Mining Representative Items and Itemsets with Binary Matrix Factorization and Instance Selection

Seyed Hamid Mirisaee

UNIVERSITÉ GRENOBLE ALPES

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 Aôut, 2006

Présentée par

Seyed Hamid Mirisaee

Thèse dirigée par **Eric Gaussier**

et codirigée par **Alexandre Termier**

préparée au sein du **Laboratoire d'Informatique de Grenoble**

et de **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Fouille d'Items et d'Itemsets Représentatifs avec des Méthodes de Décomposition de Matrices Binaires et de Sélection d'Instances

Mining Representative Items and Itemsets with Binary Matrix Factorization and Instance Selection

Thèse soutenue publiquement le **16 Septembre, 2015**,

devant le jury composé de :

Pascal Poncelet

Professeur, Université de Montpellier, Rapporteur

Céline Robardet

Professeur, INSA de Lyon, Rapporteur

Arno Siebes

Professeur, Universiteit Utrecht, Examinateur

Christel Vrain

Professeur, Université d'Orléans, Président

Eric Gaussier

Professeur, Université Joseph Fourier, Directeur de thèse

Alexandre Termier

Professeur, Université de Rennes 1, Co-Directeur de thèse

Abstract

This thesis focuses on mining representative items and itemsets using Binary Matrix Factorization (BMF) and instance selection. To accomplish this task, we first, in Chapter 1, consider the BMF problem by studying the literature on matrix decomposition techniques and the state-of-the-art algorithms. Then, we establish a connection between BMF problem and Unconstrained Binary Quadratic Programming (UBQP) problem in order to use UBQP's algorithms and heuristics, available in the literature, in case of BMF solutions. Next, in Chapter 2, we propose a new, efficient heuristic which flips 1 bit at the time in order to improve the solutions of BMF. Using the established link discussed in Chapter 2, which enables us to use heuristics of UBQP, we compare the proposed technique, called 1-opt-BMF with that of UBQP, called 1opt-UBQP as well as the standard approach, called 1-opt-Standard. We then show, theoretically and experimentally, the efficiency of 1-opt-BMF on a wide range of publicly available datasets. Next, in Chapter 3, we explore addressing the problem of finding representative itemsets via BMF. To do that, we first consider the theoretical relation between the frequent itemset mining problem and BMF; while established, we propose a new technique called Decomposition Itemset Miner (DIM). We then design a set of experiments to show the efficiency of DIM and the quality of its results.

Finally, in Chapter 4, we consider the problem of finding representative objects (instances) in big, high-dimensional datasets. These objects helps us to find objects providing a global, top-view of the data and are very important in data analysis process. We first study the available methods for finding representative objects and discuss the pros and cons of each. We then formally define the Instance Selection Problem (ISP), provide three variants of that and examine their complexities before providing their solutions. In the experimental section, we show that although the ISP solutions can outperform other methods in some cases, in general it should be considered as a complementary technique in the context of finding representative objects.

Résumé

Dans cette thèse, nous nous intéressons à la recherche d'"items" et d'"itemsets" d'intérêt via la décomposition de matrice binaire (Binary Matrix Factorization, BMF) et à la recherche d'objets représentatifs. Pour cela, nous étudions l'état de l'art des techniques de décomposition matricielle. Nous établissons, dans le premier Chapitre, un lien entre BMF et le problème de programmation binaire quadratique sans contraintes (Unconstrained Binary Quadratic Programming, UBQP) afin d'utiliser les algorithmes et heuristiques existant dans la littérature pour UBQP et les appliquer à BMF. Nous proposons dans le Chapitre 2 une nouvelle heuristique adaptée au calcul de BMF. Cette technique efficace optimise les solutions de BMF ligne par ligne (ou colonne par colonne) en inversant 1 bit à chaque fois. En utilisant le lien établi dans le Chapitre 2 qui nous permet d'appliquer les algorithmes et heuristiques d'UBQP à BMF, nous comparons la méthode proposée (1-opt-BMF) avec les heuristiques spécialisées pour UBQP (1-opt-UBQP) ainsi que les heuristiques classiques (1-opt-Standard). Nous montrons ensuite, en théorie et en pratique, l'efficacité de 1-opt-BMF sur une large variété de données publiques. Dans le Chapitre 3, nous nous intéressons au problème de la recherche des itemsets représentatifs en utilisant BMF et 1-opt-BMF. Pour cela, nous considérons dans un premier temps le lien entre le problème de "frequent itemset mining" et BMF, et proposons une nouvelle méthode que nous appelons "Decomposition Itemset Miner" (DIM). Une série d'expérience montre la qualité des résultats obtenus et l'efficacité de notre méthode.

Enfinf, nous nous intéressons, dans le Chapitre 4, à la recherche d'objets représentatifs (qui donnent une vue globale sur les données) dans des données de grandes dimensions. Nous examinons les méthodes disponibles dans la littérature en donnant les avantages et les inconvénients de chacune. Ensuite, nous défnissons mathématiquement le problème de sélection d'instance (Instance Selection Problem: ISP) et présentons trois variantes à ce problème ainsi que leur solutions. Dans les expériences, nous montrons que, bien qu'ISP puisse surpasser les autres méthodes dans certains cas, il vaut mieux le considérer en général comme une technique complémentaire dans le cadre de la recherche des objets représentatifs.

Acknowledgments

Firstly, I would like to thank my supervisor, Eric Gaussier, for his continuous support and patience during the period my PhD studies. Without his motivation and immense knowledge, it would have been certainly impossible to conduct this research.

I also would like to express my sincere gratitude to my co-supervisor, Alexandre Termier, whose encouragement and insightful comments helped me in every single step of this research, particularly in the process of writing the papers as well as this thesis.

Besides, I would like to thank all jury members for their interesting comments and questions during the defense session. Additionally, I would like to appreciate the reports of the reviewers of this thesis which helped me to improve the quality of this manuscript.

I must also acknowledge Francine Chen, from FXPAL, who offered me a summer internship in Fuji Xerox Palo Alto, where I could broaden my knowledge an expertise by attending many brilliant talks and participating in interesting projects.

Appreciation also goes out to all members of the AMA team in LIG Lab. with whom I had a wonderful time during my PhD studies.

I also would like to thank the French Ministry of Higher Education which financially supported my PhD studies.

Finally, I would like to thank my family, specifically my mother, for the support they provided me through my entire life.

تقدیم به مهربانترینم

CONTENTS

List of Figures

List of Tables

INTRODUCTION

As computer hardware is getting cheaper and cheaper, and the computers get faster and faster, data collection becomes easier and easier. However, as this process is done easily, the data analysis task became a challenging problem. Due to the fast growth of social networks over the past decade, one can simply obtain a large set of data in a short amount of time. This data, nevertheless, is tremendously large and potentially noisy. As a result, new methods for analyzing large sets became a hot topic over the past few years.

Data mining is the process of extracting information out of a database or a data warehouse [Witten and Frank, 2005]. It is a wide domain which covers a large number of techniques (from neural networks and statistics to high-performance computing and data visualization) in order to find useful information from datasets. As, in general, the datasets considered in data mining applications are large and high-dimensional, (and as a result are hard to understand and analyze) data simplification is one of the most important methods used while dealing with large data.

For instance, consider the case of social network datasets such as Twitter. The tweets are short messages (up to 140 characters) which are posted by Twitter users. Tweets have attracted a lot of attentions recently as they could provide a wide range of interesting information in many areas such as topic detection, sentiment analysis and studies on modern society [Pang and Lee, 2008, Bittner, 1973, Cataldi et al., 2010, Pak and Paroubek, 2010, Chen and Mirisaee, 2014, Junco et al., 2013]. Consider the problem of sentiment analysis in Twitter where one is interested in assigning a label to each tweet: positive, negative and neutral. A positive tweet is a tweet which has a positive polarity. For example "*My new iphone is great.*" is a positive tweet. Sentiment analysis can be used in many areas such as marketing and product evaluation and feedback. As one can guess, the content of a tweet could vary: from simple daily status to a worldwide disaster. They could be in any language (even mixture of several languages), could contain typos and may have emoticons. One can simply collect a large

amount of tweets for some social network analysis task. However, according to the above-mentioned issues, directly applying data mining techniques on such dataset will be probably very time consuming (since initially it is high-dimensional) and produce poor results (since it is extremely noisy). Accordingly, one can apply a data simplification technique in order to address (some of if not all of) these problems [Zhu et al., 2014].

In data mining, the data is usually represented using matrices where rows represents the objects and columns represent the features. Following the example discussed above on Twitter, rows (objects/instances) can be the tweets and columns (features/attributes) can be the terms of the tweet. Many techniques have been studied to simplify such data. Each of these techniques target certain characteristics of the data in order to make it easier to analyze. For instance, some focus on number of objects and try to reduce it in a smart way. Others work on the number of attributes and try to reduce it without loosing too much information. As the data is mostly represented as matrices, *matrix decomposition* or *matrix factorization* is one of the most popular methods in this regard. It has shown a significant potential in a wide range of fields and attracts more and more attention as time goes by.

In general, matrix decomposition is a process which enables us to break a large matrix into two or more smaller matrices such that the multiplication of these matrices approximates the original data. As mentioned before, data could be very large, high-dimensional and noisy. As a result, the main idea of matrix factorization techniques is to simplify the data and make it more understandable. These methods try to find the *main* and *latent* information of the data. Accordingly, they can simply detect the noise (as long as the noise is not a dominant part of the data, it is not considered as the main information and is filtered out through the approximation process) and provide us with important and useful information.

