Ostrobothnia. Given that the overlaps happen in the borders of neighbouring dialects, the phenomenon can be considered natural.

When the number of features and municipalities is increased to 8 (Figure 5.10(b)), some changes occur. As `ALS` adjusts its selection of columns (and rows, in `ALS`^{T}) based on the parameter k, there are differences in more than one dialect.

For both values of k, the municipalities selected (columns of $\mathbf{C}$) are mostly close to the geographical midpoint of their dialect regions. This is intuitive, as the 'purest' dialect can be assumed to be found in the middle of its spread.

If the results of `ALS`^{T} contained many non-overlapping regions, the same cannot be said about the results of `LocNCUR` (Figure 5.10(c)), where the overlapping is more a rule than an exception: only south-western and south-eastern dialects are recognizable.

5.6.6 Conclusion of Experiments

Overall, the proposed algorithms performed very well when compared against other algorithms. `LocNCX`, in particular, succeeded in finding NCX decompositions that were even better than the best CX decompositions found. When applied to finding CX decompositions, `LocNCX` outperformed `2Step` constantly, if also narrowly. `ALS` followed `LocNCX` with slightly higher reconstruction errors.

With CUR and NCUR decompositions, the results were similar. That `LocNCX`^{T} performed so well against `LocNCUR` in terms of reconstruction errors is somewhat surprising. The problematic interpretation of the results reported by `LocNCUR` seem to indicate that transposing an NCX algorithm, no matter how naïve it sounds, is the preferred approach for NCUR algorithms.

CHAPTER 6

The Boolean Column and Column-Row Decomposition Problems

Where the problems are motived as a combination of BMF *and* NCX/NCUR*, and new algorithms are proposed and compared against algorithms for* BMF *and for* CX *and* CUR*.*

6.1 Problem Definitions

In this chapter we return to the Boolean world, but without forgetting the ideas from the previous chapter. Indeed, this chapter presents a combination of ideas from the previous chapters: Boolean decomposition of Chapter 4 is combined with CX and CUR decompositions of Chapter 5 resulting to the Boolean column and Boolean column-row decomposition problems (BCX and BCUR for short). These are the two main problems studied in this chapter. They are defined in a natural way.

Boolean Column Decomposition Problem (BCX)**.** Given a binary matrix $\mathbf{A} \in \{0,1\}^{n\times m}$ and a positive integer k, find k columns of $\mathbf{A}$, constituting the matrix $\mathbf{C}$, and a binary matrix $\mathbf{X} \in \{0,1\}^{k\times m}$ such that $\mathbf{C}$ and $\mathbf{X}$ minimize

$$\mathsf{cost}_{\mathrm{BCX}}(\mathbf{A},\mathbf{C},\mathbf{X}) = d_1(\mathbf{A} - \mathbf{C} \circ \mathbf{X}). \tag{6.1}$$

Boolean Column-Row Decomposition Problem (BCUR)**.** Given a binary matrix $\mathbf{A} \in \{0,1\}^{n \times m}$ and positive integers k_c and k_r, find k_c columns of $\mathbf{A}$, constituting the matrix $\mathbf{C}$, k_r rows of $\mathbf{A}$, constituting the matrix $\mathbf{R}$, and a binary matrix $\mathbf{U} \in \{0,1\}^{k_c \times k_r}$ such that $\mathbf{C}$, $\mathbf{U}$, and $\mathbf{R}$ minimize

$$\mathsf{cost}_{\text{BCUR}}(\mathbf{A}, \mathbf{C}, \mathbf{U}, \mathbf{R}) = d_1(\mathbf{A} - \mathbf{C} \circ \mathbf{U} \circ \mathbf{R}). \tag{6.2}$$

6.2 Applications to Data Mining

The ideas behind the BCX and BCUR decompositions were taken from the previous chapters, and, indeed, also their applications are similar. Naturally, these decompositions are suitable only for binary-valued data, and the usage of Boolean matrix decomposition makes the inherent assumption that the set union is a meaningful way of combining two columns in the data. Assuming this, the results are supposedly easier to interpret as they are just binary matrices, that is, they represent collections of sets and their unions.

On the other hand, the general idea of CX and CUR decompositions also supports interpretability by selecting actual columns (and rows) of the data. These ‘prototypical representatives’ could give valuable information, for example, when selecting the objects for further studies.

6.3 Computational Complexity of the BCX Problem

The connection between the BCX and BU problems is of course evident, and this connection seems to suggest that BCX is hard even to approximate. Such a strong result, however, is something we do not know how to prove (or even if it is true), but we can only prove a slightly modest result of the NP-completeness of the decision version of the BCX problem (d-BCX). Notice, that this is still more than what is known about the counterparts of BCX, the (nonnegative) Column Subset Selection problem (see Section 5.3).

To study the complexity of the d-BCX problem, first notice that although the columns of $\mathbf{C}$ are known to be from $\mathbf{A}$, this does not make the BU problem any easier, not at least as long as $k < n$ (i.e. as long as not all columns of $\mathbf{A}$ appear in $\mathbf{C}$). The columns of $\mathbf{A}$ present in $\mathbf{C}$ are of course easy to cover, but recall that BU is hard already when $\mathbf{A}$ has only one column. In other words, covering the columns of $\mathbf{A}$ not in $\mathbf{C}$, even if there is only one of them, is a hard problem even to approximate.

Then why is this relation not enough to prove a strong inapproximability result to BCX as well? Because the hardness of BU is based on the adversary selection of the columns in $\mathbf{C}$, something we can decide when solving the BCX problem. Yet, it is possible to construct an instance of BCX where finding the optimal solution requires solving a hard instance of BU, and this is exactly what is done to prove Theorem 6.1. Alas, this reduction does not preserve the approximability.

Theorem 6.1. *The d-BCX problem is* NP*-complete.*

Proof. It is obvious that d-BCX belongs to NP. To show its NP-hardness, we need to reduce d-BU to it. To this end, let $\mathbf{a}$ be an n-dimensional binary vector, $\mathbf{C}$ a binary matrix of size n-by-k, and $t < n$ a nonnegative integer such that $(\mathbf{a}, \mathbf{C}, t)$ is an instance of the d-BU problem with $\mathbf{A}$ having only one column. We will see how to reduce this to an instance of d-BCX.

Create a binary matrix $\mathbf{A}'$ with $n + 2k$ rows and $k(n + k) + 1$ columns. Let the first n rows of the first column of $\mathbf{A}'$ be equal to $\mathbf{a}$, the next k rows be full of 1s, and the last k rows be full of 0s. Copy $\mathbf{C}$ at the top of the next k columns of $\mathbf{A}'$ and below that place two k-by-k identity matrices. Copy these k columns (i.e. not the first column) $n + k - 1$ times. Thus the final matrix $\mathbf{A}'$ contains $n + k$ copies of $\mathbf{C}$, that is,

$$\mathbf{A}' = \begin{pmatrix} \mathbf{a} & \mathbf{C} & & \mathbf{C} \\ \mathbf{1}_k & \mathbf{I}_k & \cdots & \mathbf{I}_k \\ \mathbf{0}_k & \mathbf{I}_k & & \mathbf{I}_k \end{pmatrix},$$

where $\mathbf{1}_k$ is a k-dimensional column vector full of 1s, $\mathbf{0}_k$ is a k-dimensional column vector full of 0s, and $\mathbf{I}_k$ is a k-by-k identity matrix. Let $k' = k$ and $t' = t + k$ to complete the construction of the instance $(\mathbf{A}', k', t')$.

We need to show that the d-BU instance $(\mathbf{a}, \mathbf{C}, t)$ has an answer 'yes' if and only if the constructed d-BCX instance has an answer 'yes'. Let us start by assuming that $\mathbf{x}$ is such that $d_1(\mathbf{A} - \mathbf{C} \circ \mathbf{x}) \leq t$. We can construct $\mathbf{C}'$ by taking one column from each of the $n + k$ identical columns that correspond to the columns of $\mathbf{C}$. The first column of $\mathbf{X}'$ is $\mathbf{x}$. The remaining columns of $\mathbf{X}'$ can map the columns of $\mathbf{C}'$ to their identical columns introducing no cost. The first column of $\mathbf{X}'$ defines a cover of the first n rows of $\mathbf{A}'$ which is exactly the cover defined by $\mathbf{x}$ to $\mathbf{a}$. No matter what columns of $\mathbf{C}'$ are used to cover the first column of $\mathbf{A}'$, the last $2k$ rows introduce error of k proving this side of the claim.

For the other direction, let $\mathbf{C}'$ and $\mathbf{X}'$ be such that $d_1(\mathbf{A}' - \mathbf{C}' \circ \mathbf{X}') \leq t + k$. First we show that $\mathbf{C}'$ contains columns corresponding to each column of $\mathbf{C}$. For a contradiction, assume that this is not the case. It is easy to see that to cover the $n + k$ columns corresponding to the missing column, one must have at least a unit error in each column, yielding total cost of at least $n + k > t + k$. Thus, a column corresponding to each column of $\mathbf{C}$ must be present in $\mathbf{C}'$ and, moreover, the first column of $\mathbf{A}'$ cannot be in $\mathbf{C}'$. Hence, no matter which columns of $\mathbf{C}'$ are used to cover the first column of $\mathbf{A}'$, the last $2k$ rows will introduce an error of k and therefore, to cover $\mathbf{A}'$ with cost at most $t + k$, the first n rows of the first column can introduce error of at most t. □

The above proof does not preserve approximability, and hence tells us nothing about the approximability of the BCX problem. But while we do not know much about the approximability of BCX in general, we do know something about a special case of it, namely, we can show that the exact-BCX problem can be solved in polynomial time.

Proposition 6.2. *The* exact-BCX *problem can be solved in polynomial time with respect to the instance size.*

We need some new terminology before we can proceed to the proof. A binary vector $\mathbf{a} \in \{0, 1\}^n$ is *dominated* by binary vector $\mathbf{b} \in \{0, 1\}^n$ if $a_i = 0$ whenever $b_i = 0$, and we write $\mathbf{a} \leq \mathbf{b}$. This defines a partial ordering on $\{0, 1\}^n$. If I is a subset of $[n]$, we say that $\mathbf{a}$ is dominated by $\mathbf{b}$ *in* I (and write $\mathbf{a}_I \leq \mathbf{b}_I$) if $a_i = 0$ when $b_i = 0$ for all $i \in I$. If $J \subset I$, then $\mathbf{a}_I \leq \mathbf{b}_I$ implies $\mathbf{a}_J \leq \mathbf{b}_J$; in particular, if $\mathbf{a} \leq \mathbf{b}$, then $\mathbf{a}_I \leq \mathbf{b}_I$ for all $I \subseteq [n]$. If $\mathbf{c} \in \{0, 1\}^n$ is a binary vector, we say that $\mathbf{a}$ is dominated by $\mathbf{b}$ in $\mathbf{c}$ ($\mathbf{a}_\mathbf{c} \leq \mathbf{b}_\mathbf{c}$) if $\mathbf{a}_I \leq \mathbf{b}_I$ for $I = \{i : c_i = 0\}$ (notice that it is not $c_i = 1$).

Proof of Proposition 6.2. We are given a pair $(\mathbf{A} \in \{0,1\}^{n \times m}, k)$ and we need to decide whether there exists a subset of k columns of $\mathbf{A}$ (denoted $\mathbf{C}$) and a k-by-m binary matrix $\mathbf{X}$ such that $\mathbf{A} = \mathbf{C} \circ \mathbf{X}$. For this, we construct the matrix $\mathbf{C}$ column-by-column together with matrix $\mathbf{X}$. The idea is to go through the columns of $\mathbf{A}$ in the (partial) order defined by the relation $\leq$, defined above. We will see that this will either give the desired matrix $\mathbf{C}$ or show that no such matrix exists.

We start by selecting the column $\mathbf{a}$ of $\mathbf{A}$ that has the least 1s (assuming all columns have at least one 1 and breaking ties arbitrarily). Vector $\mathbf{a}$ must be in $\mathbf{C}$ because if one wants to express $\mathbf{a}$ as a Boolean combination of other binary vectors, $\mathbf{a} = \mathbf{d} \circ \mathbf{e} \circ \cdots \circ \mathbf{z}$, it must be that $\mathbf{d}, \mathbf{e}, \ldots, \mathbf{z} \leq \mathbf{a}$, and clearly there are no such vectors in $\mathbf{A}$. We continue by removing this column from $\mathbf{A}$ and considering the column that has the least 1s and that does *not* dominate $\mathbf{a}$. By the same argument as above, this matrix must also be in $\mathbf{C}$, and it is removed from $\mathbf{A}$. The next column we consider is the one with the least 1s and that does not dominate either of the selected vectors. This process is continued as long as there are columns that do not dominate any of the selected columns.

Call the matrix containing the columns selected above $C_{(1)}$, and construct the respective matrix $\mathbf{X}_{(1)}$ by setting $(\mathbf{X}_{(1)})_{ij} = 1$ if $\mathbf{c}^j_{(1)} \leq \mathbf{a}^i$ and $x_{ij} = 0$ otherwise. Let $\mathbf{A}'_{(1)} = \mathbf{C}_{(1)} \circ \mathbf{X}_{(1)}$. Following the terminology of previous chapters, we say that an element a_{ij} of $\mathbf{A}$ is *covered* by $\mathbf{A}'$ if $a'_{ij} = 1$. A column of $\mathbf{A}$ is covered completely if all of its 1s are covered. By the construction we know that no a_{ij} that is 0 can be covered by $\mathbf{A}'$. In what follows we only consider the columns of $\mathbf{A}$ that are not covered completely.

Among the remaining columns of $\mathbf{A}$ select the one that has the least number of *uncovered* 1s (again breaking ties arbitrarily). Call the selected vector $\mathbf{a}$. Vector $\mathbf{a}$ must be in $\mathbf{C}$: to be able to cover $\mathbf{a}$ there must be at least one column in $\mathbf{A}$, call it $\mathbf{b}$, that is not in $\mathbf{C}_{(1)}$ and that is dominated by $\mathbf{a}$, but such $\mathbf{b}$ cannot exist as by the above construction we must have some columns $\mathbf{d}, \mathbf{e}, \ldots, \mathbf{z}$ of $\mathbf{C}_{(1)}$ such that $\mathbf{d} \circ \mathbf{e} \circ \cdots \circ \mathbf{z} \leq \mathbf{b} \leq \mathbf{a}$, implying that $\mathbf{b}$ should have less uncovered 1s than $\mathbf{a}$.