As an example, Principal Component Analysis (PCA) [Pearson, 1901, Borgne and Bontempi, 2007] is a very well known decomposition technique. To better understand the importance of a decomposition techniques like PCA, let us consider the example of a high-dimensional dataset were the features are not only numerous but also redundant and possibly correlated. To handle such cases, one may use feature selection algorithms which try to filter the irrelevant or redundant features in order to find more precise information from data in a shorter amount of time [Varmuza, 1980]. PCA is one of these techniques which groups some features together (as a linear combination for example). More precisely, it is a transformation from original attributes to a set

of linearly uncorrelated variables. Invented by Pearson in 1901, PCA is one of the pioneers of data reduction field and has shown the importance of the problem since then.

A closely related method for data simplification is Singular Value Decomposition (SVD) which has been widely used in many contexts such as biological systems, image coding, signal processing, speech recognition and noise reduction [Golub and van Loan, 1996, DeAngelis et al., 1995, Andrews and Patterson III, 1976, Dendrinos et al., 1991]. Like PCA, SVD could also be seen as a technique to transform a set of correlated variables to a set of uncorrelated ones. It also could be seen as a method which can find the attributes along which most variations are observed. This latter point of view provides us with a technique to best approximate the data with fewer dimensions.

So far, we have seen that the analysis of big data is a challenging task and a number of approaches have been developed to deal with this issue. However, we did not discuss the data format yet. Intuitively, when we use a data simplification technique (such as matrix decomposition techniques), we would like to have the simplified data (*i.e.* approximated data) in the same form as the input data. For example, if the input is a matrix of real numbers, we are interested in approximating such input with a real-valued matrix, or if it is non-negative, we need the approximation to be also non-negative. As many datasets are real-world observations, they usually contain non-negative values. Consider the example of a document-term matrix where each line represents a document and each column represents a term; each cell in such a matrix represents the number of occurrences of the corresponding term in the corresponding document. Clearly this data is a non-negative, integer-valued matrix. Another example is the transaction-item matrix in which rows represent transactions (a ticket of a customer in a supermarket for example) and columns represent items (the products available in this supermarket). Then each cell shows if the corresponding items has been bought in the corresponding customer visit or not. In this case also we have a non-negative real-valued matrix as an input of our mining algorithm. Accordingly a new trend of research has been started for matrices satisfying these conditions.

Non-negative Matrix factorization (NMF) [Lee and Seung, 1999a, Lee and Seung, 2001] is a very well known method which tries to tackle the aforementioned cases. This technique gets a non-negative matrix as an input and provides two (and in some formulations several) non-negative matrices whose product approximates the input. This approach was first used to learn the parts of faces and semantic features [Lee and Seung, 1999a] and then became quite popular in data mining, data analysis and

many other communities. Two important points of the NMF method is that it induces sparsity (which is always suitable for data analysis) and leads to a part-based decomposition (which also facilitates the interpretation process). NMF has been successfully applied to many applications such as object and face recognition, denoising, hashing and texture classification [Soukup and Bajla, 2008, Mairal et al., 2010, Monga and Mıhçak, 2007, Sandler and Lindenbaum, 2011].

In some applications, the input data is not a real-valued matrix, neither an integer-valued one, but it is simply a binary matrix, *i.e.* it simply contains zeros and ones. For example, one can simply consider a tweet-term matrix as a binary matrix. The main reason is that tweets are very short texts and one can simply ignore the repetition of terms in a tweet, which is very unlikely if we remove the stop words (very frequent non-informative words). Another example is a frequent itemset mining dataset where, as explained before, lines represent transactions and columns represent items (products). In this case, we also can consider the input as a binary matrix since the number of products is not always of interest. What is interesting is the items purchased together frequently and not the quantity of each item. Protein-protein interactions and discrete pattern mining are other examples where the input data is a binary matrix [Tu et al., 2011, Shen et al., 2009].

Considering the above-mentioned issues, in this thesis, we study the use of data simplification techniques, in particular Binary Matrix Decomposition (BMF), in order to find representative itemsets in frequent itemset mining problem. Accordingly, in Chapter 1, we study the state-of-the-art and examine different decomposition techniques. We then formalize the BMF problem and, as our first contribution, find a link between the BMF problem and the Unconstrained Binary Quadratic Program (UBQP) problem [Mirisaee et al., 2015c]. Then in Chapter 2, we use this established link in order to use the heuristics of UBQP to improve the BMF solutions. Afterwards, we provide an efficient heuristic, as our second contribution, for BMF and show that it performs efficiently with respect to other techniques [Mirisaee et al., 2015b, Mirisaee et al., 2015c]. In Chapter 3, we discuss our third contribution and provide a theoretical link between BMF and frequent itemset mining problem. We illustrate that, using this link, one can find very few, high-quality representative itemsets [Mirisaee et al., 2014]. Finally, in Chapter 4, we try to find representative objects (instances) from large, high-dimensional data using different techniques and compare the results with the state-of-the-art methods. There, as our fourth contribution, we show that the proposed technique can be considered as an alternative, complementary method to

the existing ones [Mirisaee et al., 2015a].

I Binary Matrix Factorization for Data Mining Purposes

1 General Considerations on Binary Matrix Factorization

1.1 Introduction

Many datasets of interest for scientific or industrial applications are high dimensional binary matrices. This high dimensionality negatively impacts the performance of traditional data analysis algorithms such as clustering or association rule mining. Additionally, they could suffer from a wide range of overlap and noise which can make some difficulties for the data mining tasks, such as expensive computations or large amount of mostly redundant results. Although many feature selection methods could be applied in order to reduce the data dimension, in most cases the correlation among the features is quite complex and local [Zhang et al., 2007]. Accordingly, these approaches may not be efficient in many applications. In such cases, one can use matrix factorization techniques.

Matrix factorization is a way to compress the data while preserving most of the characteristic patterns found inside. Matrix decomposition is a technique to find latent information, remove the noise and simplify the data. In general, matrix factorization is the process of decomposing a matrix into two or more matrices, called factors, such that the multiplication of these factors approximates the original matrix.

Over the past years, different matrix decomposition techniques have been successfully used in many applications such as image processing and clustering [Lee and Seung, 1999b, Zhang and Li, 2010, Zhang et al., 2005, Drineas et al., 2004]. Depending on the application one may use different decomposition techniques. For instance, Singular Value Decomposition (SVD) [Golub and van Loan, 1996], which is probably the most known method in this context, finds the singular values of a given matrix as well as its right and left singular vectors. This technique has many applications such as

image retrieval [Praks et al., 2003] and noise reduction [Sadasivan and Dutt, 1996]. The main idea in SVD is to map highly variable data points to a lower dimensional space. In general, given a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, SVD is formulated as follows:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices and they contain eigenvectors of $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}^T X$, respectively. $\Sigma \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing singular values of $\mathbf{X}$ in descending order. Computing an SVD decomposition might be expensive as it has a time complexity of $O(min\{mn^2, m^2n\})$ which is very time consuming for large matrices [Holmes et al., 2007].

PCA, which is very similar to SVD, is considered as another popular method of matrix decomposition [Borgne and Bontempi, 2007, Jolliffe, 2002]. In particular, it is used to deal with datasets having a very large number of attributes among which there exits a large number of correlated attributes. Consequently, it is regarded as a proper method of dimension reduction. PCA first makes the input zero-centered, then computes the eigenvectors of the covariance matrix of the attributes. Finally, according to the desired number of attributes, it takes the eigenvectors corresponding to the eigenvalues sorted in decreasing order. The time complexity of PCA depends on the computation of eigenvalues and eigenvectors of the covariance matrix. Using the Jacobi's method described in [Golub and van Loan, 1996] the time complexity of applying PCA on a matrix of size $m \times n$ is $O(n^3)$.

Non-negative Matrix Factorization (NMF) [Lee and Seung, 1999a, Lee and Seung, 2001] is another popular method in which the input as well as the decomposition factors need to be binary. Although the initial idea of NMF, which was named Positive Matrix Factorization (PMF) at the time, has been proposed by Paatero et. al. in [Paatero and Tapper, 1994a], Lee and Seung were the first to introduce what we know today as NMF. They first used that to learn the parts of faces and semantic features [Lee and Seung, 1999a]; however, other researchers have found many applications for this technique, such as sound source separation [Virtanen, 2007], recommender systems [Koren et al., 2009a] and spectral data analysis [Pauca et al., 2006]. After the original NMF method, some other approaches have been considered for update rules. The projected gradient, studied in [Lin, 2007], is an example of such methods. The

general formulation of the NMF problem is expressed as the following:

$$\underset{\mathbf{W},\mathbf{H}}{\operatorname{argmin}} ||\mathbf{X} - \mathbf{W} \times \mathbf{H}||_2$$
$$\text{subject to} \quad \mathbf{X}, \mathbf{W}, \mathbf{H} \geq 0$$

Different methods have been proposed to solve the above problem. One approach, called Alternative Least Squares (ALS), has been proposed in [Paatero and Tapper, 1994b] and consists of solving the following equations for $\mathbf{H}$ and $\mathbf{W}$ respectively:

$$\mathbf{W}^T\mathbf{W}\mathbf{H} = \mathbf{W}^T\mathbf{X}$$
$$\mathbf{H}\mathbf{H}^T\mathbf{W}^T = \mathbf{H}\mathbf{X}^T$$

and then simply project all the negative values to zero. One very important advantage of this approach is that zero elements are not locked (unlike the multiplicative update which is explained shortly). A wide range of improvements have been made on ALS algorithm such as [Langville et al., 2014] and [Paatero, 1999].

Another method to solve the NMF problem is introduced by Lee and Seung in [Lee and Seung, 2001] and called multiplicative updates. This method starts from $\mathbf{W}^1 \geq 0$, $\mathbf{H}^1 \geq 0$ (the superscripts denotes the iteration number), and then, iteratively, updates them using the following rules until the stopping condition is satisfied:

$$\mathbf{W}^{k+1} = \mathbf{W}^k \frac{(\mathbf{X}(\mathbf{H}^k)^T)}{(\mathbf{W}^k\mathbf{H}^k(\mathbf{H}^k)^T)} \tag{1.1}$$

$$\mathbf{H}^{k+1} = \mathbf{H}^k \frac{((\mathbf{W}^{k+1})^T\mathbf{X})}{((\mathbf{W}^{k+1})^T\mathbf{W}^{k+1}\mathbf{H}^k)} \tag{1.2}$$

Multiplicative update approach is sensitive to initial values. Another disadvantage of this algorithm is that zero elements are locked. In other terms, if an element of $\mathbf{W}$ or $\mathbf{H}$ becomes zero in one iteration, it can never become positive again. Another problem of this approach is that, sometimes, the denominators of Eq. (1.1) and Eq. (1.2) could be zero. To handle that, some studies, such as [Piper et al., 2004], suggests to add a small value to the denominators.

An alternative approach to solve the NMF problem is called alternating least squares using projected gradient. This method tries to minimize the cost function using gradient descend together with projection [Lin, 2005]. The projection is a necessary step in this type of bound-constrained optimization. In this problem, we need to minimize the cost function $f(x)$ subject to $\forall i,\ l_i \leq x_i \leq u_i$, where $f(x) : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function and l and u are lower and upper bounds, respectively.

Assuming that k is the iteration index, projected gradient approach for two consequent solutions is defined as follows [Lin, 2005]:

$$x^{k+1} = P[x^k - \alpha^k \bigtriangledown f(x^k)] \tag{1.3}$$

where

$$P[x] = \begin{cases} x_i & \text{if } l_i < x_i < u_i \\ u_i & \text{if } x_i \geq u_i \\ l_i & \text{if } x_i \leq l_i \end{cases} \tag{1.4}$$

In projected gradient approach, it is necessary to ensure that we obtain sufficient decrease of the cost function in each iteration. In other words:

$$f(x^{k+1}) - f(x^k) \leq \sigma \bigtriangledown f(x^k)^T (x^{k+1} - x^k) \tag{1.5}$$

where σ is a small positive value. As mentioned in [Lin, 2005], the condition (1.5) is used in most projected gradient methods. Bertsekas in [Bertsekas, 1976] showed that using $\alpha^k > 0$ satisfying (1.5) always exists, and every limit point of x^k for $k = 1, 2, ...$ is a stationary point of $f(x)$. Based on the alternative behavior of this approach (rotation between fixing W and fixing H according to the terminology used in [Koren et al., 2009b]), we can speed up the process using the matrix-based operations. However, there is still one time consuming step which is searching a good step size α^k in each iteration [Lin, 2005].

So far we discussed many different decomposition techniques. However, in many applications of data mining, the input matrix is represented as a binary matrix, *i.e.* it represents, for each object, the presence or the absence of features. As mentioned before, frequent itemset mining and tweet-term matrices are examples of such case. In these cases, what is taken into account is the presence or absence of a given attribute in an instance. These dataset could also be very large, millions of tweets is commonplace for example. Accordingly, a new body of research has been dedicated to this subject: Binary Matrix Factorization.

When the input matrix and the factors are both binary matrices, the operation is called a Binary Matrix Factorization (BMF). The binary factors preserve the property of the input data, as they are binary, and impose sparsity. BMF has been widely used in many data mining tasks such as gene expression analysis [Zhang et al., 2007, Zhang et al., 2010], digits reconstruction [Meedsa et al., 2006], document clustering [Li, 2005, Zhang et al., 2007] and frequent itemset mining [Mirisaee et al., 2014]. To better understand the importance of the BMF process, consider the following example

from frequent itemset mining domain where we are looking for rectangles of ones in a given binary matrix:

Example 1.1

$$\mathbf{X} = \begin{pmatrix} \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} \\ \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & 0 & \boldsymbol{1} & 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 \end{pmatrix}$$

As one can see, three main blocks of ones (shown in boldface) exist in such matrix while one of them (the one in the middle) contains a hole; a zero value which could be noise or a missing value (shown in red). However, for an analyst, this block of one is still considered as one single pattern. For him/her the lonely one in the first row (also shown in red) is *noise* or an *unimportant* value. Having this matrix, if we try to find the patterns with a classical approach this block will be broken to several patterns. An unsatisfying solution over the block of ones in the middle could look like the following:

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

As one can see, the "noisy" block is broken to two sub-blocks (shown in red and blue). Now, we consider a BMF approach to analyze the patterns. Consider the following results from a BMF decomposition:

Example 1.2

$$\mathbf{W} = \begin{pmatrix} \boldsymbol{1} & 0 & 0 \\ \boldsymbol{1} & 0 & \boldsymbol{1} \\ \boldsymbol{1} & 0 & \boldsymbol{1} \\ 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & \boldsymbol{1} & 0 \\ 0 & \boldsymbol{1} & 0 \end{pmatrix} \quad \mathbf{H} = \begin{pmatrix} \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} \end{pmatrix}$$

This gives $\mathbf{X}' = \mathbf{W} \times \mathbf{H}$ *as an approximation of* $\mathbf{X}$*:*

$$\mathbf{X}' = \begin{pmatrix} \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} \\ \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & \boldsymbol{1} & \boldsymbol{1} \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & 0 & 0 & 0 \end{pmatrix}$$

which, first of all, flipped the zero on line 5 column 6 from zero to one and makes the block of ones in the middle as a perfect pattern. Secondly, it could flip the one in row 1 column 8 (also shown in red in previous example) to zero which was an uninteresting pattern. As this example shows, the BMF technique can be used in many cases to retrieve important information from a dataset.

Given $\mathbf{X} \in \{0,1\}^{M \times N}$ and $K \in \mathbb{N},\ K << \min(M,N)$, the general problem of rank K binary matrix factorization is of the following form:

$$\begin{cases} \underset{\mathbf{W},\mathbf{H}}{\text{argmin}} & ||\mathbf{X} - \mathbf{W} \times \mathbf{H}||_p \ (p = 1 \text{ or } p = 2) \\ \text{subject to} & \mathbf{W} \in \{0,1\}^{M \times K}, \mathbf{H} \in \{0,1\}^{K \times N} \\ & \text{(optional) } \mathbf{W} \times \mathbf{H} \in \{0,1\}^{M \times N} \end{cases} \tag{1.6}$$

where $||.||_p$ denotes either the L_1-norm ($p = 1$) or the L_2-norm ($p = 2$). The optional constraint is here referred to as the *binary reconstruction constraint*.

Different methods have been proposed to solve (more precisely, to provide an approximation of the solution of) the above problem. For the L_2-norm, efficient approaches usually solve a relaxed version of the problem, e.g. through non-negative

Algorithm 1 Grid search for NMF

Input: $\mathbf{X} \geq 0$, step size α

Output: $W_{bin}, H_{bin} \in \{0, 1\}$

Apply NMF on $\mathbf{X}$ to obtain $W, H \geq 0$
$\epsilon = \infty$
$min_W = min \sum_{i=1}^{m} \sum_{j=1}^{k} W_{ij}$ and $max_W = max \sum_{i=1}^{m} \sum_{j=1}^{k} W_{ij}$
$min_H = min \sum_{i=1}^{k} \sum_{j=1}^{n} H_{ij}$ and $max_H = max \sum_{i=1}^{k} \sum_{j=1}^{n} W_{ij}$
$W' = W$, $H' = H$
while $min_W \leq max_W$ **do**
    while $min_H \leq max_H$ **do**
        IF $\mathbf{W}'_{ij} < min_W$ THEN $\mathbf{W}'_{ij} = 0$ ELSE $\mathbf{W}'_{ij} = 1$
        IF $\mathbf{H}'_{ij} < min_H$ THEN $\mathbf{H}'_{ij} = 0$ ELSE $\mathbf{H}'_{ij} = 1$
        IF $||\mathbf{X} - \mathbf{W}' \times \mathbf{H}'|| < \epsilon$ THEN update ϵ, W_{bin} and H_{bin}
        $min_H = min_H + \alpha$
        $W' = W$, $H' = H$
    end while
    $min_W = min_W + \alpha$
end while
return W_{bin} and H_{bin}

matrix factorization (NMF) or singular value decomposition (SVD), and then project the solution onto the admissible domain. This can be done through a simple thresholding or a grid search. The former is to simply project the values above a certain threshold θ to one and the values below that to zero. The grid search, on the other hand, tries different thresholds for each factor and returns the best result as the output. Algorithm 1 illustrates the grid search process for NMF (SVD is treated in the same way).