We continue by selecting the next column as the column with the least number of 1s not covered by $\mathbf{A}'_{(1)}$ and that do not dominate $\mathbf{a}$. By the same argument as above, that column must also be in the final matrix $\mathbf{C}$. This process is continued until all remaining columns of $\mathbf{A}$ dominate at least one of the recently-selected columns (or are covered completely by $\mathbf{A}_{(1)}$). Add the columns selected in this phase to the columns of $\mathbf{C}_{(1)}$ to obtain the matrix $\mathbf{C}_{(2)}$, and compute the matrix $\mathbf{X}_{(2)}$ as above, that is, $(\mathbf{X}_{(2)})_{ij} = 1$ if $\mathbf{c}^j_{(1)} \leq \mathbf{a}^i$.

We continue as above. In phase l we have matrix $\mathbf{C}_{(l)}$ of columns we know must be in the final matrix $\mathbf{C}$, and matrix $\mathbf{X}_{(l)}$ telling us which 1s of $\mathbf{A}$ we can cover with $\mathbf{C}_{(l)}$ without covering any 0s. Hence, when we consider a column of $\mathbf{A}$ that has the least number of 1s not covered by $\mathbf{A}'_{(l)}$ we know, by the above argument, that it must be in $\mathbf{C}$.

The above process is followed until one of the following conditions hold: (1) we have an exact decomposition, that is, $\mathbf{A} = \mathbf{C}_{(l)} \circ \mathbf{X}_{(l)}$ and $\mathbf{C}_{(l)}$ has at most k columns; or (2) matrix $\mathbf{C}_{(l)}$ has more than k columns.

This process can be done in polynomial time. At each phase we can compute the approximation $\mathbf{A}'_{(l)}$ in polynomial time, and hence also the number of uncovered 1s for each column of $\mathbf{A}$. Because also the relation 'is dominated by' is polynomially computable, we can select each column of $\mathbf{C}$ in polynomial time. Selecting a column of $\mathbf{C}$ is irreversible, and we cannot select more than n columns, showing that the overall process can be done in polynomial time. □

Proposition 6.2 shows that we can achieve at least some sort of approximation guarantee for BCX, namely, the trivial nm for n-by-m matrices $\mathbf{A}$. What the best possible approximation guarantee is remains an open problem.

6.4 The Mixing Matrix Problem

What the BU problem was to the BCX problem, the Mixing Matrix (MM) problem is to the BCUR problem. But when in BU we can, in general, assume that the columns of $\mathbf{C}$ are arbitrary, in MM we must assume that they are columns of $\mathbf{A}$.

Mixing Matrix Problem (MM)**.** Given an n-by-m binary matrix $\mathbf{A}$, an n-by-k_c binary matrix $\mathbf{C}$ containing k_c columns of $\mathbf{A}$, and a k_r-by-m binary matrix $\mathbf{R}$ containing k_r rows of $\mathbf{A}$, find a k_c-by-k_r binary matrix $\mathbf{U}$ that minimizes (6.2).

In the BU problem, the columns of $\mathbf{X}$ are independent, and it was this feature that made the problem applicable to both BMF and BCX. On the other hand, neither the columns nor the rows of $\mathbf{U}$ are independent in the MM problem. It is this why the columns of $\mathbf{C}$ and rows of $\mathbf{R}$ are required to be subsets of columns and rows of $\mathbf{A}$ in the definition of MM, and it is this why the naïve algorithm for MM must try all $2^{k_c k_r}$ different matrices $\mathbf{U}$, whereas a similar algorithm for BU can construct $\mathbf{X}$ column-by-column.

Assume here and henceforth that $k_c = k_r = k$. The role of $\mathbf{U}$ is best clarified when looking a single element of the matrix $\mathbf{C} \circ \mathbf{U} \circ \mathbf{R}$:

$$(\mathbf{C} \circ \mathbf{U} \circ \mathbf{R})_{ij} = \bigvee_{h=1}^{k} c_{ih} \wedge (\mathbf{U} \circ \mathbf{R})_{hj} = \bigvee_{h=1}^{k} \bigvee_{l=1}^{k} c_{ih} \wedge u_{hl} \wedge r_{lj}. \quad (6.3)$$

Thus, unlike with the CX decomposition, each element of $\mathbf{U}$ can potentially affect each element of $\mathbf{C} \circ \mathbf{U} \circ \mathbf{R}$. Nevertheless, we have a relation between MM and $\pm$PSC, analogous to the relation between BU and $\pm$PSC.

Theorem 6.3. *The* MM *problem can be reduced to the* $\pm$PSC *problem in an approximation preserving way.*

Proof. From (6.3) we see that the value of u_{hl} can affect the value of $(\mathbf{C} \circ \mathbf{U} \circ \mathbf{R})_{ij}$ – and thus the accuracy of the decomposition – only when $c_{ih} = r_{lj} = 1$. Denoting the set of interesting index pairs by $I_{ij} = \{(h,l) : c_{ih} = r_{lj} = 1\}$, we may write $(\mathbf{C} \circ \mathbf{U} \circ \mathbf{R})_{ij} = \bigvee_{(h,l)\in I_{ij}} u_{hl}$. Now, consider an instance of the MM problem, that is, a triplet $(\mathbf{A}, \mathbf{C}, \mathbf{R})$. To map this to an instance of $\pm$PSC let $P = \{a_{ij} : a_{ij} = 1\}$, $N = \{a_{ij} : a_{ij} = 0\}$, and $\mathcal{S} = \{S_{1,1}, \ldots, S_{k,k}\}$ with $S_{hl} = \{a_{ij} : (h,l) \in I_{ij}\}$. We also need to be able to recover $\mathbf{U}$ from the $\pm$PSC solution, that is, from the collection $\mathcal{C}$, and this is done by letting $u_{hl} = 1$ if and only if $S_{hl} \in \mathcal{C}$.

To prove that this reduction preserves the approximability, it is enough to show that the cost $d_1(\mathbf{A} - \mathbf{C} \circ \mathbf{U} \circ \mathbf{R})$ is equal to the cost induced by P, N, and $\mathcal{C}$ in $\pm$PSC. To this end, select an arbitrary element a_{ij}. Consider first the case that $a_{ij} = 1$. We know that $(\mathbf{C} \circ \mathbf{U} \circ \mathbf{R})_{ij} = 1$ if and only if $u_{hl} = 1$ for some $(h,l) \in I_{ij}$, in which case there is also a set $S_{hl} \in \mathcal{C}$ such that $a_{ij} \in S_{hl}$. Otherwise the error in both MM and $\pm$PSC is increased by 1. Second, assume that $a_{ij} = 0$. By construction $(\mathbf{C} \circ \mathbf{U} \circ \mathbf{R})_{ij} = 0$ if and only if $u_{hl} = 0$

for all $(h, l) \in I_{ij}$. This is only possible if for all $S_{hl} \in \mathcal{C}$ it holds that $a_{ij} \notin S_{hl}$ and neither solution introduces any error. Otherwise both solutions must introduce a unit error. The claim follows as the errors introduced by the solutions are equivalent for each a_{ij}. □

The above reduction shows that we can use the algorithm for the ±PSC problem to solve MM. If α denotes the number of 1s in $\mathbf{A}$, the algorithm achieves the approximation factor of $2\sqrt{(k^2+\alpha)\log\alpha}$. The reduction also gives us a way to check if the given MM instance has a solution with zero cost, as solving that question is easy for ±PSC. Henceforth we assume that the optimal solution to any MM instance has cost at least 1.

The reduction from ±PSC to MM is not that straight forward. An additional constant factor will be lost, but that does not matter in our asymptotic considerations. Also, the number of positive elements $|P|$ has no logical counterpart in MM in this reduction. Thus, only the quasi-NP lower bound is applicable.

Theorem 6.4. *The ±PSC problem can be reduced to the MM problem preserving the approximability up to constant factors.*

Proof. Let $(P, N, \mathcal{S})$ be an instance of ±PSC with $|P| + |N| = n$ and $|\mathcal{S}| = k$. Construct the matrices $\mathbf{A}$, $\mathbf{C}$, and $\mathbf{R}$ as follows. First make an n-dimensional binary row vector $\mathbf{v}$ with $v_i = 1$ if the ith element is in P and $v_i = 0$ otherwise, and create a binary $nk(k+1)$-by-n matrix $\mathbf{V}$ by setting each of its rows to $\mathbf{v}$. Second, let a k-by-n binary matrix $\mathbf{T}$ be the transpose of the incidence matrix of $\mathcal{S}$. To create the matrix $\mathbf{A}$, place $\mathbf{T}$ below $\mathbf{V}$ and add an n-dimensional row vector full of 0's below them. Finally, add an additional column full of 1s as the $(n+1)$st column of the matrix $\mathbf{A}$, yielding an $(nk(k+1)+k+1)$-by-$(n+1)$ matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{V} & \mathbf{J}_{nk(k+1)\times 1} \\ \mathbf{T} & \mathbf{J}_{k\times 1} \\ \mathbf{0}_{1\times n} & 1 \end{pmatrix} .$$

Let the matrix $\mathbf{C}$ be the last column of $\mathbf{A}$, that is, the column full of 1s, and the matrix $\mathbf{R}$ to be the last $k+1$ rows of $\mathbf{A}$, that is, $\mathbf{R} = \left(\begin{smallmatrix} \mathbf{T} & \mathbf{J}_{k\times 1} \\ \mathbf{0}_{1\times n} & 1 \end{smallmatrix}\right)$. Notice that as $\mathbf{C}$ has only one column, the matrix $\mathbf{U}$ can have only one row. Therefore we can identify $\mathbf{U}$ as a row vector. To create the collection $\mathcal{C}$ from $\mathbf{U}$, let $S_i \in \mathcal{C}$ if and only if $u_i = 1$.

We need to show that if $\frac{\mathsf{cost}_{\mathrm{MM}}(\mathbf{U})}{\mathsf{cost}^*_{\mathrm{MM}}} \leq r$, then $\frac{\mathsf{cost}_{\pm\mathrm{PSC}}(\mathcal{C})}{\mathsf{cost}^*_{\pm\mathrm{PSC}}} \leq 2r$. For that, start by considering $\mathsf{cost}^*_{\mathrm{MM}}$. Notice first that the optimal solution can cover the last column of $\mathbf{A}$ simply by setting $u_{k+1} = 1$. It remains to cover the submatrix $\mathbf{A} = (\,\mathbf{V}^T\ \mathbf{T}^T\ \mathbf{0}_{k\times 1}\,)^T$. Because $\mathbf{V}$ has $nk(k+1)$ identical rows, making a single error in one of its rows will induce an error of $nk(k+1)$, which is k times more than making an error in *each* of the other elements. Therefore any optimal solution must cover $\mathbf{V}$ optimally. But to cover $\mathbf{V}$ it is enough to find a way to cover a single row $\mathbf{v}$, and covering $\mathbf{v}$ optimally is equal to optimally solving the $\pm$PSC instance. The proof of this is similar to the one above.

Any error done in covering $\mathbf{v}$ is multiplied $nk(k+1)$ times in $\mathbf{V}$, and covering the rest of the matrix can add at most $(k+1)n$ error. Thus we have that $\mathsf{cost}^*_{\mathrm{MM}} \leq nk(k+1)\mathsf{cost}^*_{\pm\mathrm{PSC}} + (k+1)n$. Analogously to the optimal costs, the error $\mathbf{U}$ induces in covering $\mathbf{V}$ is $nk(k+1)$ times the error $\mathcal{C}$ induces in total. Thus $nk(k+1)\mathsf{cost}_{\pm\mathrm{PSC}}(\mathcal{C}) \leq \mathsf{cost}_{\mathrm{MM}}(\mathbf{U})$. Plugging in these two inequalities, we get

$$\frac{\mathsf{cost}_{\mathrm{MM}}(\mathbf{U})}{\mathsf{cost}^*_{\mathrm{MM}}} \geq \frac{nk(k+1)\mathsf{cost}_{\pm\mathrm{PSC}}(\mathcal{C})}{nk(k+1)\mathsf{cost}^*_{\pm\mathrm{PSC}} + (k+1)n} \geq \frac{\mathsf{cost}_{\pm\mathrm{PSC}}(\mathcal{C})}{\mathsf{cost}^*_{\pm\mathrm{PSC}} + 1/k}.$$

The claim follows as the optimal cost and k are both at least 1. □

The most intuitive counterpart for $|\mathcal{S}|$ in this reduction is the number k_r of rows in $\mathbf{R}$. By the construction we have $k_r = |\mathcal{S}| + 1 \leq 2\,|\mathcal{S}|$. Transposing the above construction shows that the same holds for k_c. Thus the lower bound result for MM is: for any $\varepsilon > 0$ it is quasi-NP-hard to approximate MM to within a factor of $\Omega(2^{\log^{1-\varepsilon}(\max\{k_r,k_c\}/2)^4})$.

6.5 Algorithms for the BCX, BCUR, and MM Problems

The previous results show that there is little or no hope of finding the best possible BCX and BCUR decompositions within a feasible time, and thus the best we can hope for would be an algorithm with some approximation guarantee. Unfortunately, the algorithms presented here do not come with any approximation guarantees, notwithstanding the lack of any lower bounds for approximability. This increases the importance of careful empirical evaluation of the algorithms, and such evaluation will be carried out in forthcoming sections.

6.5.1 The `LocBCX` Algorithm

A simple local-search algorithm would work as follows: the starting solution would be a random sample of columns of **A** constituting the matrix **C**, and the neighbourhood would be the set of matrices obtained by changing exactly one of **C**'s columns. The `LocBCX` algorithm, however, implements a certain level of greediness in it. In `LocBCX`, columns of **C** are considered one-by-one. When column i is considered, the local search is applied to the neighbourhood consisting of all matrices obtained from **C** by changing column i to some other column, not currently in **C** (as opposed to considering changing *any* column of **C** to any other column). When all columns are considered, the process is restarted if any of the columns was changed, and stopped if no upgrades were possible. This type of local search was chosen after initial tests, where it was noticed to converge faster and – somewhat surprisingly – to better solutions than the more straight forward implementation.