Alternatively, one can iteratively solve simpler problems with $K = 1$, and then aggregate the solutions to obtain the solution to the original problem. This is typically the approach adopted in the *Proximus* system [Koyuturk and Grama, 2003, Koyuturk et al., 2005]. This method will be explained in detail in the next section.

In both NMF-based approaches and *Proximus*, the approximate solution is found by iteratively fixing one matrix, $\mathbf{W}$ or $\mathbf{H}$, and solving for the other. For the L_1-norm, the most efficient approaches, to the best of our knowledge, consider the problem as a clustering problem and make use of a variant of the K-means algorithm to solve

it [Jiang et al., 2014].

Previous studies have shown that computing a rank 1 ($K = 1$) approximation could be reformulated as a 0-1 integer linear programming problem [Shen et al., 2009]. They exploit this reformulation to guarantee the computation of an optimal solution on small matrices and, through relaxations, they provide approximations with a bounded error rate on larger matrices.

In the following sections, we first review some related studies in the matrix decomposition domain by first considering the standard methods (explained briefly in this section) and then reviewing the variants as well as the methods dedicated to the *binary* case. We also mention, very briefly, a totally different trend of decomposition, called *boolean matrix decomposition* at the end of related work section. Next, we consider a known problem called Unconstrained Binary Quadratic Programming (UBQP) problem and establish a link between this problem and binary matrix factorization as our first contribution [Mirisaee et al., 2015b, Mirisaee et al., 2015c]. Eventually the last section provides some closing remarks on the materials covered in the chapter.

1.2 Related Work

1.2.1 Standard methods

The optimization problem (1.6) can be seen as a non-negative matrix factorization (NMF) [Lee and Seung, 1999a, Lee and Seung, 2001] problem with additional binary constraints. Two main approaches have been followed along this line. The first one (used in [Miettinen et al., 2008]) amounts to first solve a standard NMF problem and then to project the solution onto the $\{0,1\}^{M\times K} \times \{0,1\}^{K\times N}$ sub-space. The projection step amounts to setting to 1 all values of $\mathbf{W}$ (resp. $\mathbf{H}$) above a threshold θ_w (resp. θ_h) and to 0 all the other ones. θ_w and θ_h are either learned from the data (typically using a grid search) or set to a predefined value, as 0.5. Learning the thresholds, even though more costly, usually leads to better results [Miettinen et al., 2008]. The grid search has been discussed previously and illustrated in Algorithm 1.

The second approach (used in [Zhang et al., 2007] and [Zhang et al., 2010] for example) uses the regularization terms to integrate the binary constraints into the

objective function being minimized in NMF, leading to:

$$\begin{cases} \underset{\mathbf{W},\mathbf{H}}{\text{argmin}} & ||\mathbf{X}-\mathbf{W}\times\mathbf{H}||_2 \\ & +\lambda_1||\mathbf{W}^{(2)}-\mathbf{W}||_2+\lambda_2||\mathbf{H}^{(2)}-\mathbf{H}||_2 \\ \text{subject to} & \mathbf{W}\geq 0, \mathbf{H}\geq 0 \end{cases} \tag{1.7}$$

where $\mathbf{W}^{(2)}_{ij}=(\mathbf{W}_{ij})^2$ (and similarly for $\mathbf{H}^2$). Hence, the second and third terms of the objective function force the factors to be binary, as $||\mathbf{W}^{(2)}-\mathbf{W}||_F$ (resp. $||\mathbf{H}^{(2)}-\mathbf{H}||_F$) is null for binary matrices and strictly positive otherwise. However, this approach does not in general yield binary factors; setting λ_1 and λ_2 to very high values may result in binary factors, but at the expense of having a large reconstruction error. It is nevertheless possible to couple this approach with the preceding one, as the factors obtained prior to thresholding will be closer to binary values than the ones obtained through standard NMF. The same approach can be used for SVD, dropping in this case the positive constraints in the optimization problem (as done in [Miettinen, 2008a] for example, for a similar problem on boolean, and not binary, matrix decomposition).

A problem similar to (1.6) is formulated in [Li, 2005]. In this latter work, however, a third, non-binary matrix is introduced making the problem addressed different from the one we are interested in. This said, the approach developed in [Li, 2005] bears strong similarities with the ones used for solving Problem (1.6). The formulation provided in this study, to cluster binary data, is the following:

$$\mathbf{W}=\mathbf{AXB}^T+\mathbf{E} \tag{1.8}$$

where $\mathbf{A}$ and $\mathbf{B}$ represent the membership values for clusters and $\mathbf{X}$ denotes the cluster representation. $\mathbf{E}$ is the error component this formulation. In this study, also, the authors provide an iterative approach to solve the clustering problem: given $\mathbf{X}$ and $\mathbf{B}$, we can optimize $\mathbf{A}$; given $\mathbf{X}$ and $\mathbf{A}$ we can optimize $\mathbf{B}$ and finally given $\mathbf{A}$ and $\mathbf{B}$ we can simply compute $\mathbf{X}$. In [Li, 2005], some variations of this general model, such as one side K-means and iterative feature and data clustering, has been introduced.

1.2.2 PROXIMUS

Another technique to solve (1.6) is provided in [Koyuturk et al., 2005, Koyuturk and Grama, 2003] and called Proximus. In Proximus, which satisfies the binary reconstruction constraint as well, one finds the latent factors one at the time. Given an input matrix $\mathbf{A}_{m\times n}$, the idea is to start with a column vector $x_{m\times 1}$ and find the best

corresponding row vector $y_{n\times 1}$. Vector x could be selected in different way: it could be a random binary vector, could be a random column of **A**, a combination of several columns of **A**, *etc*. Given x, which is called the presence vector, we can optimally find the best y, which is called pattern vector. More formally, we need to minimize the following quantity:

$$||\mathbf{A}-xy^T||_2 = ||\mathbf{A}||_2 - 2x^T\mathbf{A}y + ||x||_2||y||_2 \tag{1.9}$$

where $||.||_2$ denotes the Frobenius norm. This leads to maximizing $2x^T\mathbf{A}y-||x||_2||y||_2$. Fixing y, it is clear that x is optimized as the following:

$$x(i) = \begin{cases} 1 & if \;\; 2\mathbf{A}y \geq ||y||_2 \\ 0 & otherwise \end{cases}$$

The formulation which optimizes y (when x is fixed) is similar. The intuition is simple: having y fixed, we simply compare y^T with rows of **A**. If at least half of the ones in y is covered with a given row, we put a one in the corresponding cell of x, otherwise we set it to zero. Once we optimized x, we can fix it and optimize y. This iterative optimization is continued until x and y are stable. Consider the following example:

Example 1.3

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad y^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

To construct x, we need to compare y^T to all rows of **A** *to check if they can cover at least one of the ones in y^T (at leas half of the ones should be covered; as there are two ones, we need to cover at leas one of them),* i.e. *the first and the last cell (coverage means that the first and the last cell of the row of* **A** *is also one; for instance, denoting the first row of* **A** *as a_1, what we check is that if $a_1 y \geq ||y||_2/2$ or not). The third and the forth row of* **A** *satisfy this condition. As a result, the corresponding x for such y is:*

$$x^T = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

After obtaining a stable x and y, we simply remove, from **A**, the columns which have a value of one in the presence vector y. This ensures that the reconstructed matrix

is binary since the items in y in this step, will not be available in the next one (orthogonality of the pattern vectors). This also makes the algorithm run faster as we go forward since there are less and less columns in **A** and thus the computations can be done faster.

One main problem in Proximus is that, like all other similar approaches, it is sensitive to initialization. For instance, suppose that in the above-mentioned example, we start from $y^T = (1\ 1\ 1\ 1\ 1)$. This leads to a zero vector for x since there is no row in **A** that can cover at least 3 ones of y^T (at least half of the ones should be covered; as we have 5 ones, at least 3 of them should be covered). Then, starting with a zero vector as x will result in the same y. Since both vectors are stable, the algorithm will return them as a solution with the error value of 6 as 6 cells in **A** are approximated as zero while they were actually one. However if we started with a y as shown in the example, we would have ended in an approximation with an error value of 4. One way to tackle such problem is to run the algorithm several times with different initialization and take the best result as the solution. This is the strategy adapted in our experiments in the following chapters.