Akin to the `LocNCX` algorithm (see Section 5.4), also the `LocBCX` algorithm needs to solve the cost induced by a specific matrix **C** multiple times during one local search. But unlike with `LocNCX`, computing the cost of a specific matrix **C** exactly is hard even in theory, requiring to solve the BU problem exactly. The solution is, as with the `Asso` algorithm (Algorithm 2), to use the function cover to give a rough estimate of the matrix **X**.

The `LocBCX` algorithm returns matrices **C** and **X**, but as with the `Asso` algorithm, the quality of **X** can be improved after the columns of **C** are fixed. As this is exactly the BU problem, the algorithms used to solve it are those presented in Section 3.6, namely `PelegRB` and `IterX`. The combinations of these two algorithms with `LocBCX` are referred to as `LocBCX+PelegRB` and `LocBCX+IterX`.

6.5.2 The `LocBCUR` Algorithm and Solving the MM Problem

As the NCUR and BCUR problems are very much alike, similar ideas can be applied to both of them. Thus the simplest idea to solve the BCUR problem would be to first solve $\mathbf{C}$ using `LocBCX`, then solve $\mathbf{R}$ using `LocBCX` again, now to a transposed matrix, and finally solve $\mathbf{U}$ using `PelegRB` via the reduction used to prove Theorem 6.3.

This approach has certain drawbacks. First of them is that the reduction from Theorem 6.3 increases the instance size easily beyond what is practically feasible. This, combined with the generally slow performance of `PelegRB` (see Section 3.6.1), makes the final step of the algorithm a bottleneck. The approach used in `IterX` does not immediately apply here, as the columns (or the rows) of $\mathbf{U}$ are not independent, and there is no counterpart to the cover function. Instead, a simpler iterative approach is possible. Starting with a matrix $\mathbf{U}$ full of 0s, each element u_{ij} is considered, and the value of u_{ij} is changed (from 0 to 1 and vice versa in later iterations) if it reduces the reconstruction error. The process is restarted from $u_{1,1}$ until it converges. This algorithm is referred to as `IterU`.

Compared to `IterX`, `IterU` requires more elements to be updated in each iteration, but given the small size of $\mathbf{U}$ (k_c-by-k_r), this is usually still feasible.

An alternative way to solve the MM problem is to take advantage of the structure of the Boolean CUR decomposition. Recall from (6.3), that element u_{hl} can determine the value of $(\mathbf{C} \circ \mathbf{U} \circ \mathbf{R})_{ij}$ only if $c_{ih} = r_{hj} = 1$. As in Section 6.4, denote the set of index pairs (h, l) such that u_{hl} can change the value of $(\mathbf{C} \circ \mathbf{U} \circ \mathbf{R})_{ij}$ by I_{ij}, that is, $I_{ij} = \{(h, l) : c_{ih} = r_{lj} = 1\}$.

The algorithm starts with matrix $\tilde{\mathbf{U}}$ full of 0s and iterates over a_{ij}s. If $a_{ij} = 0$ it decreases the value of $\tilde{u}_{hl}$ by 1 for each $(h, l) \in I_{ij}$, and if $a_{ij} = 1$ it increments the value of $\tilde{u}_{hl}$ by some fixed $b \in]0, 1]$ in the same positions. When this is done, $\mathbf{U}$ is constructed by setting $u_{ij} = 1$ if $\tilde{u}_{ij} > 0$, and $u_{ij} = 0$ otherwise. The balancing variable b is needed because if $a_{ij} = 0$, then for all $(h, l) \in I_{ij}$, u_{hl} must be 0, but if $a_{ij} = 1$ it is enough that $u_{hl} = 1$ for one $(h, l) \in I_{ij}$. Unfortunately there does not seem to be any other way to select the balancing variable b than trying out different values. This algorithm is known as `Maj`, referring to its idea of selecting u_{hl}s based on a (scaled) majority voting.

Another problem with the naïve way of solving BCUR is that columns of $\mathbf{C}$ and rows of $\mathbf{R}$ are selected independently while in the resulting decomposition they depend on each other heavily. The solution to this is, of course, a local-search heuristic similar to `LocNCUR`, where the neighbourhood contains neighbours of $\mathbf{C}$ and neighbours of $\mathbf{R}$.

The problem with this approach is in updating the matrix $\mathbf{U}$ to compute the reconstruction error for each neighbour. In `LocNCUR` a new $\mathbf{U}$ was always computed, using `SolveNU`, but here using either `IterU` or `Maj` to compute $\mathbf{U}$ from scratch can be time-consuming. The answer is to exploit the local structure of possible changes, similar to what was done in `Maj`. The algorithm, again, starts with random $\mathbf{C}$ and $\mathbf{R}$. It then computes $\mathbf{U}$ for these matrices using `Maj`, and keeps also $\tilde{\mathbf{U}}$ used in `Maj`. It then works as follows. Assume a column $\mathbf{c}^h$ is changed to $\tilde{\mathbf{c}}^h$. We only need to consider rows i where $c_{ih} \neq \tilde{c}_{ih}$. Assume i is such a row and j is an arbitrary row. All elements of $\tilde{\mathbf{U}}$ that need to be considered for this pair (i, j) are those from row h and column l such that (h, l) is either in $I_{ij} = \{(h, l) : c_{ih} = r_{lj} = 1\}$ or in $\tilde{I}_{ij} = \{(h, l) : \tilde{c}_{ih} = r_{lj} = 1\}$. Assuming (h, l) is such, the update rules are as follows. If $c_{ih} = 0$ and $\tilde{c}_{ih} = 1$, then let

$$\tilde{u}_{hl} = \begin{cases} \tilde{u}_{hl} + b & \text{if } a_{ij} = 1 \\ \tilde{u}_{hl} - 1 & \text{if } a_{ij} = 0, \end{cases}$$

and if $c_{ih} = 1$ and $\tilde{c}_{ih} = 0$, then let

$$\tilde{u}_{hl} = \begin{cases} \tilde{u}_{hl} - b & \text{if } a_{ij} = 1 \\ \tilde{u}_{hl} + 1 & \text{if } a_{ij} = 0. \end{cases}$$

The same procedure applies *mutatis mutandis* when a row of $\mathbf{R}$ is changed instead of a column of $\mathbf{C}$.

The algorithm implementing the local search for BCUR with the above updates to $\mathbf{U}$ is called `LocBCUR`.

6.5.3 Time Complexity

Analysing the time complexity of the above algorithms requires solving two questions: what is the time a single local improvement step takes and what is the maximum number of steps that can be taken. The latter question is easy to answer: if the input matrix is n-by-m, then the maximum error any approximation can cause is nm, and as each local improvement is guaranteed to improve the result by at least 1, the maximum number of improvements is nm (cf. Section 4.7). This is, of course, a very coarse upper bound, and results in rather pessimistic upper bounds. Tighter analysis is, however, hard to achieve. As the maximum number of steps is the same for all algorithms, the time each algorithm needs to compute a single step becomes even more interesting.

For `LocBCX`, a single iteration consist of trying all columns of $\mathbf{C}$ one-by-one and trying to change them with some other column. Assume the input matrix is n-by-m. There are k columns in $\mathbf{C}$, and $m - k$ other columns to try. For each possible change, we need to compute the matrix $\mathbf{X}$, taking time $O(kmn)$, which also encompasses the time needed to compute the error and other bookkeeping required. Thus the time complexity for one iteration is $O(k(m - k)kmn) = O(k^2mn(m - k))$ yielding $O((knm)^2(m - k))$ total time complexity.

One iteration of `IterU` requires going through the matrix $\mathbf{U}$ (k_ck_r steps), and computing the product $\mathbf{C} \circ \mathbf{U} \circ \mathbf{R}$ in every step ($O(nk_ck_rm)$ time), though in practice this can be improved by considering only those elements of the product that will change. With nm as the maximum number of iterations, the total time complexity is $O\big((nmk_ck_r)^2\big)$.

The `Maj` algorithm is different from other algorithms in that it does not need any iterations, and thus its time complexity is at most $O(nk_ck_rm)$, and in practice we can assume $|I_{ij}| \ll k_ck_r$ for most I_{ij}s.

For the `LocBCUR`, the neighbourhood structure is bigger: each of the k_c columns of $\mathbf{C}$ can be changed to $(m - k_c)$ other columns and each of the k_r rows of $\mathbf{R}$ can be changed to $(n - k_r)$ other rows, totalling $k_c(m - k_c) + k_r(n - k_r)$ neighbours to try. The algorithm was devised to utilize the fact that usually $|I_{ij}| \ll k_ck_r$, but the worst-case bound must still assume $|I_{ij}| = k_ck_r$, yielding $O(nk_ck_rm)$ time for each neighbour, and $O\big((k_c(m-k_c)+k_r(n-k_r))k_ck_r(nm)^2\big)$ in total.

6.6 Experimental Evaluation

As the Boolean CX and CUR decompositions are natural combinations of the problems in the previous two chapters, the experiments are similar to the previous ones. Their purpose and settings are familiar: synthetic data is used to study the effects of various data characteristics in well-controlled settings, and real data is used to verify these results and provide results about the interpretability of the algorithms' results.

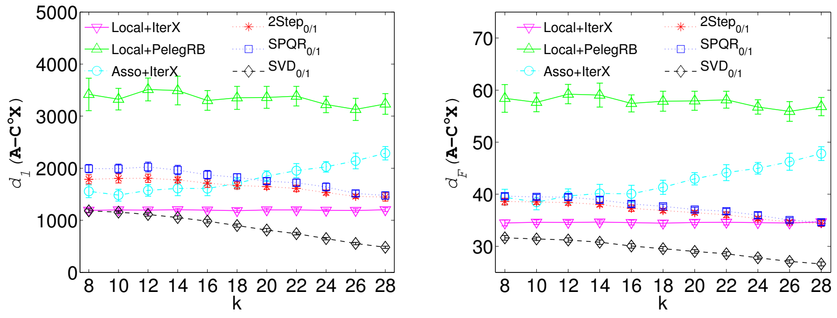
6.6.1 Algorithms Used

Except for the algorithms presented in this chapter, all other algorithms used in the experiments are familiar from the previous chapters; indeed, to the best of the author's knowledge, there are no algorithms for the BCX (and BCUR) decompositions except those presented here.

The BCX decomposition is a Boolean column decomposition, and thus algorithms for it were compared against algorithms for Boolean decomposition (i.e. BMF) and for column decomposition (CX). For the former, `Asso+IterX` was used; for the latter, `2Step` and `SPQR` were used. In addition, SVD was used to provide a benchmark. As with BMF, there are certain problems when comparing results from real-valued methods such as `2Step` or `SPQR` to results from Boolean methods (see Section 4.8.2). Also following the experiments with BMF, two versions of real-valued methods were used, where the results of the modified versions, $\mathtt{SVD}_{0/1}$, $\mathtt{2Step}_{0/1}$, and $\mathtt{SPQR}_{0/1}$, were rounded to binary matrices. Once again, it should be noted that these algorithms are still solving a different problem, and thus they do not tell much about how good an optimal BCX (or BCUR) decomposition would be.

6.6.2 Synthetic BCX Experiments

The synthetic BCX data and the parameter values were the same that were used in Sections 3.7 and 4.8.3. Yet there was one difference. In the aforementioned sections the input data for the algorithms was of the form $\mathbf{A}' = \mathbf{B} \circ \mathbf{X}$, but to satisfy the setting of BCX, the columns of $\mathbf{B}$ (which, for the sake of consistency, is henceforth referred to



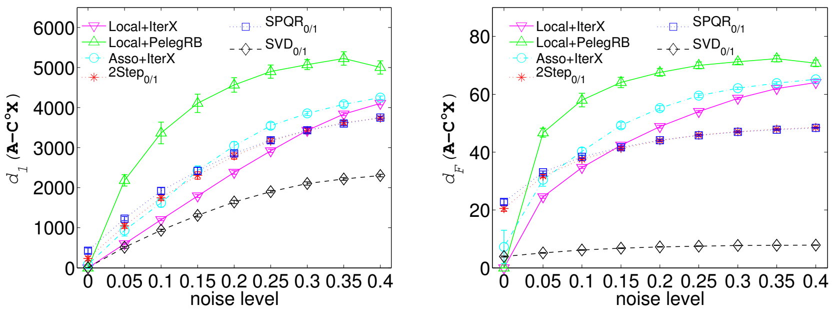
(a) (b)

Figure 6.1: Reconstruction errors of BCX decomposition when k, the number of columns in $\mathbf{C}$, varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is Boolean for `LocBCX+IterX`, `LocBCX+PelegRB`, and `Asso+IterX`, and normal for the other methods.

as $\mathbf{C}$) were appended as the first k columns of $\mathbf{A}'$ yielding the final $\mathbf{A}$ used in the experiments. In other words, when matrices $\mathbf{C}$ and $\mathbf{X}$ were generated as in Section 3.7.1, matrix $\mathbf{X}'$ was set to $(\mathbf{I}_k\ \mathbf{X})$, and $\mathbf{A}$ was constructed as $\mathbf{C} \circ \mathbf{X}'$. The varied parameters and their values were the same as in the aforementioned sections.

Number of columns in C. The number of columns in $\mathbf{C}$, k, varied from 8 to 28 with steps of size 2. The total density of the resulting matrices was approximately 0.35. The results are presented in Figure 6.1. One can see that, first, increasing k does not, in general, seem to have adversarial effects to the proposed algorithms, and second, that `LocBCX+IterX` performs very well: with d_1 distance (Figure 6.1(a)) it is second to none but $\texttt{SVD}_{0/1}$, and even with Frobenius distance it is the second-best up until $k = 26$ when `2Step` and `SPQR` reach its level.