1.2.3 CBMF AND UBMF

All the methods explained so far solve the L_2 version of BMF. Another recent approach to solve (1.6) is the one introduced in [Jiang et al., 2014] which considers the L_1 case. This study addresses the BMF problem using a $K-$means approach. This technique, which is called CBMF (Constrained Binary Matrix Factorization), ensures the binary reconstruction by imposing the orthogonality. Fixing **W**, consider the following subproblem of optimizing j^{th} column of **H**, denoted as h, with respect to j^{th} column of **X**, denoted as x:

$$min \sum_{i=1}^{n} ||x_{ij} - \mathbf{W}h_{ij}||_1$$
$$\text{s.t. } ||h||_1 \leq 1 \quad \text{(orthogonality constraint)}$$

where the orthogonality constraint expresses that the column vector h can contain at most one non-zero element. The optimal solution to this problem is simply obtained by comparing the distance between vector x and all columns of **W**. If x is closer to a column of **W**, say w_t, than to the null vector, we simply set h_{tj} to 1. If x is closer to null vector than to the columns of **W**, then the optimal solution is also to

have a null vector as h. Based on that, the authors in [Jiang et al., 2014] establish a link between CBMF and $K-$means clustering by showing that fixing one factor and optimizing the other has the same optimal solution as $K-$means. Accordingly, the method proposed in [Jiang et al., 2014] uses a clustering technique: starting from K initial cluster representatives (which constructs the initial $\mathbf{W}$) from column vectors of $\mathbf{X}$, we assign each column vector of $\mathbf{X}$ to the class of the closest representative. Based on the columns assigned to each cluster, we build the first $\mathbf{H}$. New binary representatives (corresponding to a $0-1$ average) are then computed, and the process is repeated until the clusters do not evolve. A "zero" cluster, the representative of which does not change and is set to the null vector, is used to capture column vectors of $\mathbf{X}$ that are best approximated by the null vector in the reconstructed matrix. The final representatives thus obtained constitute the matrix $\mathbf{W}$, the matrix $\mathbf{H}$ being obtained through a simple operation. The following example illustrates this method.

Example 1.4 *Consider the following matrix* $\mathbf{X}$ *and suppose that the initial centers (which construct the initial* $\mathbf{W}$*, denoted as* $\mathbf{W}^1$*) are the first and the second column of* $\mathbf{X}$*, shown in red and blue respectively:*

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad \mathbf{W}^1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Then the $\mathbf{H}^1$ *obtained by the CBMF approach is:*

$$\mathbf{H}^1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

As the first column of $\mathbf{X}$ *is closer to the first center (they are identical), we assign it to the first cluster (*$H_{11} = 1$ *and consequently* $H_{21} = 0$ *as it is a hard clustering,* i.e. *each column is assigned to exactly one center). All other columns are closer to the second center (the second column of* $\mathbf{W}$*) and thus we simply set the other values of* $\mathbf{H}$*. In the next iteration, the first center remains unchanged (the first column of* $\mathbf{W}$ *shown in red) as it has only one column assigned to it (no averaging needed) while the second center (the second column of* $\mathbf{W}$ *shown in blue) does not: the new center is a* $0-1$ *averaging of the last three columns*

of **X**. *Accordingly, the new* **W**, *denoted as* $\mathbf{W}^2$ *will be:*

$$\mathbf{W}^2 = \begin{pmatrix} 1 & avg_{01}(0,1,0) \\ 1 & avg_{01}(1,0,0) \\ 1 & avg_{01}(0,0,0) \\ 0 & avg_{01}(1,1,1) \\ 0 & avg_{01}(0,0,1) \\ 0 & avg_{01}(0,1,1) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

where avg_{01} *denotes the* $0-1$ *averaging function.*

A deterministic two-approximation of the CBMF problem is provided based on the technique mentioned above. It consists of starting with all possible k subsets of columns of **X**. This method is extremely time consuming as there exists $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ different initialization which are followed by an iterative clustering procedure. Accordingly, a faster approach is not to search exhaustively all possibilities, but to start with a *good* one. This is where the authors use the $K-$means++ initialization [Arthur and Vassilvitskii, 2007a] to obtain the starting centers. K-means++ tries to find the initial centers in different areas of the space in order to cover all the search space (a random initialization, for instance, is strongly affected by the distribution of the data and may select many initial centers on the same region of the space).

A slight modification of this approach is also used in [Jiang et al., 2014], referred to as *UBMF*, to solve the problem without the binary reconstruction constraint. Using the results from [Hasewaga et al., 1993], it can be shown that CBMF yields a $4(2+\log K)$ approximation of the optimal solution.

Because of its reliance on K-means, CBMF can *a priori* be improved through local search procedures designed for K-means. Kanungo *et al.* [Kanungo et al., 2002] introduce such a local search, referred to here as the *swap procedure*, and that consists, when applied to our problem, in randomly selecting one of the final representatives of the current CBMF solution and replace it with one random column vector of **X** (the column vector of **X** becomes a new representative). The K-means algorithm is then applied to the new set of representatives; if the solution obtained is better, it becomes the new current solution. This process is repeated a certain number of times or until the time budget has been exhausted.

1.2.4 BOOLEAN DECOMPOSITION

A totally different decomposition technique is called *Boolean Matrix Factorization* [Miettinen et al., 2008, Miettinen, 2008a, Miettinen, 2008b]. In this technique the multiplication between the factors is no longer the ordinary multiplication but is the boolean one where $1+1=1$. Consider the following example where $\circ$ denotes the boolean matrix multiplication:

Example 1.5

$$x=\begin{pmatrix}1 & 1 & 0\end{pmatrix} \quad y=\begin{pmatrix}1\\1\\1\end{pmatrix} \quad x\circ y = 1*1+1*1+1*0=1$$

as $1+1=1$*, while and ordinary matrix decomposition, denoted as* $\times$*, results in a value of 2:*

$$x\times y = 1*1+1*1+1*0=2$$

The problem of boolean decomposition is expressed as:

$$min|\mathbf{A}-\mathbf{B}\circ\mathbf{C}|$$
$$\text{s.t} \quad \mathbf{A},\mathbf{B},\mathbf{C}\in\{0,1\}$$

where $\circ$ represents boolean multiplication. As SVD provides the otpimal rank-k representation of matrices with respect to the Frobenius norm, in [Miettinen et al., 2008], the authors introduce the notion of *boolean rank* which is computed when the operator is boolean.

In [Miettinen, 2008b], two boolean decomposition problems, namely *BCX* and *BCUR*, are considered which minimize $|\mathbf{A}-\mathbf{C}\circ\mathbf{X}|$ and $|\mathbf{A}-\mathbf{C}\circ\mathbf{U}\circ\mathbf{R}|$ respectively. Using Positive-Negative Partial Set Cover problem [Miettinen, 2008c], it is shown that these problems are NP-hard. Then a cover function, which is simply a trade-off function between the 1s covered and the 0s covered, is provided in order to find a solution to those problems.

The boolean decomposition, though very interesting, is not studied in our research as it has a completely different interpretation and theories. We here focus on the ordinary decomposition and the theorems and examples provided later will not be necessarily valid for the boolean decomposition case.

1.3 RELATION TO UBQP

So far, we have seen similar problems and the state-of-the-art in binary matrix decomposition domain. In general, one way to solve a new problem is to find a relation or a link to other known problems and use the algorithms designed for the known problem in order to solve the new problem. In this section, we follow this line of thought and establish a link between the BMF problem and Unconstrained Binary Quadratic Programming (UBQP) problem as our first contribution [Mirisaee et al., 2015c, Mirisaee et al., 2015b]. This allows us to use the UBQP heuristics [Beasley, 1998] in order to tackle the BMF problem in the following chapter.

UBQP is a quadratic maximization problem with binary variables. The general formulation of this problem is to maximize the following:

$$f(x) = x^T Q x = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij} x_i x_j \quad \text{subject to} \quad x_i \in \{0,1\}\ \forall i = 1,...,n \tag{1.10}$$

This problem is NP-hard and has many applications such as financial analysis problems and machine scheduling [Merz and Freisleben, 2002]. According to its wide range of applications, the solutions to UBQP have always been of interest. Some exact methods have been proposed to solve (1.10) such as [Helmberg and Rendl, 1998, Billionnet and Sutter, 1994]; however, according to the computational complexity, researchers tried to find heuristics for that. Tabu search and simulated annealing are some examples of these heuristics [Beasley, 1998, Katayama et al., 2000].

As mentioned before, establishing a link between the BMF problem and the UBQP problem allows us to use the heuristics developed for UBQP in order to solve the BMF problem or improve its solutions. In the following, we show that such relation exists when the L_2 norm is concerned and then, in the next chapter, we use it as a technique to improve the solutions of L_2-BMF and compare it to the one proposed therein.

As explained in the previous sections, standard methods for L_2-BMF fix one matrix, $\mathbf{W}$ or $\mathbf{H}$, and solve for the other. The quantity to be minimized in L_2-BMF can be rewritten as:

$$||\mathbf{X} - \mathbf{WH}||_2^2 = \sum_{i=1}^{M}\sum_{j=1}^{N}(x_{ij} - \sum_{k=1}^{K} w_{ik}h_{kj})^2$$

with:

$$(x_{ij} - \sum_{k=1}^{K} w_{ik}h_{kj})^2 = x_{ij}^2 + (\sum_{k=1}^{K} w_{ik}h_{kj})^2 - 2x_{ij}\sum_{k=1}^{K} w_{ik}h_{kj}$$