Notice that `LocBCX+IterX` is constantly better than `Asso+IterX`, even though the BMF decomposition presents a theoretical lower bound for the BCX decomposition. These results seem to suggest that when the data has the inherent BCX structure `LocBCX+IterX`, knowing what to look for, is able to find it while `Asso+IterX` fails to find a comparable general BMF decomposition. The `LocBCX+PelegRB` is the worst of all algorithms, and this is in line with the previous results for `PelegRB`. (cf. Sections 3.7 and 4.8.3).



(a) (b)

Figure 6.2: Reconstruction errors of BCX decomposition when the level of noise varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is Boolean for `LocBCX+IterX`, `LocBCX+PelegRB`, and `Asso+IterX`, and normal for the other methods.

Noise level. The level of noise varied from 0 to 0.4 with steps of size 0.05. The results are presented in Figure 6.2. Increasing noise has the expected effect to the error of all methods – it increases. But again `LocBCX+IterX` performs very well, especially with d_1 distance (Figure 6.2(a)), where it is the second-best method (next to only $\texttt{SVD}_{0/1}$) up until 30% of noise; with d_F distance (Figure 6.2(b)) it is second-best until 15% of noise, but constantly the best Boolean method, being always better than `Asso+IterX`.

Density of columns of $\mathbf{B}$. The mean density (number of 1s divided by the total number of elements) of columns of $\mathbf{B}$ varied from 0.05 to 0.3. The total density of the resulting matrices varied from approximately 0.2 to approximately 0.7. The results can be seen in Figure 4.3.

Increasing the density decreases other algorithms' errors somewhat, but `LocBCX+IterX` seems to be immune to it, being second only to `SVD` (and $\texttt{SVD}_{0/1}$) in all evaluation points and with both error measures.

Mean number of columns of $\mathbf{B}$ used for each column of $\mathbf{A}$. The mean number of columns of $\mathbf{B}$ involved in the Boolean combinations used to create columns of $\mathbf{A}$ had three possible values, 4, 6, and 8. The resulting matrices had an approximate total density between 0.35 and 0.55. As above, `LocBCX+IterX` is again the second

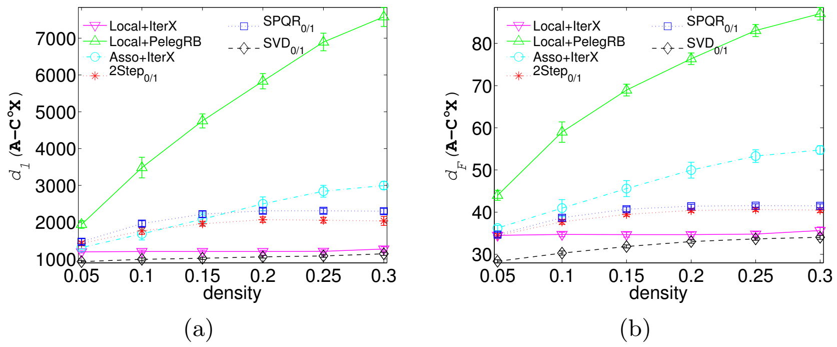


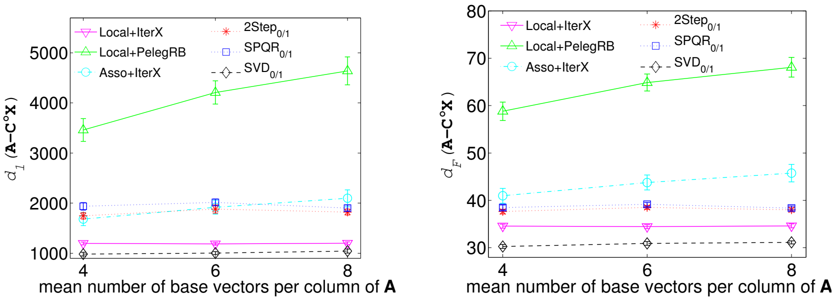
Figure 6.3: Reconstruction errors of BCX decomposition when the density varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is Boolean for `LocBCX+IterX`, `LocBCX+PelegRB`, and `Asso+IterX`, and normal for the other methods.

best method with both distance measures (Figure 6.4), and totally immune to this parameter. Most of the other methods also perform with constant quality, but `Asso+IterX`'s error increase slightly and `LocBCX+PelegRB`'s error increases linearly, as was expected (cf. Section 3.7).

6.6.3 Synthetic BCUR Experiments

The synthetic BCUR data was similar to synthetic NCUR data (see Section 5.6.3), except that it was binary-valued and Boolean matrix multiplication was used in the generation process; the exact generation process is described below. The phenomena studied with these data were analogous to those in previous experiments, namely the effects of (*1*) the number of columns in **C** and the number of rows in **R**; (*2*) the noise level; and (*3*) the density of **C** and **R**.

Data generation process. For a fair experiment, the synthetic data was generated so that it had an exact BCUR decomposition before noise was added. To achieve this, the data generation process used a special type of BCUR decomposition, similar to that used in Section 5.6.3. First, the parameters n and m were set to 150 and 80, respectively. The final matrices had $n + k$ rows and $m + k$ columns, where k is the number of columns in **C**, which was equal



(a) (b)

Figure 6.4: Reconstruction errors of BCX decomposition when the mean number of **C**'s columns used for each column of **A** varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is Boolean for `LocBCX+IterX`, `LocBCX+PelegRB`, and `Asso+IterX`, and normal for the other methods.

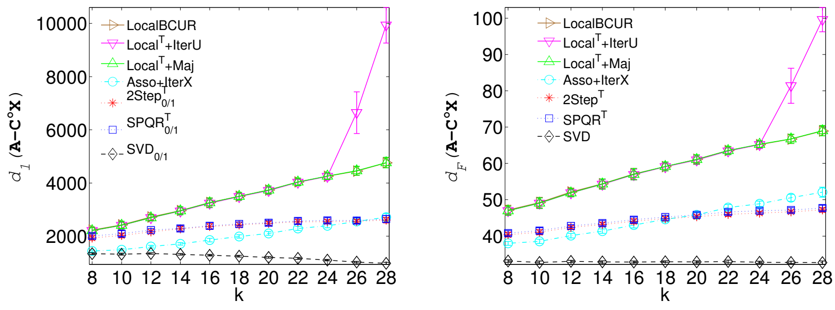
to the number of rows in **R**. The matrices **C** and **R** had a special structure, too:

$$\mathbf{C} = \begin{pmatrix} \mathbf{I}_k \\ \mathbf{X}_{n\times k} \end{pmatrix} \quad \text{and} \quad \mathbf{R} = \begin{pmatrix} \mathbf{I}_k & \mathbf{X'}_{k\times m} \end{pmatrix} .$$

Matrices $\mathbf{X}_{n\times k}$ and $\mathbf{X'}_{k\times m}$ are n-by-k and k-by-m random matrices with elements sampled independently from the Bernoulli distribution. The mixing matrix was a k-by-k identity matrix $\mathbf{I}_k$, and the (noise-free) data matrix was constructed as $\mathbf{A} = \mathbf{C} \circ \mathbf{R}$. The first k columns of **A** are the columns of **C** and the first k rows the rows of **R**. Noise was added by flipping the values of randomly selected elements. Twenty matrices were made for each configuration of parameters, and the default values for parameters were $k = 16$, 10% of noise, and density of 0.1.

Number of columns in C and rows in R. Number of columns in **C** (as well as the number of rows in **R**) varied from $k = 8$ to $k = 28$ with steps of size 2. The density of the resulting matrices varied between 0.14 and 0.26.

The results, presented in Figure 6.5, show that this parameter has small, yet recognizable effect to `LocBCUR` and to `LocBCX`^{T}`+Maj`. These two algorithms report exactly the same results in each data



(a) (b)

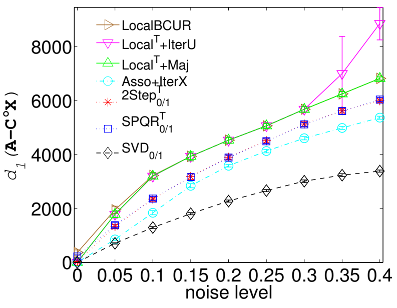
Figure 6.5: Reconstruction errors of BCUR decomposition when the number of columns in $\mathbf{C}$ and rows in $\mathbf{R}$, k, varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is Boolean for `LocBCUR`, `LocBCX`^{T}`+IterU`, `LocBCX`^{T}`+Maj`, and `Asso+IterX`, and normal for the other methods.

points. The third algorithm, `LocBCX`^{T}`+IterU`, faces considerable problems when k increases over 24; no intuitive explanation was found for this sudden and sharp increase in error.

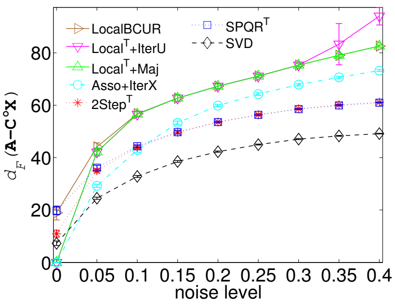
All these three methods lose to the methods they were compared to. The other methods, of course, are not solving BCUR, and thus these results can be considered to point out the hardness of finding a good BCUR decomposition. The hardness of finding good BCUR (or CUR) decompositions is further emphasized by the good performance of `Asso+IterX`. It is worth noting, however, that the relative performances of the methods are almost identical with d_1 and d_F distances.

Noise level. The level of noise varied from 0 to 0.4 with steps of size 0.05. The resulting matrices' density varied between 0.11 and 0.43. The results are presented in Figure 6.6.

The results are similar to those with changing k: `LocBCUR` and `LocBCX`^{T}`+Maj` are (almost) indistinguishable, but `LocBCX`^{T}`+IterU`'s error increases faster with high values of noise. Again, the proposed methods lose to the other methods, yet this time all algorithms (except, perhaps, `SVD`) follow a very similar trend, and the difference between, say, `LocBCUR` and `SPQR`$^T_{0/1}$ is relatively small, at least with d_1 distance.



(a)



(b)

Figure 6.6: Reconstruction errors of BCUR decomposition when the level of noise varies with (a) d_1 and (b) d_F distances. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is Boolean for `LocBCUR`, `LocBCX`^{T}`+IterU`, `LocBCX`^{T}`+Maj`, and `Asso+IterX`, and normal for the other methods.

Density of C and R. The density of the random parts of **C** and **R** varied from 0.05 to 0.3; the density of the resulting matrices varied between 0.12 and 0.65. The results, presented in Figure 6.7, show that density's effects to the proposed algorithms are not straight forward.

The results also rule out one explanation for `LocBCX`^{T}`+IterU`'s behaviour in previous experiments: denser matrices cause problems to `LocBCX`^{T}`+IterU`. Unlike in previous experiments, here `LocBCX`^{T}`+IterU` is the best of the proposed methods, and its results even improve with increased density. The growth in the error caused by `LocBCUR` starts to flatten when the error of `LocBCX`^{T}`+IterU` starts to decrease, but the error caused by `LocBCX`^{T}`+Maj` peaks at that very point. The results of `2Step`$^T_{0/1}$ and `SPQR`$^T_{0/1}$ follow the same trend as `LocBCX`^{T}`+IterU` (Figure 6.7(a)), as do the results of `2Step`^{T} and `SPQR`^{T} when the d_F distance is used (Figure 6.7(b)).

Conclusions. The experiments with synthetic data showed that the proposed algorithms behave much as was expected: increasing noise, as well as increasing the number of columns and rows in the factor matrices, increases the error. Density's effects to the proposed algorithms were not as straight forward, but it is possible that the most problematic case for the algorithms is when the data has density about 1/2.

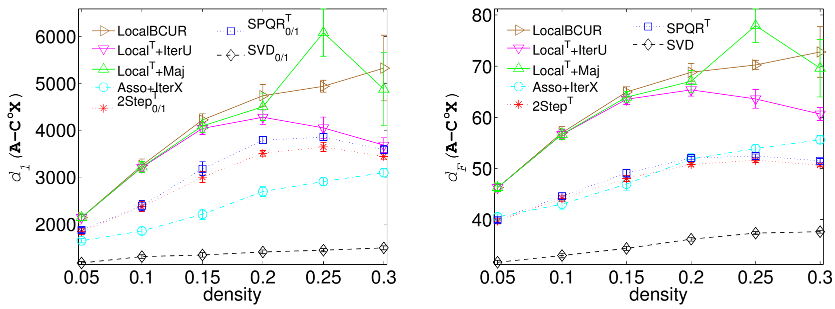


Figure 6.7: Reconstruction errors of BCUR decomposition when the density of $\mathbf{C}$ and $\mathbf{R}$ varies with (a) d_1 and (b) d_F distances. The values of x-axes refer to the success probability of Bernoulli distribution used to create the matrices, not to the actual density of the data. Markers are mean values over twenty instances, and the width of error bars is two times the standard deviation. Matrix multiplication is Boolean for `LocBCUR`, `LocBCX`^{T}`+IterU`, `LocBCX`^{T}`+Maj`, and `Asso+IterX`, and normal for the other methods.

The proposed methods were inferior to the other methods in all experiments. As stated before, one must remember that the other methods are solving different problems, and thus straight forward conclusions about the quality of the proposed algorithms cannot be made. It seems, however, that finding a good BCUR decomposition is a very challenging task, although it is possible that even with the optimal BCUR decomposition one could not attain much smaller reconstruction errors than those of the proposed algorithms.

6.6.4 Experiments with Real-World Data

The experiments done with the real-world data sets are analogous to those in previous chapters. The goals were to verify that the algorithms' behaviour with real-world data follows that observed with synthetic data, to study the significance of the results using randomization, and to study the interpretability of the results. The four real-world data sets used were 4News, DBLP, Dialect, and NOW. The last three were identical to those explained in Section 4.8.4. The first data set was explained in Section 5.6.5 but for the purposes of this chapter, all values greater than 0 were set to 1.

The results are reported in the following order. First, the reconstruction errors for different decomposition sizes are listed. The significance of these results is studied next, and the last part contains the discussion about the interpretability of the results.

Reconstruction errors. The reconstruction errors with respect to d_1 distance are given in Table 6.1, and with respect to d_F distance in Table 6.2.

Of these two tables, Table 6.1 is naturally more interesting. The first thing to note there is that LocBCX's results do not improve when it is combined with IterX, except with Dialect data, where LocBCX+IterX performs much better than plain LocBCX. As could be expected from the previous results, LocBCX+PelegRB's results are inferior to both LocBCX and LocBCX+IterX.

Interestingly, LocBCX's (and LocBCX+IterX's) results with 4News are better than the results of Asso+IterX when k is 5 or 10, albeit the latter should present a lower bound for the former. This suggests that 4News has a strong latent BCX structure which the LocBCX algorithm was able to find. In other data sets, Asso+IterX is better than (or at least as good as) any BCX algorithm, but one could expect an even larger marginal between LocBCX+IterX and Asso+IterX.

The rounded 2Step algorithm, $\texttt{2Step}_{0/1}$, shows good performance over all datasets. In the first four rows of Table 6.1 its reconstruction error is bigger than the error caused by LocBCX+IterX, but in the remaining rows it is smaller. The rounded SPQR, $\texttt{SPQR}_{0/1}$, on the other hand, has more variations in its results against LocBCX+IterX: with 4News it is constantly worse, with DBLP and $k = 5$ it causes enormous error, but with the same data and $k = 15$, it has very small reconstruction error, being more than ten times better than LocBCX+IterX. The rounded SVD is superb in its reconstruction error when compared against all other methods; a sole exception to this is DBLP data with $k = 15$.

The case of DBLP data set with $k = 15$ is interesting for other reasons, as well. Namely, all studied methods obtain major decrease in the reconstruction errors of the data when k increases from 10 to 15. For LocBCX the error with $k = 10$ is almost 4 times the error with $k = 15$, while for $\texttt{SPQR}_{0/1}$ the error with $k = 10$ is 22.5 times the error with $k = 15$.

When the d_F distance is used instead of d_1 distance, the results change only in that the continuous methods, namely, 2Step, SPQR,

Table 6.1: Reconstruction errors of BCX and CX decompositions with real-world data and d_1 distance.

Data	k	`LocBCX`	`LocBCX` `+IterX`	`LocBCX` `+PelegRB`	`Asso` `+IterX`	$\texttt{2Step}_{0/1}$	$\texttt{SPQR}_{0/1}$	$\texttt{SVD}_{0/1}$
4News	5	1807	1807	1876	1814	1916	1972	1701
	10	1605	1602	1731	1613	1737	1780	1348
	15	1439	1439	1585	1431	1449	1539	1075
DBLP	5	9250	9250	9537	8676	9827	14420	8384
	10	5423	5423	5602	4503	5367	2727	3863
	15	1519	1519	1519	1519	774	120	988
Dialect	5	59192	55759	107858	55057	53967	67677	40846
	10	54018	43952	106560	41649	42339	52457	28136
	15	49606	36850	105557	35825	35318	42972	22379
NOW	5	1625	1625	1881	1569	1664	1701	1379
	10	1450	1446	1791	1396	1427	1384	1020
	15	1332	1310	1717	1272	1208	1173	760

Table 6.2: Reconstruction errors of BCX and CX decompositions with real-world data and d_F distance.

Data	k	LocBCX	LocBCX +IterX	LocBCX +PelegRB	Asso +IterX	2Step	SPQR	SVD
4News	5	42.51	42.51	43.31	42.59	39.86	39.32	36.25
	10	40.06	40.02	41.61	40.16	36.69	36.47	33.04
	15	37.93	37.93	39.81	37.83	33.94	34.16	30.38
DBLP	5	96.18	96.18	97.66	93.15	87.49	101.44	81.01
	10	73.64	73.64	74.85	67.10	62.85	73.48	58.27
	15	38.97	38.97	38.97	38.97	37.70	49.10	33.11
Dialect	5	243.29	236.13	328.42	234.64	205.73	228.46	177.72
	10	232.42	209.65	326.44	204.08	181.82	202.03	152.09
	15	222.72	191.96	324.90	189.27	165.67	184.30	137.53
NOW	5	40.31	40.31	43.37	39.61	36.05	36.12	31.93
	10	38.08	38.03	42.32	37.36	32.76	32.60	28.46
	15	36.50	36.19	41.44	35.67	29.92	29.90	25.84

and SVD, perform consistently better than the Boolean ones. Nevertheless, LocBCX+IterX performs relatively well even when compared to SVD.

The reconstruction errors of BCUR and CUR decompositions are reported in Tables 6.3 (for d_1) and 6.4 (for d_F). No BCUR (or CUR) decomposition was done to the DBLP data as it has only 19 rows. The LocBCUR algorithm was considered to take too much time with the Dialect data, and it was omitted. Overall, it was clearly the slowest method.

With 4News data and when k was 5 or 10, the LocBCUR algorithm was slightly worse than LocBCX^T+IterU; when $k = 15$, it was slightly better. With NOW data, LocBCUR gave the best BCUR decomposition. Between LocBCX^T+IterU and LocBCX^T+Maj the former was constantly better than the latter, albeit the difference was sometimes very small.

On the contrary to the BCX results, the $\mathtt{2Step}_{0/1}^{T}$ algorithm was not constantly better than the best of BCUR algorithms. Indeed, it presented smaller error only in three cases: with Dialect and $k = 15$, and with NOW and k being 10 and 15. The results for $\mathtt{SPQR}_{0/1}^{T}$ were similar, it being better than the best of BCUR algorithms only with NOW and $k = 15$. What was said also holds if we do not consider LocBCUR at all, that is, if we compare $\mathtt{2Step}_{0/1}^{T}$ and $\mathtt{SPQR}_{0/1}^{T}$ to LocBCX^T+IterU only.

Perhaps the most interesting result with the d_F distance is that both LocBCX^T+IterU and LocBCX^T+Maj are better than SPQR^T with Dialect data and k being 5 or 10.

Randomization results. The data was randomized using the swap randomization method, as described in Section 4.8.5. Only d_1 distance was considered, as it is the distance measure used by the proposed algorithms. The results of the randomization experiments tell in how many random data sets the algorithm was able to perform better than it performed on the original data.

The results for BCX and rounded CX decompositions are given in Table 6.5. The first thing to notice is LocBCX+PelegRB's column: the randomized data yields smaller reconstruction error in almost every case. But this is not as surprising as it might look at first sight. Recall that the approximation guarantee of PelegRB depends on the number of 1s in the columns of the data matrix, and as was shown in Section 3.7, the algorithm's performance actually depend

Table 6.3: Reconstruction errors of BCUR and CUR decompositions with real-world data and d_1 distance. Missing results are denoted by —.

Data	k	`LocBCUR`	`LocBCX`^{T} `+IterU`	`LocBCX`^{T} `+Maj`	`Asso` `+IterX`	`2Step`$^T_{0/1}$	`SPQR`$^T_{0/1}$	`SVD`$_{0/1}$
4News	5	1912	1896	1908	1814	2082	2148	1701
	10	1794	1755	1773	1613	1865	2001	1348
	15	1642	1651	1652	1431	1660	1792	1075
Dialect	5	—	58799	59034	55057	64944	80452	40846
	10	—	48438	50817	41649	49773	65746	28136
	15	—	45160	52527	35825	42009	53722	22379
NOW	5	1743	1874	1880	1569	1882	1920	1379
	10	1658	1740	1804	1396	1633	1738	1020
	15	1623	1731	1753	1272	1503	1545	760

Table 6.4: Reconstruction errors of BCUR and CUR decompositions with real-world data and d_F distance. Missing results are denoted by —.

Data	k	`LocBCUR`	`LocBCX`^{T} `+IterU`	`LocBCX`^{T} `+Maj`	`Asso` `+IterX`	`2Step`^{T}	`SPQR`^{T}	`SVD`
4News	5	43.73	43.54	43.68	42.59	42.03	42.86	36.25
	10	42.36	41.89	42.11	40.16	39.90	40.54	33.04
	15	40.52	40.63	40.64	37.83	37.44	38.71	30.38
Dialect	5	—	242.49	242.97	234.64	225.64	247.09	177.72
	10	—	220.09	225.43	204.08	201.44	226.12	152.09
	15	—	212.51	229.19	189.27	186.04	209.24	137.53
NOW	5	41.75	43.29	43.36	39.61	38.60	38.97	31.93
	10	40.72	41.71	42.47	37.36	35.67	36.59	28.46
	15	40.29	41.61	41.87	35.67	33.72	34.37	25.84

on this. The randomization technique applied keeps the marginal sums constant, in other words, the number of 1s in data matrix's columns stays constant over the random samples. But what is surprising, is that in many cases, the `LocBCX+PelegRB` algorithm performs worse in the original data than it performs in *any* of the random samples.

The `LocBCX+PelegRB` algorithm is not the only one performing better in the randomized versions of the DBLP data. For k equals to 5 and 10, the results of `LocBCX`, `LocBCX+IterX`, and $\texttt{2Step}_{0/1}$ can be considered to be an artifact of the marginal sums more than the other structures of the data (or, at least, such hypothesis cannot be rejected). But, interestingly enough, with $k = 15$, all these methods, `LocBCX+PelegRB` included, have the reconstruction error with the original data smaller than with any randomized version of the data. This is perhaps explained by the huge drop in reconstruction errors when k increases from 10 to 15 (cf. above). For other data sets we can safely assume that the algorithms, `LocBCX+PelegRB` excluded, did not found their results only because the marginal sums of the data.

The results for the randomization tests with BCUR and rounded CUR decompositions are given in Table 6.6. There, the results for which we cannot reject the null hypothesis are those obtained for 4News data using $\texttt{LocBCX}^T\texttt{+IterU}$, $\texttt{LocBCX}^T\texttt{+Maj}$, and $\texttt{2Step}^T_{0/1}$, albeit for $\texttt{2Step}^T_{0/1}$ this is only true for $k = 5$, and – depending on the level of confidence we want to have – perhaps also for $k = 10$. No straight forward reasons were found for why these algorithms work so well with the randomized versions of 4News.

Interpretability. Not all data sets, nor all algorithms, were used for interpretability tests. For example, interpreting the results from the NOW data would require knowledge on palaeontology, and the results from `LocBCX+PelegRB` are inferior in reconstruction error and, according to the randomization tests, of questionable statistical significance.

The first results are from 4News data, using `LocBCX`. As the data is a terms-by-documents matrix, the BCX decomposition of it would select some documents in $\mathbf{C}$. The actual documents selected are not the only thing that is interesting here, but also the newsgroups to which the selected documents belong. Assuming all newsgroups have their own special vocabulary, and the news have approximately

Table 6.5: Percentage of random data sets having smaller BCX or CX reconstruction error than the original one, d_1 distance. Missing results are denoted by —.

Data	k	`LocBCX`	`LocBCX` `+IterX`	`LocBCX` `+PelegRB`	`Asso` `+IterX`	`2Step`$_{0/1}$	`SPQR`$_{0/1}$	`SVD`$_{0/1}$
4News	5	0	0	65	0	0	0	0
	10	0	0	85	0	0	0	0
	15	0	0	79	0	0	0	0
DBLP	5	100	100	100	0	49	0	0
	10	100	100	100	0	82	0	0
	15	0	0	0	0	0	0	0
Dialect	5	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0
NOW	5	0	0	100	0	0	0	0
	10	0	0	100	0	0	0	0
	15	0	0	100	0	0	0	0

Table 6.6: Percentage of random data sets having smaller BCUR or CUR reconstruction error than the original one, d_1 distance. Missing results are denoted by —.

Data	k	LocBCUR	LocBCX^T +IterU	LocBCX^T +Maj	Asso +IterX	2Step$^T_{0/1}$	SPQR$^T_{0/1}$	SVD$_{0/1}$
4News	5	1	12	38	0	83	0	0
	10	0	17	33	0	1	0	0
	15	0	17	15	0	0	0	0
Dialect	5	—	0	0	0	0	0	0
	10	—	0	0	0	0	0	0
	15	—	0	0	0	0	0	0
NOW	5	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0

Table 6.7: Newsgroups of posts in **C** of BCX decomposition of 4News data. The `LocBCX` algorithm.

$k = 4$	$k = 6$	$k = 8$
`christian`	`christian`	`christian`
`med`	`crypt`	`crypt`
`crypt`	`christian`	`christian`
`crypt`	`crypt`	`crypt`
	`space`	`space`
	`med`	`christian`
		`med`
		`space`

the same frequency of terms over the newsgroups, with $k = 4$ we could assume the algorithm to select one post from each group. The results for various values of k are presented in Table 6.7.

As can be seen from Table 6.7, the assumption fails: for $k = 4$, the algorithm selects two news posts from `sci.crypt` and none from `sci.space`. Newsgroups `sci.crypt` and `soc.religion.christian` tend to appear more often than `sci.med` and `sci.space` when k is increased. Why is this? The assumption of equidense submatrices is false. The submatrix containing the `soc.religion.christian` newsgroup has density of approximately 0.08 and the submatrix containing the `sci.crypt` newsgroup has density of approximately 0.07, but the submatrices containing `sci.med` and `sci.space` have densities of approximately 0.05. Thus the number of posts selected from different newsgroups is not equal, but rather reflects the underlying density.

One could infer the topics of two or three of the newsgroups given the documents selected by `LocBCX` with $k = 8$: an article with words 'encrypt', 'public', and 'chip' suggests the cryptography quite clearly; an article with words 'faith', 'Christian', and 'God' the Christian religion; and an article that could suggest space as the topic of one of the newsgroups was a laconic single-word article: 'space'. Cryptography and Christian religion were also clear topics of two other posts, whereas the remaining three posts contained more general words. Although one of the posts was from `sci.med`, the terms present in it did not suggest specifically medical content.