Fixing e.g. $\mathbf{H}$ and solving for $\mathbf{W}$ thus amounts to solve the following minimization problem, $\forall i,\ 1 \leq i \leq M$ (*i.e.* for all rows of $\mathbf{W}$):

$$\text{argmin}_{\mathbf{w}_{i.}} \sum_{j=1}^{N} (x_{ij}^2 + (\sum_{k=1}^{K} w_{ik} h_{kj})^2 - 2x_{ij} \sum_{k=1}^{K} w_{ik} h_{kj}) \tag{1.11}$$

where $\mathbf{w}_{i.}$ is the i^{th} row vector of $\mathbf{W}$. The above minimization problem can be rewritten as

$$\text{argmin}_{\mathbf{w}_{i.}} \sum_{j=1}^{N} (\sum_{k=1}^{K} w_{ik} h_{kj})^2 + \sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N} (-2x_{ij} h_{kj}) \tag{1.12}$$

since x_{ij} is not subject to any change and does not play any role in the minimization problem (it is only a constant). The first term in (1.12), $\sum_{j=1}^{N} (\sum_{k=1}^{K} w_{ik} h_{kj})^2$, can be rewritten as:

$$\sum_{j=1}^{N} (\sum_{k=1}^{K} w_{ik} h_{kj})^2 = \sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N} h_{kj} + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik} w_{ik'} 2 \sum_{j=1}^{N} h_{kj} h_{k'j} \tag{1.13}$$

Now if we put (1.13) in (1.12), we need to minimize the following quantity with respect to $w_{i.}$:

$$\underbrace{\sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N} h_{kj}}_{\alpha} + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik} w_{ik'} 2 \sum_{j=1}^{N} h_{kj} h_{k'j} + \underbrace{\sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N} (-2x_{ij} h_{kj})}_{\beta} \tag{1.14}$$

Now we factorize $\sum_{k=1}^{K} w_{ik}$ in α and β:

$$\sum_{k=1}^{K} w_{ik} (\sum_{j=1}^{N} h_{kj} - \sum_{j=1}^{N} (2x_{ij} h_{kj})) + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik} w_{ik'} 2 \sum_{j=1}^{N} h_{kj} h_{k'j} \tag{1.15}$$

which is equal to

$$\sum_{k=1}^{K} w_{ik} (\sum_{j=1}^{N} h_{kj} (1 - 2x_{ij})) + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik} w_{ik'} 2 \sum_{j=1}^{N} h_{kj} h_{k'j} \tag{1.16}$$

Now, let us consider the symmetric, $K \times K$ matrix $\mathbf{Q}$:

$$q_{kk} = \sum_{j=1}^{N} h_{kj} (1 - 2x_{ij}),\ q_{kk'} = \sum_{j=1}^{N} h_{kj} h_{k'j}\ (k \neq k')$$

The quantity minimized in problem (1.12) can be rewritten with $\mathbf{Q}$ as follows:

$$\sum_{k=1}^{K} w_{ik}^2 q_{kk} + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} 2 w_{ik} w_{ik'} q_{kk'} = \mathbf{w}_{i.} \mathbf{Q} \mathbf{w}_{i.}^T$$

and thus problem (1.12) can be reformulated as:

$$\text{argmin}_{\mathbf{w}_{i.}}\ \mathbf{w}_{i.} \mathbf{Q} \mathbf{w}_{i.}^T \tag{1.17}$$

which corresponds to a UBQP problem [Beasley, 1998, Merz and Freisleben, 2002]. One should note that the placement of the transposed vector is not important in this problem and it could be either before the matrix of weights (**Q**) or after that. Applying the same development on **H** when **W** is fixed leads to the following property.

Property 1.1 *Iteratively solving the L_2-BMF problem by fixing either* **W** *or* **H** *and solving for the other is equivalent to iteratively solving UBQP problems of the form:*

$$\text{argmin}_{\mathbf{v}}\, \mathbf{v}^T \mathbf{Q} \mathbf{v} \tag{1.18}$$

with:

$$q_{kk} = \sum_{j=1}^{N} h_{kj}(1-2x_{ij}) \qquad OR \qquad \sum_{i=1}^{M} w_{ik}(1-2x_{ij})$$

$$q_{kk'} = \sum_{j=1}^{N} h_{kj}h_{k'j}\ (k \neq k') \quad OR \qquad \sum_{i=1}^{M} w_{ik}w_{ik'}\ (k \neq k')$$

Note that when $k = 1$, the optimal **W** or **H**, when the other one is fixed, is directly obtained by setting the i^{th} element of **W** (or **H**) to 1 if more than half of the elements with a 1 in **H** (or **W**) have also a 1 in the i^{th} row (or column) of **X**, and to 0 otherwise. This corresponds to the alternating strategy used in Proximus. When $k > 1$, this not longer holds and approximate solutions are usually obtained. Resorting to UBQP solvers in this case does not however represent an interesting alternative, as the solutions provided by *e.g.* NMF in the continuous space are in general faster to obtain and the projection on the binary domain can be done efficiently as mentioned in previous sections. Nevertheless, it might still be interesting to improve the NMF or Proximus solutions through local search algorithms designed for UBQP. This issue will be studied deeply in the following chapter.

One should note that, based on the developments provided so far, we can iteratively solve the BMF problem. The idea is to consider one row of **W**, say $w_{i.}$, at the time. Using the Property 1.1, one can simply compute the matrix of weights **Q**. Having matrix **Q**, we can directly apply the heuristics of UBQP to the BMF problem. In the next chapter, we will see how matrix **Q** needs to be updated and, using that, we will have a heuristic with a complexity of $O(KN)$ for updating one row of **W**. With the same reasoning, one can see updating a column of **H** has the same complexity. We will see the complexity analysis in detail in the next chapter.

In such updating procedure, once we are done with all rows of **W**, we redo the process with columns of **H** and optimize them with respect to columns of **X**. As before, the computation of **Q** is expensive for the first column and, only after that, it can be

updated more quickly. One should note that for the second round of optimizing **W**, matrix **Q** should be again computed completely for the very first row as **H** has been changed compared to the first iteration. More detailed analysis of the complexities as well as a more efficient technique for updating **W** and **H** will be provided in the next chapter.

1.4 Closing remarks

In this chapter, we first studied the importance of matrix factorization techniques in data analysis and understanding. As mentioned, these techniques can simplify the data and find latent factors or patterns. We then considered a particular decomposition method which is of interest in many applications, namely binary matrix factorization (BMF), in which both the input and the decomposition factors are binary, and the reconstruction could be optionally binary as well. Finally, it has been shown, as our first contribution [Mirisaee et al., 2015c,Mirisaee et al., 2015b], that the BMF problem can be reformulated as a UBQP problem. This reformulation will allow us to use the heuristics of UBQP in BMF cases. In the following chapter, we see how we can use this reformulation in order to establish a heuristic local search for BMF. We then provide a more efficient local search and show (theoretically and experimentally) its efficiency with respect to the UBQP approach and the standard approach.

2 HEURISTIC LOCAL SEARCH FOR BMF

2.1 INTRODUCTION

Local search algorithms are heuristics aimed at improving a current solution by searching in its neighborhood for a better solution (hence the name "local"). For instance in the traveling salesman problem (TSP) problem, a 2-change neighborhood is a set of tours that are different in only two edges [Johnson et al., 1988]. In other words, a local search is a search in which, given an independently obtained solution, we try to find a better solution repeatedly in a "close" neighborhood.

In the context of BMF, the neighborhood of size p of a given $\mathbf{W}$ (or $\mathbf{H}$) is defined by the set of matrices that can be obtained from $\mathbf{W}$ (or $\mathbf{H}$) by flipping at most p cells (here, flipping means changing a 1 to a 0 and *vice versa*). The following example illustrates this point.

Example 2.1 *Consider the following toy matrices of CBMF (see Section 1.2.3) decomposition of the form* $\mathbf{X} = \mathbf{W} \times \mathbf{H} + \epsilon$ *where* $||\epsilon||_1 = 5$.

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In this example, we can simply look at a neighborhood of size one of $\mathbf{H}$, i.e. *flipping only one bit of* $\mathbf{H}$. *Doing a simple bit flip operation, one can see that by flipping (from 0 to 1) the cell shown in red in* $\mathbf{H}$, *we obtain the following solution of the form* $\mathbf{X} = \mathbf{W} \times \mathbf{H}' + \epsilon'$

where $||\epsilon'||_1 = 3$ *which is an improvement over the initial decomposition.*

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

According to this example, we can observe the importance of a local search. Obviously, the main question is "*which bit should we flip?*". One naive approach is to simply flip every single bit and check if the new solution is better or not, which is clearly an expensive solution considering the number of bits to check in each round. One may also think of flipping the first bit which brings improvement to the solution and drop the rest of the checking process. Computational-wise, what is difficult to do is the multiplication required in order to see if flipping the bit under consideration can improve the solution or not. Since this approach is iterative and needs to be done for each bit until stabilization, it could be very time consuming. Accordingly, different studies have been dedicated to find fast local heuristics for this problem.

The UBQP problem and its relation to BMF have been discussed in the previous chapter. As mentioned there, the UBQP is formulated as a maximization problem, where the function to maximize is:

$$f(x) = x^T Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j \quad \text{subject to} \quad x_i \in \{0,1\} \; \forall i = 1, ..., n$$

Having a link between the UBQP problem and BMF allows us to use the heuristics developed for UBQP in the BMF domain. One of these heuristics has been studied in [Beasley, 1998]. In that paper, according to the hardness of UBQP, Beasley proposes two heuristics: one based on tabu search and the other based on simulated-annealing. By introducing a new variable, the UBQP problem has been reformulated as binary linear program where the convex hull of the solution of constraints is known as a boolean quadratic polytope [Mehrotra, 1997]. One can simply relax this binary programming to find the bounds on the optimal solution of the UBQP problem. The idea of the tabu search proposed in [Beasley, 1998] is to start from a zero vector x, and then to examine all non-tabu moves. In the simulated-annealing method, Beasley proposes to set the initial temperature to the problem size (size of x) and the parameter α to 0.995. The computational results show that these methods can solve fairly large sizes of UBQP problem in a reasonable amount of time.