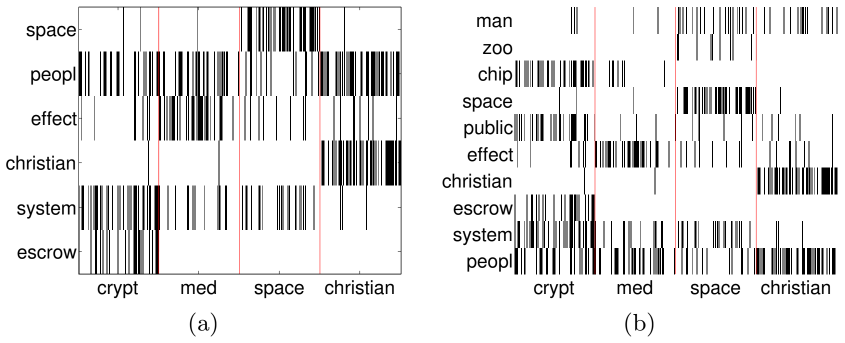


Figure 6.8: Terms selected, and their appearance, by LocBCX in BCX decomposition of a transpose of 4News. Labels on y axis represent the selected terms, and labels on x axis the newsgroups. (a) $k = 6$; (b) $k = 10$.

Lack of clearly medical terminology was also recognized below when the terms were selected instead of documents.

The LocBCX algorithm was executed with transposed 4News as input, this time to select some terms. The submatrices of the selected terms for $k = 6$ and $k = 10$ are presented in Figure 6.8. The algorithm clearly selects both terms specific to a single newsgroup and terms appearing frequently in all newsgroups. For example, from Figure 6.8(a) we can see that the term ‘space’ – not surprisingly – appears almost solely in newsgroup `sci.space`, whereas the term ‘peopl[e]’ appears in all newsgroups.

One should be able to identify the topics of two newsgroups given Figure 6.8(a) (without x labels, of course); the terms ‘space’ and ‘christian’ are rather suggestive, and define the borders of the newsgroups well. The border between `sci.crypt` and `sci.med` can be decided using the term ‘escrow’, but to define the topics of the newsgroups is not that easy without other information.

A BCUR decomposition of the Dialect data was obtained using the LocBCUR and LocBCX^T+IterU algorithms for $k_c = k_r = 7$ and 8. The results are presented in Figure 6.9.

From Figure 6.9(a) we can see that LocBCUR with $k = 7$ recreates the proposed ground truth (see Figure 4.5 at page 83) up to some accuracy: the far north dialects are combined with the Central and North Ostrobothnian dialects, and this dialect is also present in a narrow area between Tavastian and Savonian dialects; the small

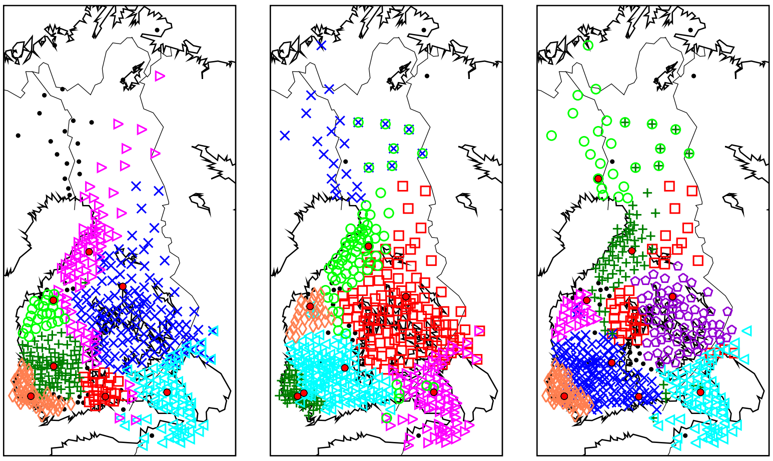

(a) (b) (c)

Figure 6.9: Spread of selected features (rows of $\mathbf{R}$) and the selected municipalities (columns of $\mathbf{C}$) as reported by (a) `LocBCUR` with $k = 7$, (b) `LocBCX`^{T}`+IterU` with $k = 7$, and (c) `LocBCX`^{T}`+IterU` with $k = 8$.

central south-western dialects are migrated to Tavastian dialects; and a new dialect is found in central Southern Finland.

When `LocBCX`^{T}`+IterU` is used, the results change somewhat. First notable thing in Figures 6.9(b) and 6.9(c) is that the selected municipalities do not correspond with the spreads of the selected features, unlike with, say, `LocBCUR`. This could be expected, as with `LocBCX`^{T}`+IterU` the two are selected independently of each other, whereas with `LocBCUR` they are selected concurrently. With $k = 7$, the spread of the features does not present any big anomalies: the central south-western dialects were, once again, combined with south-western dialects, but all other dialects are present, if with somewhat different borders. With $k = 8$ the new dialect does not appear in place of central south-western dialects, but instead splits the Savonian dialects, and the dialect of far north spreads in some new municipalities.

Conclusions. The experiments with the real-world data agreed with results obtained in synthetic settings, confirming the results of the previous section. The proposed algorithms performed relatively well against other methods, showing also the meaningfulness of the BCX and BCUR concepts. Randomization tests revealed some results that seemed to be more a consequence of the marginal sums of the data than any deeper latent structure. The proposed algorithms were not the only ones to get such results, and the case emphasizes how one should be careful when one presents claims about latent structures supposedly found by one's algorithms. The interpretability of the results was somewhat twofold: the results were mostly intuitive, but – perhaps due to the very restrictive nature of BCX (and BCUR) decompositions – they were not able to fully encompass the most important aspects of the data. Nevertheless, the proposed methods provide an enhancement to data miners' toolbox for binary data.

CHAPTER 7

Conclusions

This thesis studied six matrix decompositions, namely, BMF, BMP, NCX, NCUR, BCX, and BCUR. The last four of them are novel formulations, BMP is semi-novel – it is essentially a well-known problem, but rephrased as a matrix decomposition – and BMF is not new, but its use in data mining is. The key motivation behind all decompositions was the interpretability of the results; restrictions were placed for factor matrices to increase the interpretability. This does not mean that the results would always be interpretable, though – different applications and data call for different methods. New matrix decompositions will likely be introduced in the future, addressing other issues and emphasizing different aspects of interpretability.

The main issue with the studied decompositions is their computational complexity (or lack of knowledge thereof). Most of the known results concerning decompositions using Boolean algebra can be traced back to the properties of ±PSC, and for BCUR, for which nothing is known, it is certainly plausible that similar results exist. The computational complexity, and possible hardness of approximation, of NCX and NCUR (or CX and CUR) is an interesting open problem, and is hopefully solved in the near future.

The hardness of constructing the proposed decompositions limits the worst-case performance of any algorithm for the task – often the best that could be hoped for is not much better than what any naïve algorithm could produce. For this reason, the focus in this thesis was on applying heuristics to find decompositions, and measuring their empirical performance in constructed as well as real-world instances.

The experiments demonstrated the algorithms' capabilities on producing meaningful results and compared their reconstruction errors to other methods. No proposed algorithm was able to outperform `SVD` or `NMF`, notwithstanding the fact that BMF decomposition could, in some cases, yield smaller reconstruction errors. To some extent the somewhat poor performance of the Boolean methods can be considered to be a consequence of the hardness of comparing real-valued and Boolean methods, but it also shows the hardness of finding such decompositions. Nevertheless, if one is willing to sacrifice some reconstruction accuracy for higher interpretability, then the proposed methods should present a viable alternative.

For the real-valued algorithms for NCX and NCUR the situation was somewhat different. These algorithms were, in some cases, even better than the algorithms for CX and CUR, and close to the theoretical lower bound presented by NMF and SVD.

Throughout this thesis the size of the decomposition, k, has been defined by the user as a part of the problem instance. This approach is desirable when, say, the user has expert knowledge of the data or of the proper size of the decomposition. But on the other cases, no such knowledge exists or it cannot be used. Then a method of automatically selecting the size of the decomposition is required. These model-selection methods are not covered in this thesis, but they definitely form an interesting and important topic for future research.

The main focus of this thesis was not on the scalability of the algorithms. Many heuristics applied, for example local search, are regularly employed in practical applications. Yet their scalability to huge data sets is somewhat questionable. The purpose of the algorithms was to provide first examples of methods that could construct the proposed decompositions, and to (empirically) verify that it can be done despite the theoretical complexity of the problems concerned. For the sake of clear and succinct representation the algorithms, or the implementations thereof, did not employ many standard techniques to speed up the computation such as different pruning methods for local search.

Another topic not discussed in this thesis is parallel computation, while many of the proposed methods are, in fact, suitable for at least some level of parallelization. Developing faster, and more accurate, algorithms for the tasks is an important topic for future research.

The decompositions presented in this thesis are intuitive and have many properties that make them attractive methods for data mining. They also have proved to have applications outside the field of data mining. It is therefore to be hoped that future research will provide the missing results, close the current gaps, and present more efficient algorithms, hopefully with provable approximation guarantees. But the most important goal, and a prerequisite for the other goals, is to increase the awareness of the Boolean and column decompositions in all fields of computer science.

References

[ABS08] Marcel R. Ackermann, Johannes Blömer, and Christian Sohler. Clustering for metric and non-metric distance measures. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*, pages 799–808, 2008.

[AGK$^+$04] Vijay Arya, Naveen Garg, Rohit Kjandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[AGM04] Foto Afrati, Aristides Gionis, and Heikki Mannila. Approximating a collection of frequent sets. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*, pages 12–19, 2004.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[AM07] Dimitris Achlioptas and Frank McSherry. Fast computation of low-rank matrix approximations. *Journal of the ACM*, 54(2), 2007. Article 9.

[AMS06] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006.

[AV07] David Arthur and Sergei Vassilvitskii. `k-means++`: The advantages of careful seeding. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, pages 1027–1035, 2007.

[Bal89] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[BBL$^+$07] Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis*, 52(1):155–173, 2007.

[BDG$^+$04] Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. A generalized maximum entropy approach to Bregman co-clustering and matrix approximations. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*, pages 509–514, 2004.

[BJ06] Wray Buntine and Aleks Jakulin. Discrete component analysis. In *Subspace, Latent Structure and Feature Selection*, volume 3940 of *Lecture Notes in Computer Science*, pages 1–33, 2006.

[BKF09] Ella Bingham, Ata Kabán, and Mikael Fortelius. The aspect Bernoulli model: Multiple causes of presences and absences. *Pattern Analysis & Applications*, 12(1):55–78, 2009.

[BMD08] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. Unsupervised feature selection for principal components analysis. In *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*, pages 61–69, 2008.

[BMD09] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proc. 19th Annual*

ACM-SIAM Symposium on Discrete Algorithms (SODA '09), pages 968–977, 2009.

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[BPRB06] Jérémy Besson, Ruggero G. Pensa, Céline Robardet, and Jean-François Boulicaut. Constraint-based mining of fault-tolerant patterns from Boolean data. In *Knowledge Discovery in Inductive Databases: 4th International Workshop, KDID*, volume 3933 of *Lecture Notes in Computer Science*, pages 55–71, 2006.

[BPS05] Michael W. Berry, Shakhina A. Pulatova, and G. W. Stewart. Algorithm 844: Computing sparce reduced-rank approximations to sparce matrices. *ACM Transactions on Mathematical Software*, 31(2):252–269, 2005.

[Bun02] Wray Buntine. Variational extensions to EM and multinomial PCA. In *Proc. 13th European Conference on Machine Learning (ECML '02)*, volume 2430 of *Lecture Notes in Computer Science*, pages 23–34, 2002.

[BV06] Radim Bělohlávek and Vilém Vychodil. On Boolean factor analysis with formal concepts as factors. In *Joint Proc. International Conference on Soft Computing and intelligent systems and International Symposium on Advanced Intelligent systems (SCIS & ISIS '06)*, pages 1054–1059, 2006.

[CC03] George W. Cobb and Yung-Pin Chen. An application of Markov chain Monte Carlo to community ecology. *The American Mathematical Monthly*, 110(4):265–288, 2003.

[CDKM00] Robert D. Carr, Srinivas Doddi, Goran Konjevod, and Madhav Marathe. On the red-blue set cover problem. In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 345–353, 2000.

[CG99] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc. 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*, pages 378–388, 1999.

[Che06] Ke Chen. On k-median clustering in high dimensions. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*, pages 1177–1185, 2006.

[Chv79] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[CKX06] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *Proc. Mathematical Foundations of Computer Science (MFCS)*, volume 4162 of *Lecture Notes in Computer Science*, pages 238–249, 2006.

[CKY06] Marek Chrobak, Claire Kenyon, and Neal Young. The reverse greedy algorithm for the metric k-median problem. *Information Processing Letters*, 97:68–72, 2006.

[ÇMI07] Ali Çivril and Malik Magdon-Ismail. Finding maximum volume sub-matrices of a matrix. Technical Report 07-08, Department of Computer Science, Rensselaer Polytechnic Institute, 2007.

[CR93] Joel E. Cohen and Uriel G. Rothblum. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra and Its Applications*, 190:149–168, 1993.

[DF95] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in computer science. Springer, New York, 1999.

[DKM06a] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.

[DKM06b] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36(1):184–206, 2006.

[DMM06a] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-based methods. In *Proc. 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 10th International Workshop on Randomization and Computation (APPROX + RANDOM)*, volume 4110 of *Lecture Notes in Computer Science*, pages 316–326, 2006.

[DMM06b] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-row-based methods. In *Proc. 14th European Symposium on Algorithms (ESA '06)*, volume 4168 of *Lecture Notes in Computer Science*, pages 304–314, 2006.

[DMM07] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. Technical report, August 2007. arXiv:0708.3696v1 [cs.DS].

[DS04] Irit Dinur and Shmuel Safra. On the hardness of approximating label-cover. *Information Processing Letters*, 89(5):247–254, 2004.

[EP00] Michael Elkin and David Peleg. The hardness of approximating spanner problems. In *Proc. 17th Symposium on Theoretical Aspects of Computer Science (STACS '00)*, volume 1770 of *Lecture Notes in Computer Science*, pages 370–381, 2000.

[EW97] Sheila Embleton and Eric S. Wheeler. Finnish dialect atlas for quantitative studies. *Journal of Quantitative Linguistics*, 4(1–3):99–102, 1997.

[EW00] Sheila M. Embleton and Eric S. Wheeler. Computerized dialect atlas of Finnish: Dealing with ambiguity. *Journal of Quantitative Linguistics*, 7(3):227–231, 2000.

[EY36] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[F^+03] M. Fortelius et al. Neogene of the old world database of fossil mammals (NOW). Online, 2003. `http://www.helsinki.fi/science/now/`, Accessed 8 December 2008.

[Fei98] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, 2006.

[FGJM06] Mikael Fortelius, Aristides Gionis, Jukka Jernvall, and Heikki Mannila. Spectral ordering and biochronology of European fossil mammals. *Paleobiology*, 32(2):206–214, 2006.

[FKV04] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041, 2004.

[GE96] Ming Gu and Stanley C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

[GGM04] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *Proc. Discovery Science: 7th International Conference*, volume 3245 of *Lecture Notes in Computer Science*, pages 278–289, 2004.

[GH06] Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys*, 38(3):Article 9, 2006.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability: A guide to the theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison–Wesley, Upper Saddle River, NJ, second edition, 1994.

[GMMT07] Aristides Gionis, Heikki Mannila, Taneli Mielikäinen, and Panayiotis Tsaparas. Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data*, 1(3), 2007.

[GMS04] Aristides Gionis, Heikki Mannila, and Jouni K. Seppänen. Geometric and combinatorial tiles in 0–1 data. In *Proc. Principles and Practice of Knowledge Discovery in Databases (PKDD '04)*, volume 3202 of *Lecture Notes in Computer Science*, pages 173–184, 2004.

[GP83] D. A. Gregory and N. J. Pullman. Semiring rank: Boolean rank and nonnegative rank factorizations. *Journal of Combinatorics, Information & System Sciences*, 8(3):223–233, 1983.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, third edition, 1996.

[GW95] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[Har72] J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, San Francisco, 2001.

[HMS01] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining.* The MIT Press, Cambridge, Massachusetts, 2001.

[HMT08] Saara Hyvönen, Pauli Miettinen, and Evimaria Terzi. Interpretable nonnegative matrix decompositions. In *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*, pages 345–353, 2008.

[Hof99] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proc. 22nd ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50–57, 1999.

[Itk65] Terho Itkonen. *Proto-Finnic Final Consonants: Their history in the Finnic languages with particular reference to the Finnish dialects, part I: 1, Introduction and The History of* -k *in Finnish*, volume 138(1) of *Mémoires de la Société Finno-Ougrienne*. Suomalaisen Kirjallisuuden Kirjapaino Oy, Helsinki, 1965.

[Joh74] David S. Johnson. Approximation algorithms for combinatiorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[KGR05] Mehmet Koyutürk, Ananth Grama, and Naren Ramakrishnan. Compression, clustering, and pattern discovery in very-high-dimensional discrete-attribute data sets. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):447–461, 2005.

[Kim82] Ki Hang Kim. *Boolean matrix theory and applications.* Marcel Dekker, Inc., New York, 1982.

[KO98] Tamara G. Kolda and Dianne P. O'Leary. A semidiscrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Transactions on Information Systems*, 16(4):322–346, 1998.

[KO00] Tamara G. Kolda and Dianne P. O'Leary. Algortihm 805: Computation and uses of the semidiscrete matrix decomposition. *ACM Transactions on Mathematical Software*, 26(3):415–435, 2000.

[KTH79] M. K. Kozlov, S. P. Tarasov, and L. G. Hačijan. Polynomial solvability of convex quadratic programming. *Soviet Mathematics. Doklady*, 20(5):1108–1111, 1979.

[Li08] Tao Li. Clustering based on matrix approximation: a unifying view. *Knowledge and Information Systems*, 17(1):1–15, 2008.

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[LS99] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[LVA08] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. Optimal Boolean matrix decomposition: Application to role engineering. In *Proc. 24th IEEE International Conference on Data Engineering (ICDE '08)*, pages 297–306, 2008.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[Mie06] Pauli Miettinen. The discrete basis problem. Master's thesis, Report C-2006-010, Department of Computer Science, University of Helsinki, 2006.

[Mie08a] Pauli Miettinen. The Boolean column and column-row matrix decompositions. *Data Mining and Knowledge Discovery*, 17(1):39–56, 2008.

[Mie08b] Pauli Miettinen. On the positive–negative partial set cover problem. *Information Processing Letters*, 108(4):219–221, 2008.

[Mir60] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *The Quarterly Journal of Mathematics. Oxford. Second series*, 11:50–59, 1960.

[MMG$^+$06] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. In *Proc. 10th conference on Principles and Practice on Knowldedge Discovery in Databases (PKDD '06)*, volume 4213 of *Lecture Notes in Computer Science*, pages 335–346, 2006.

[MMG$^+$08] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering*, 20(10):1348–1362, 2008.

[MO04] Sara C. Madeira and Arlindo L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[MOP04] Adam Meyerson, Liadan O'Callaghan, and Serge Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56:61–87, 2004.

[MPR95] Sylvia D. Monson, Norman J. Pullman, and Rolf Rees. A survey of clique and biclique coverings and factorizations of $(0, 1)$-matrices. *Bulletin of the ICA*, 14:17–86, 1995.

[MRS04] Nina Mishra, Dana Ron, and Ram Swaminathan. A new conceptual clustering framework. *Machine Learning*, 56:115–151, 2004.

[MS84] Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

[OP83] Dianne P. O'Leary and Shmuel Peleg. Digital image compression by outer product expansion. *IEEE Transactions on Communications*, 31(3):441–444, 1983.

[PDL$^+$08] Peristera Paschou, Petros Drineas, Jamey Lewis, et al. Tracing sub-structure in the European American population with PCA-informative markers. *PLoS Genetics*, 4(7), 2008.

[Pel07] David Peleg. Approximation algorithms for the label-cover$_{\mathrm{MAX}}$ and red-blue set cover problems. *Journal of Discrete Algorithms*, 5(1):55–64, 2007.

[Por80] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization Algorithms and Complexity.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[PT94] Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5:111–126, 1994.

[RF01] Céline Robardet and Fabien Feschet. Efficient local search in conceptual clustering. In *Proc. Discovery Science*, volume 2226 of *Lecture Notes in Computer Science*, pages 323–335, 2001.

[RS97] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th annual ACM symposium on Theory of computing (STOC '97)*, pages 475–484, 1997.

[SBM03] Jouni K. Seppänen, Ella Bingham, and Heikki Mannila. A simple algorithm for topic identification in 0–1 data. In *Proc. Principles and Practice on Knowledge Discovery in Databases (PKDD '03)*, volume 2838 of *Lecture Notes in Computer Science*, pages 423–434, 2003.

[SG08] Ajit P. Singh and Geoffrey J. Gordon. A unified view of matrix factorization models. In *Proc. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD '08)*, volume 5212 of *Lecture Notes in Computer Science*, pages 358–373, 2008.

[Sim90] Hans Ulrich Simon. On approximate solutions for combinatorial optimization problems. *SIAM Journal on Discrete Mathematics*, 3(2):294–310, 1990.

[SKPH08] Václav Snášel, Pavel Krömer, Jan Platoš, and Dušan Húsek. On the implementation of Boolean matrix factorization. In *Proc. 19th International Conference on Database and Expert Systems Application*, pages 554–558, 2008.

[SPK$^+$08] Václav Snášel, Jan Platoš, Pavel Krömer, Dušan Húsek, Roman Neruda, and Alexander A. Frolov. Investigating Boolean matrix factorization. In *Proc. Workshop on Data Mining using Matrices and Tensors*, 2008.

[SSU03] Andrew I. Schein, Lawrence K. Saul, and Lyle H. Ungar. A generalized linear model for principal component analysis of binary data. In *Proc. 9th Internation Workshop on Artificial Intelligence and Statistics*, 2003.

[Ste93] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35(4):551–566, 1993.

[Ste99] G. W. Stewart. Four algorithms for the efficient computation of truncated pivoted QR approximations to a sparse matrix. *Numerische Mathematik*, 83:313–323, 1999.

[Ste05] G. W. Stewart. Error analysis of the quasi-Gram–Schmidt algorithm. *SIAM Journal on Matrix Analysis and Applications*, 27(2):493–506, 2005.

[SYL94] Ilkka Savijärvi and Eeva Yli-Luukko. *Jämsän äijän murrekirja*, volume 618 of *Suomalaisen Kirjallisuuden Seuran Toimituksia.* Suomalaisen kirjallisuuden seura, Helsinki, 1994.

[VAG07] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: Finding a minimal descriptive set of roles. In *Proc. 12th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 175–184, 2007.

[ZLDZ07] Zhongyuan Zhang, Tao Li, Chris Ding, and Xiangsun Zhang. Binary matrix factorization with applications. In *Proc. 7th IEEE International Conference on Data Mining (ICDM '07)*, pages 391–400, 2007.

APPENDIX A

Proof of Lemma 3.4

Recall that the claim of Lemma 3.4 was the following.

Lemma A.1 (Lemma 3.4). *Let* $g_c \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ *be defined by*

$$g_c(x) = 2^{(\log x)^{1-(\log\log x)^{-c}}}$$

for constant $c < 1/2$. *Then*

$$g_c\left(\frac{x}{O(g_{c'}(x)(\log\log x)^{c'})}\right) = \Omega\left(2^{\log^{1-\varepsilon} x}\right) \tag{A.1}$$

for any $c, c' < 1/2$ *and* $\varepsilon > 0$.

Before proceeding to the actual proof, let us consider some simple examples giving intuition why, at first place, the lemma might be true and whether it is a special property of just these functions or a consequence of a more general property. First important thing to note is that $g_c(x) = \Omega\left(2^{\log^{1-\varepsilon} x}\right)$. This is relatively easy to see by considering the logarithms of the functions:

$$\log(g_c(x)) = \log^{1-(\log\log x)^{-c}} x \tag{A.2a}$$

$$\log\left(2^{\log^{1-\varepsilon} x}\right) = \log^{1-\varepsilon} x. \tag{A.2b}$$

For x large enough, $1/(\log\log x)^c < \varepsilon$, and from that point on the exponent of (A.2a) is bigger than the exponent of (A.2b), and thus the former grows faster.

The problem, however, comes from the argument of g_c in (A.1): it is (essentially) $x/g_{c'}(x)$. We aim at proving a lower bound, and thus we hope a fast-enough growing algorithm. But at the same time, if g_c grows too fast, $x/g_{c'}(x)$ does not grow fast enough. Indeed, a superlinear polynomial $x \mapsto x^p$, $p > 1$, provides us a simple example of too-fast-growing algorithm, as $(x/x^p)^p = x^{p-p^2} \to 0$ as $x \to \infty$. On the other hand, a polylogarithm $x \mapsto \log^p x$ grows slow enough: $(\log(x/\log^p x))^p = (\log x - p \log\log x)^p = \Theta(\log^p x)$. As the function g_c is superpolylogarithmic and subpolynomial (see Section 3.3), neither of these observations furnishes us the desired result.

To start the actual proof, we shall first remove the big-O notation from the denominator of g_c's argument. The function g_c is an increasing function, and thus the smaller its argument, the smaller its value. To prove a lower bound for it we can safely underestimate its argument's value by replacing $O(g_{c'}(x)(\log\log x)^{c'})$ with $M_0 g_{1/2}(x)(\log\log x)^{1/2}$ for some constant M_0 and considering only large enough x. The constant M_0 comes from the definition of big-O, and it is easy to see that, for a fixed x, $g_{c'}(x)(\log\log x)^{c'}$ increases with c', attaining its maximum at $c' = 1/2$. Moreover, we shall omit M_0 for the sake of clearer presentation. Hence for our task, to prove (A.1), it is enough to prove that

$$\begin{aligned} &g_c\left(\frac{x}{g_{1/2}(x)(\log\log x)^{1/2}}\right) \\ &= 2^{\left(\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right)\right)^{1-\left(\log\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right)\right)^{-c}}} \end{aligned}$$

is bounded below by $2^{\log^{1-\varepsilon} x}$.

To simplify more, we can take logarithms of these functions, yielding claim

$$\begin{aligned} &\left(\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right)\right)^{1-\left(\log\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right)\right)^{-c}} \\ &= \Omega\left(\log^{1-\varepsilon} x\right). \end{aligned} \tag{A.3}$$

To bound the left-hand side of (A.3) from below, we shall find a lower bound for the base and for the exponent. Assume that $f_1(x)$ is an asymptotic lower bound for the base and $f_2(x)$ is an asymptotic lower bound for the exponent. Then there exists constants m_1 and m_2 such that the left-hand side of (A.3) is at least $(m_1 f_1(x))^{m_2 f_2(x)}$

for x large enough. Our goal is, of course, to find functions f_1 and f_2 that are easier to analyse than those in (A.3).

We start with the base $\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right)$. We have that

$$\begin{aligned}
&\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right)\\
&= \log x - \log(g_{1/2}(x)) - 1/2 \log\log\log x\\
&= \log x - (\log x)^{1-(\log\log x)^{-1/2}} - o(\log x).
\end{aligned}$$

To analyse the middle term's relation to $\log x$, we can notice that, although $1 - (\log\log x)^{-1/2} \to 1$ as $x \to \infty$, it is still less than 1 for all (finite) values of x, and thus $(\log x)^{1-(\log\log x)^{-1/2}}$ is less than $\log x$. This is formalized in the following lemma.