In [Merz and Freisleben, 2002] a local search heuristics, called *p-opt local search* for UBQP is presented. The *p-opt* local search algorithm proposed in [Merz and Freisleben, 2002] looks for the solution in the neighborhood of size p that maximizes the gain with respect to the current solution and adopts this solution if the gain is positive. The *p-opt* local search, which parallels the tabu search of [Glover F., 1998] discussed in [Beasley, 1998] and based on [Kernighan and Lin, 1970] and [Lin and Kernighan, 1973], is similar for the neighborhood of size p, except that, for computational reasons, one does not look for the solution that maximizes the gain but only explores a part of the neighborhood looking for a solution that improves the current one. This exploration corresponds to a recursive application of the *1-opt* solution.

p-opt local search algorithms are of course only interesting if the gains can be computed efficiently; the complexity of the *p-opt* local search algorithm proposed in [Merz and Freisleben, 2002], when applied to the i^{th} row of $\mathbf{W}$ in our case (problem 1.17) is $O(K^2)$ once the matrix $\mathbf{Q}$, which depends on i, has been constructed. One can recall, from the previous chapter, that matrix $\mathbf{Q}$ in case of BMF is computed is the following

$$q_{kk} = \sum_{j=1}^{N} h_{kj}(1-2x_{ij}) \qquad \text{OR} \qquad \sum_{i=1}^{M} w_{ik}(1-2x_{ij})$$

$$q_{kk'} = \sum_{j=1}^{N} h_{kj}h_{k'j} \ (k \neq k') \quad \text{OR} \qquad \sum_{i=1}^{M} w_{ik}w_{ik'} \ (k \neq k')$$

In each round of updating $\mathbf{W}$, matrix $\mathbf{Q}$ is constructed for the first row with a cost of $O(K^2N)$ as the diagonal of $\mathbf{Q}$ can be computed via K dot products of vectors of size N and the upper triangle of $\mathbf{Q}$ has $\frac{K(K-1)}{2}$ elements, each of which requiring a dot product with vectors of size N. Note that $\mathbf{Q}$ is symmetric and we only need to compute the upper (or the lower) triangle. Once $\mathbf{Q}$ is constructed for the first row of $\mathbf{W}$, it can then be reused for other rows with a simple update on the diagonal which can be done in $O(KN)$, as explained before. Accordingly, computing $\mathbf{Q}$ for all rows of $\mathbf{W}$ has the time complexity of $O(KMN)$ as it is one time K^2N and $M-1$ times KN. As the heuristic of [Merz and Freisleben, 2002] has a time complexity of $O(K^2)$ for each row of $\mathbf{W}$, for M rows we have $O(MK^2)$ for the heuristic phase. As a result, the total complexity for updating $\mathbf{W}$ in one round is $O(MKN + MK^2)$ and since $N \geq K$ the complexity will be $O(MKN)$. Dividing this quantity by the number of rows, M, we will have $O(KN)$ as the complexity of updating one row of $\mathbf{W}$ using the *1-opt* discussed in [Merz and Freisleben, 2002]. We denote this procedure as *1-opt-UBQP*. One can simply see that updating a column of $\mathbf{H}$ has exactly the same complexity.

As mentioned in the previous chapter, BMF has been used in many applications recently. Consequently, in this chapter, we focus on heuristic local search for BMF. As discussed so far, there are different BMF approaches such as (projected) NMF and (projected) SVD, CBMF, UBMF and Proximus. In the following section, as our second contribution [Mirisaee et al., 2015b, Mirisaee et al., 2015c], we provide an efficient heuristic which, given a solution, finds a better solution in a close neighborhood. We will then show that this technique is faster than the state-of-the-art methods, and the improvement is statistically significant in many cases.

The local search introduced here is known as *p-opt* and it consists of, when applied to our problem, finding new solutions by flipping p bits. Finding the best p bits to flip can be done in a greedy or an exhaustive way. Here, we focus on *1-opt*, in which only one bit is flipped at a time. However, the procedure to apply the *p-opt* is simply a repeated use of *1-opt*.

2.2 *p-opt* LOCAL SEARCH FOR BMF

We describe here a *p-opt* local search procedure dedicated to rank K binary matrix factorization. As mentioned before, rows of $\mathbf{W}$ (or columns of $\mathbf{H}$) can be optimized independently of each other. We show here how to optimize, in a neighborhood of size 1, a row of $\mathbf{W}$ (the reasoning is the same for columns of $\mathbf{H}$). For each row $\mathbf{w}_{i.}$ of $\mathbf{W}$, we first compute a partition of columns of $\mathbf{H}$. The main rational of such partitioning is to compute, for each partition, the gains separately. Furthermore, using this partitioning, one is able to replace many vector multiplications with straightforward summations. As mentioned before, the computational cost of the *1-opt* procedures are increased when multiplications are involved. Accordingly, if one can avoid (at least a part of if not all) multiplications, s/he can obtain a faster *1-opt* procedure. In the following, we first show how this partitioning is done. We then provide examples to illustrate how this technique works on real matrices and how it can accelerate the process.

Definition 2.1 *For a given row* $\mathbf{w}_{i.}$ $(1 \leq i \leq M)$ *of* $\mathbf{W}$ *and a given* $\mathbf{H}$*, we define three sets of column vectors of* $\mathbf{H}$*:*

(i) $W_i^0 = \{\mathbf{h}_{.l} \mid 1 \leq l \leq N,\ x_{il} = 0\}$

(ii) $W_i^{\perp} = \{\mathbf{h}_{.l} \mid 1 \leq l \leq N,\ x_{il} = 1,\ < \mathbf{w}_{i.}, \mathbf{h}_{.l} >= 0\}$

(iii) $W_i^{\not\perp} = \{\mathbf{h}_{.l} \mid 1 \leq l \leq N,\ x_{il} = 1,\ < \mathbf{w}_{i.}, \mathbf{h}_{.l} >\neq 0\}$

where $\mathbf{h}_{.l}$ *is the binary vector corresponding to the* l^{th} *column of* $\mathbf{H}$*,* x_{il} *the cell of* $\mathbf{X}$ *at row* i *and column* l*;* $< .,. >$ *denotes the dot product. We have of course:* $|W_i^0|+|W_i^{\perp}|+|W_i^{\not\perp}| = N$.

The following example illustrates the three sets defined above.

Example 2.2

Let $\mathbf{x}_{i.} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$, $\mathbf{w}_{i.} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$ *and* $\mathbf{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$

Then:

$$W_i^0 = \{\mathbf{h}_{.3}, \mathbf{h}_{.4}\} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

$$W_i^{\perp} = \{\mathbf{h}_{.2}, \mathbf{h}_{.5}\} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$W_i^{\not\perp} = \{\mathbf{h}_{.1}, \mathbf{h}_{.6}\} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Given a row of $\mathbf{W}$ (resp. a column of $\mathbf{H}$), the idea is to exploit the characteristics and the distribution of the corresponding row (resp. column) of $\mathbf{X}$ and $\mathbf{H}$ (resp. $\mathbf{W}$) in order to compute the gain we obtain by flipping a bit. What we need now, in order to define an efficient *p-opt* local search, is a fast way to compute the gain when flipping the j^{th} bit of $\mathbf{w}_{i.}$ from 0 to 1 and *vice versa*. Such a gain (which can be positive or negative) is defined as

$$\Delta E(i,j) = E(i,j)_{\text{new}} - E(i,j)_{\text{old}} = ||\mathbf{x}_{i.} - \mathbf{w}_{i.}^j \mathbf{H}||_p - ||\mathbf{x}_{i.} - \mathbf{w}_{i.} \times \mathbf{H}||_p$$

where $\mathbf{w}_{i.}^j$ is obtained from $\mathbf{w}_{i.}$ by flipping the j^{th} bit (either from 0 to 1 or from 1 to 0). Theorem 2.1 provides simple expressions for $\Delta E(i,j)$ for both L_1 and L_2 norms; expressions that can be computed efficiently (with respect to other methods), as it will be shown in Theorem 2.2.

Theorem 2.1 *Let $\Delta E(i,j;0\rightarrow 1,L_2)$ (resp. $\Delta E(i,j;1\rightarrow 0,L_2)$) be the gain obtained when flipping the j^{th} bit of $\mathbf{w}_{i.}$ from 0 to 1 (resp. from 1 to 0), measured by the L_2-norm. Then:*

$$\Delta E(i,j;0\rightarrow 1,L_2)=\sum_{\mathbf{h}\in W_i^0}(1+2<\mathbf{w}_{i.},\mathbf{h}>)h_j+\sum_{\mathbf{h}\in W_i^{\not\perp}}(2<\mathbf{w}_{i.},\mathbf{h}>-1)h_j-\sum_{\mathbf{h}\in W_i^{\perp}}h_j \quad (2.1)$$

$$\Delta E(i,j;1\rightarrow 0,L_2)=\sum_{\mathbf{h}\in W_i^0}(1-2<\mathbf{w}_{i.},\mathbf{h}>)h_j+\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j=1}}1+\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}}3-2<\mathbf{w}_{i.},\mathbf{h}> \quad (2.2)$$

Similarly for the L_1-norm we have:

$$\Delta E(i,j;0\rightarrow 1,L_1)=\sum_{\mathbf{h}\in W_i^0}h_j+\sum_{\mathbf{h}\in W_i^{\not\perp}}h_j-\sum_{\mathbf{h}\in W_i^{\perp}}h_j \quad (2.3)$$

$$\Delta E(i,j;1\rightarrow 0,L_1)=\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j=1}}1-\sum_{\mathbf{h}\in W_i^0}h_j-\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}}1 \quad (2.4)$$

Proof For each of these formulations, we consider the error change with respect to the sets defined in Def. 2.1.