Lemma A.2. *We have*

$$\lim_{x\to\infty} \frac{\log x}{(\log x)^{1-(\log\log x)^{-1/2}}} = \infty.$$

Proof. The proof is by l'Hôpital's rule. The first derivative of the nominator is

$$D\log x = \frac{1}{x\ln 2}, \tag{A.4}$$

and that of the denominator is

$$\begin{aligned}
&D(\log x)^{1-(\log\log x)^{-1/2}} =\\
&\quad \left(\frac{\ln x}{\ln 2}\right)^{1-\left(\sqrt{\ln\left(\frac{\ln x}{\ln 2}\right)(\ln 2)^{-1}}\right)^{-1}} \left(1/2\ \ln\left(\frac{\ln x}{\ln 2}\right)\right.\\
&\quad \left(\ln\left(\frac{\ln x}{\ln 2}\right)(\ln 2)^{-1}\right)^{-3/2} x^{-1}\,(\ln x)^{-1}\,(\ln 2)^{-1}\\
&\qquad \left. + \left(1 - \left(\sqrt{\ln\left(\frac{\ln x}{\ln 2}\right)(\ln 2)^{-1}}\right)^{-1}\right) x^{-1}\,(\ln x)^{-1}\right)
\end{aligned}$$

which simplifies to

$$\begin{aligned}
&-1/2\ \left(\ln 2 - 2\sqrt{\ln 2}\sqrt{-\ln\ln 2 + \ln\ln x}\right)\\
&\quad (\ln 2)^{1/2\,\frac{-3\sqrt{\ln 2}\sqrt{-\ln\ln 2+\ln\ln x}+2\ \ln 2}{\sqrt{\ln 2}\sqrt{-\ln\ln 2+\ln\ln x}}}\,(\ln x)^{-\frac{\sqrt{\ln 2}}{\sqrt{-\ln\ln 2+\ln\ln x}}}\\
&\qquad x^{-1}\left(\sqrt{-\ln\ln 2 + \ln\ln x}\right)^{-1}
\end{aligned} \tag{A.5}$$

The $\ln 2$ (and $\ln\ln 2$) terms in (A.5) do not affect to our asymptotic considerations. Thus the first row of (A.5) simplifies to $O(\sqrt{\ln\ln x})$, the first factor of the second row to $\ln(2)^{1/\Theta(\sqrt{\ln\ln x})} = O(1)$, and the second factor of the second row to $\ln(x)^{-1/O(\sqrt{\ln\ln x})}$. The last row simplifies to $\left(x\,O(\sqrt{\ln\ln x})\right)^{-1}$.

Combining (A.4) and the asymptotic version of (A.5) gives

$$\begin{aligned}\frac{D\log x}{D(\log x)^{1-(\log\log x)^{-1/2}}} &= \frac{1}{x\ln 2}\left(\frac{O(\sqrt{\ln(\ln(x))})\,O(1)}{x(\ln x)^{1/O(\sqrt{\ln\ln x})}\,O(\sqrt{\ln\ln x})}\right)^{-1}\\ &= \frac{(\ln x)^{1/O(\sqrt{\ln\ln x})}\,O(\sqrt{\ln\ln x})}{O(\sqrt{\ln\ln x})\,O(1)}\\ &= \ln(x)^{1/O(\sqrt{\ln\ln x})}/O(1) \to \infty,\end{aligned} \tag{A.6}$$

as $x \to \infty$, and the claim follows from l'Hôpital's rule. $\square$

Thus we have that

$$\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right) = \log x - o(\log x) \geq m_1 \log x$$

for some constant m_1 and when x is big enough. Analysing the exponent goes similarly:

$$\begin{aligned}&1 - \left(\log\log\left(x/(g_{1/2}(x)(\log\log x)^{1/2})\right)\right)^{-c}\\ &= 1 - \left(\log\left(\log x - (\log x)^{1-(\log\log x)^{-1/2}} - 1/2\,\log\log\log x\right)\right)^{-c}\\ &= 1 - \left(\log\left(\log x - o(\log x)\right)\right)^{-c}\\ &\geq 1 - (\log(m_2\log x))^{-c}.\end{aligned}$$

The second equation follows from Lemma A.2, and the inequality from the fact that $\log x - o(\log x) = \Theta(\log x)$ [GKP94, eq. 9.20].

The final part of the proof is to show that

$$(m_1\log x)^{1-(\log(m_2\log x))^{-c}} = \Omega(\log^{1-\varepsilon} x). \tag{A.7}$$

But the left-hand side is, omitting m_2, $m_1 \log g_c(x)$, and even with m_2 in the exponent, we can follow our earlier reasoning by taking logarithms and noticing that, with x big enough, $(\log(m_2\log x))^{-c} < \varepsilon$, from which the claim follows. $\square$

APPENDIX B

Proof of Theorem 4.4

Recall Theorem 4.4.

Theorem 4.4. *There exists instances* $(\mathbf{A} \in \{0,1\}^{n\times m}, k)$ *of the* BMF *problem that cannot be approximated with an additive error of* $c(n,m) = \max\{\sqrt[4]{n}, \sqrt[4]{m}\}$.

We shall prove the theorem for case $n \leq m$; the other case is analogous. The proof resembles that of Theorem 4.3. Hence, for a contradiction, let us assume that $\mathfrak{A}$ is a polynomial-time algorithm for BMF such that for every $\mathbf{A} \in \{0,1\}^{n\times m}$, and for every $k < n$,

$$\mathsf{cost}_{\text{BMF}}(\mathbf{A}, \mathfrak{A}(\mathbf{A},k)) \leq \mathsf{cost}^*_{\text{BMF}}(\mathbf{A},k) + \sqrt[4]{m}. \tag{B.1}$$

Construct a new instance $(\mathbf{A}', k)$ by taking $f(c(m)) = f(\sqrt[4]{m})$ copies of $\mathbf{A}$, with f being a polynomial to be defined later. Thus we have an n-by-$f(\sqrt[4]{m})$ matrix

$$\mathbf{A}' = (\underbrace{\mathbf{A} \quad \mathbf{A} \quad \cdots \quad \mathbf{A}}_{f(\sqrt[4]{m})\text{ copies}}).$$

The optimal solution of $(\mathbf{A}', k)$ decomposes $\mathbf{A}$ optimally, and repeats that decomposition $f(\sqrt[4]{m})$ times, with

$$\mathsf{cost}^*(\mathbf{A}',k) = f(\sqrt[4]{m})\mathsf{cost}^*(\mathbf{A},k).$$

Matrix $\mathbf{A}'$ has $m' = f(\sqrt[4]{m})m$ columns, and thus we assume $\mathfrak{A}$'s results to admit bound

$$\begin{aligned}\mathsf{cost}(\mathbf{A}', \mathfrak{A}(\mathbf{A}',k)) &\leq f(\sqrt[4]{m})\mathsf{cost}^*(\mathbf{A},k) + c(m') \\ &= f(\sqrt[4]{m})\mathsf{cost}^*(\mathbf{A},k) + \sqrt[4]{f(\sqrt[4]{m})m}.\end{aligned} \tag{B.2}$$

To attain this bound, the average error caused by each copy of **A** must be at most

$$\left\lfloor \frac{f(\sqrt[4]{m})\mathsf{cost}^*(\mathbf{A},k) + \sqrt[4]{f(\sqrt[4]{m})m}}{f(\sqrt[4]{m})} \right\rfloor, \tag{B.3}$$

and there must be at least one copy of **A**, denoted by $\mathbf{A}_0$, for which the error indeed is below (B.3). But this means that the additional error allowed is at most

$$\left\lfloor \frac{\sqrt[4]{f(\sqrt[4]{m})m}}{f(\sqrt[4]{m})} \right\rfloor. \tag{B.4}$$

Selecting f such that (B.4) is 0 gives us a contradiction, as with Theorem 4.3. Letting $f(x) = x^2$ is one possibility, as

$$\begin{aligned}\frac{\sqrt[4]{f(\sqrt[4]{m})m}}{f(\sqrt[4]{m})} &= \frac{\sqrt[4]{(\sqrt[4]{m})^2 m}}{(\sqrt[4]{m})^2} \\ &= \frac{m^{3/8}}{m^{1/2}} \\ &= m^{-1/8} < 1,\end{aligned}$$

proving the claim. □

TIETOJENKÄSITTELYTIETEEN LAITOS	DEPARTMENT OF COMPUTER SCIENCE
PL 68 (Gustaf Hällströmin katu 2 b)	P.O. Box 68 (Gustaf Hällströmin katu 2 b)
00014 Helsingin yliopisto	FIN-00014 University of Helsinki, FINLAND
JULKAISUSARJA **A**	SERIES OF PUBLICATIONS **A**

Reports may be ordered from: Kumpula Science Library, P.O. Box 64, FIN-00014 University of Helsinki, FINLAND.

A-2001-1	J. Rousu: Efficient range partitioning in classification learning. 68+74 pp. (Ph.D. Thesis)
A-2001-2	M. Salmenkivi: Computational methods for intensity models. 145 pp. (Ph.D. Thesis)
A-2001-3	K. Fredriksson: Rotation invariant template matching. 138 pp. (Ph.D. Thesis)
A-2002-1	A.-P. Tuovinen: Object-oriented engineering of visual languages. 185 pp. (Ph.D. Thesis)
A-2002-2	V. Ollikainen: Simulation techniques for disease gene localization in isolated populations. 149+5 pp. (Ph.D. Thesis)
A-2002-3	J. Vilo: Discovery from biosequences. 149 pp. (Ph.D. Thesis)
A-2003-1	J. Lindström: Optimistic concurrency control methods for real-time database systems. 111 pp. (Ph.D. Thesis)
A-2003-2	H. Helin: Supporting nomadic agent-based applications in the FIPA agent architecture. 200+17 pp. (Ph.D. Thesis)
A-2003-3	S. Campadello: Middleware infrastructure for distributed mobile applications. 164 pp. (Ph.D. Thesis)
A-2003-4	J. Taina: Design and analysis of a distributed database architecture for IN/GSM data. 130 pp. (Ph.D. Thesis)
A-2003-5	J. Kurhila: Considering individual differences in computer-supported special and elementary education. 135 pp. (Ph.D. Thesis)
A-2003-6	V. Mäkinen: Parameterized approximate string matching and local-similarity-based point-pattern matching. 144 pp. (Ph.D. Thesis)
A-2003-7	M. Luukkainen: A process algebraic reduction strategy for automata theoretic verification of untimed and timed concurrent systems. 141 pp. (Ph.D. Thesis)
A-2003-8	J. Manner: Provision of quality of service in IP-based mobile access networks. 191 pp. (Ph.D. Thesis)
A-2004-1	M. Koivisto: Sum-product algorithms for the analysis of genetic risks. 155 pp. (Ph.D. Thesis)
A-2004-2	A. Gurtov: Efficient data transport in wireless overlay networks. 141 pp. (Ph.D. Thesis)
A-2004-3	K. Vasko: Computational methods and models for paleoecology. 176 pp. (Ph.D. Thesis)
A-2004-4	P. Sevon: Algorithms for Association-Based Gene Mapping. 101 pp. (Ph.D. Thesis)
A-2004-5	J. Viljamaa: Applying Formal Concept Analysis to Extract Framework Reuse Interface Specifications from Source Code. 206 pp. (Ph.D. Thesis)
A-2004-6	J. Ravantti: Computational Methods for Reconstructing Macromolecular Complexes from Cryo-Electron Microscopy Images. 100 pp. (Ph.D. Thesis)
A-2004-7	M. Kääriäinen: Learning Small Trees and Graphs that Generalize. 45+49 pp. (Ph.D. Thesis)
A-2004-8	T. Kivioja: Computational Tools for a Novel Transcriptional Profiling Method. 98 pp. (Ph.D. Thesis)

A-2004-9	H. Tamm: On Minimality and Size Reduction of One-Tape and Multitape Finite Automata. 80 pp. (Ph.D. Thesis)
A-2005-1	T. Mielikäinen: Summarization Techniques for Pattern Collections in Data Mining. 201 pp. (Ph.D. Thesis)
A-2005-2	A. Doucet: Advanced Document Description, a Sequential Approach. 161 pp. (Ph.D. Thesis)
A-2006-1	A. Viljamaa: Specifying Reuse Interfaces for Task-Oriented Framework Specialization. 285 pp. (Ph.D. Thesis)
A-2006-2	S. Tarkoma: Efficient Content-based Routing, Mobility-aware Topologies, and Temporal Subspace Matching. 198 pp. (Ph.D. Thesis)
A-2006-3	M. Lehtonen: Indexing Heterogeneous XML for Full-Text Search. 185+3 pp. (Ph.D. Thesis)
A-2006-4	A. Rantanen: Algorithms for ^{13}C Metabolic Flux Analysis. 92+73 pp. (Ph.D. Thesis)
A-2006-5	E. Terzi: Problems and Algorithms for Sequence Segmentations. 141 pp. (Ph.D. Thesis)
A-2007-1	P. Sarolahti: TCP Performance in Heterogeneous Wireless Networks. (Ph.D. Thesis)
A-2007-2	M. Raento: Exploring privacy for ubiquitous computing: Tools, methods and experiments. (Ph.D. Thesis)
A-2007-3	L. Aunimo: Methods for Answer Extraction in Textual Question Answering. 127+18 pp. (Ph.D. Thesis)
A-2007-4	T. Roos: Statistical and Information-Theoretic Methods for Data Analysis. 82+75 pp. (Ph.D. Thesis)
A-2007-5	S. Leggio: A Decentralized Session Management Framework for Heterogeneous Ad-Hoc and Fixed Networks. 230 pp. (Ph.D. Thesis)
A-2007-6	O. Riva: Middleware for Mobile Sensing Applications in Urban Environments. 195 pp. (Ph.D. Thesis)
A-2007-7	K. Palin: Computational Methods for Locating and Analyzing Conserved Gene Regulatory DNA Elements. 130 pp. (Ph.D. Thesis)
A-2008-1	I. Autio: Modeling Efficient Classification as a Process of Confidence Assessment and Delegation. 212 pp. (Ph.D. Thesis)
A-2008-2	J. Kangasharju: XML Messaging for Mobile Devices. 24+255 pp. (Ph.D. Thesis).
A-2008-3	N. Haiminen: Mining Sequential Data – in Search of Segmental Structures. 60+78 pp. (Ph.D. Thesis)
A-2008-4	J. Korhonen: IP Mobility in Wireless Operator Networks. (Ph.D. Thesis)
A-2008-5	J.T. Lindgren: Learning nonlinear visual processing from natural images. 100+64 pp. (Ph.D. Thesis)
A-2009-1	K. Hätönen: Data mining for telecommunications network log analysis. 153 pp. (Ph.D. Thesis)
A-2009-2	T. Silander: The Most Probable Bayesian Network and Beyond. (Ph.D. Thesis)
A-2009-3	K. Laasonen: Mining Cell Transition Data. 148 pp. (Ph.D. Thesis)
A-2009-4	P. Miettinen: Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms. 164+6 pp. (Ph.D. Thesis)