$\Delta E(i,j;0\rightarrow 1,L_2)$

- First consider the W_i^0 set: before the flip, the error E_0^1, of $\mathbf{w}_{i.}$ wrt W_i^0 is:

$$E_0^1=\sum_{\mathbf{h}\in W_i^0}<\mathbf{w}_{i.},\mathbf{h}>^2$$

  After the flip, this error becomes

$$E_0^2=\sum_{\mathbf{h}\in W_i^0}(<\mathbf{w}_{i.},\mathbf{h}>+h_j)^2$$

  as the elements of W_i^0 correspond to the cells in $\mathbf{X}$ having zero values (Def. 2.1). Accordingly, flipping the j^{th} cell of w_i from 0 to 1 will potentially increase the

error depending on the value of h_j; thus, the difference in errors amounts to:

$$E_0^2 - E_0^1 = \sum_{\mathbf{h}\in W_i^0} (<\mathbf{w}_{i.},\mathbf{h}> + h_j)^2 - \sum_{\mathbf{h}\in W_i^0} <\mathbf{w}_{i.},\mathbf{h}>^2 \tag{2.5}$$

$$= \sum_{\mathbf{h}\in W_i^0} \left((<\mathbf{w}_{i.},\mathbf{h}> + h_j)^2 - <\mathbf{w}_{i.},\mathbf{h}>^2\right) \tag{2.6}$$

$$= \sum_{\mathbf{h}\in W_i^0} <\mathbf{w}_{i.},\mathbf{h}>^2 + 2<\mathbf{w}_{i.},\mathbf{h}> h_j + h_j^2 - <\mathbf{w}_{i.},\mathbf{h}>^2 \tag{2.7}$$

$$= \sum_{\mathbf{h}\in W_i^0} 2<\mathbf{w}_{i.},\mathbf{h}> h_j + h_j^2 \tag{2.8}$$

$$= \sum_{\mathbf{h}\in W_i^0} (1 + 2<\mathbf{w}_{i.},\mathbf{h}>) h_j \tag{2.9}$$

Note that in (2.8) we can replace h_j^2 with h_j as matrix $\mathbf{H}$ is binary.

- For $W_i^{\not\perp}$ case, we denote the error before the flip as $E_{\not\perp}^1$ and the error after the flip as $E_{\not\perp}^2$: before the flip we have

$$E_{\not\perp}^1 = \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.},\mathbf{h}> - 1)^2$$

as we need to produce 1 in this case (see Def. 2.1) and thus the initial error for the columns in $W_i^{\not\perp}$ amounts to $(<\mathbf{w}_{i.},\mathbf{h}> - 1)^2$ since we are considering the L_2-norm. After the flip, we are *potentially* introducing more error since flipping a 0 to a 1 will increase the dot product if h_j is also equal to 1 (if it is equal to zero, no additional error is introduced). Accordingly, the new error becomes

$$E_{\not\perp}^2 = \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.},\mathbf{h}> - 1 + h_j)^2$$

The difference amounts to:

$$E_{\not\perp}^2 - E_{\not\perp}^1 = \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.},\mathbf{h}> - 1 + h_j)^2 - \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.},\mathbf{h}> - 1)^2 \tag{2.10}$$

$$= \sum_{\mathbf{h}\in W_i^{\not\perp}} \left((<\mathbf{w}_{i.},\mathbf{h}> - 1 + h_j)^2 - (<\mathbf{w}_{i.},\mathbf{h}> - 1)^2\right) \tag{2.11}$$

$$= \sum_{\mathbf{h}\in W_i^{\not\perp}} (2<\mathbf{w}_{i.},\mathbf{h}> - 1) h_j \tag{2.12}$$

- Finally, we consider $W_i^{\perp}$: the error, before the flip, is the cardinality of the set (since the dot product is 0 whereas it should be 1). As a result, the flip can only

reduce this error and it does so for all the vectors in $W_i^{\perp}$ whose j^{th} element is 1. Accordingly, following our notations, the difference in error is thus

$$E_{\perp}^2 - E_{\perp}^1 = -\sum_{\mathbf{h}\in W_i^{\not\perp}} h_j \tag{2.13}$$

Summing (2.9), (2.12) and (2.13) yields Equation 2.1.

$\Delta E(i, j; 1 \to 0, L_2)$

- As before, the error with respect to W_i^0 is

$$E_0^1 = \sum_{\mathbf{h}\in W_i^0} < \mathbf{w}_{i.}, \mathbf{h} >^2$$

before the flip. After the flip, since we are changing a 1 to a 0, we are potentially decreasing the error. The reason is that columns in W_i^0 need to produce zeros when multiplied by $w_{i.}$ while the dot product may produce other values. As a result, when the j^{th} bit of $w_{i.}$ flipped to zero, it could decrease the error depending on the value of h_j; accordingly, the new error is simply

$$E_0^2 = \sum_{\mathbf{h}\in W_i^0} (< \mathbf{w}_{i.}, \mathbf{h} > -h_j)^2$$

and the difference amounts to:

$$\begin{aligned}
E_0^2 - E_0^1 &= \sum_{\mathbf{h}\in W_i^0} (< \mathbf{w}_{i.}, \mathbf{h} > -h_j)^2 - \sum_{\mathbf{h}\in W_i^0} < \mathbf{w}_{i.}, \mathbf{h} >^2 && (2.14)\\
&= \sum_{\mathbf{h}\in W_i^0} \left((< \mathbf{w}_{i.}, \mathbf{h} > -h_j)^2 - < \mathbf{w}_{i.}, \mathbf{h} >^2 \right) && (2.15)\\
&= \sum_{\mathbf{h}\in W_i^0} < \mathbf{w}_{i.}, \mathbf{h} >^2 - 2 < \mathbf{w}_{i.}, \mathbf{h} > h_j + h_j^2 - < \mathbf{w}_{i.}, \mathbf{h} >^2 && (2.16)\\
&= \sum_{\mathbf{h}\in W_i^0} -2 < \mathbf{w}_{i.}, \mathbf{h} > h_j + h_j^2 && (2.17)\\
&= \sum_{\mathbf{h}\in W_i^0} (1 - 2 < \mathbf{w}_{i.}, \mathbf{h} >) h_j && (2.18)
\end{aligned}$$

Here, we can again replace h_j^2 with h_j in (2.17) as matrix $\mathbf{H}$ is binary.

- For a given $\mathbf{h} \in W_i^{\not\perp}$, two situations arise: case *(i)* is when, for a given column $\mathbf{h} \in W_i^{\not\perp}$ we have $< \mathbf{w}_i, \mathbf{h} > h_j = 1$, which means the dot product and the j^{th} cell of h are both 1. In this case, if we flip the j^{th} cell of $w_{i.}$ to 1, we have added one

error per column. So we have the following quantity as error increase (which is simply the quantity of the set):

$$\sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j=1}} 1 \tag{2.19}$$

Case *(ii)* is when, for a given column $\mathbf{h} \in W_i^{\not\perp}$, we have $(<\mathbf{w}_i, \mathbf{h}>)h_j > 1$, which means that $h_j = 1$ and $<\mathbf{w}_i, \mathbf{h}>$ is greater than 1. In such case, the initial error we are making is

$$E_{\not\perp}^1 = \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} (<\mathbf{w}_{i.},\mathbf{h}>-1)^2$$

and by flipping the j^{th} bit results in adding one extra error:

$$E_{\not\perp}^2 = \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} (<\mathbf{w}_{i.},\mathbf{h}>-2)^2$$

Accordingly, we have

$$E_{\not\perp}^2 - E_{\not\perp}^1 = \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} (<\mathbf{w}_{i.},\mathbf{h}>-2)^2 - \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} (<\mathbf{w}_{i.},\mathbf{h}>-1)^2 \tag{2.20}$$

$$= \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} (<\mathbf{w}_{i.},\mathbf{h}>-2)^2 - (<\mathbf{w}_{i.},\mathbf{h}>-1)^2 \tag{2.21}$$

$$= \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} 3 - 2<\mathbf{w}_{i.},\mathbf{h}> \tag{2.22}$$

One should note that if $h_j = 0$ flipping the corresponding bit in $w_{i.}$ from 0 to 1 does not change the error as, in any case, their multiplication will be zero.

- Lastly, the error on $W_i^{\perp}$ is not affected by the flip as $\mathbf{w}_{i.}^j$ is already orthogonal to all the elements of $W_i^{\perp}$ and flipping a 1 to 0 will not change this orthogonality and, as a result, does not change the error value.

Summing (2.18), (2.19) and (2.22) yields Equation 2.2.

$\Delta E(i, j; 0 \to 1, L_1)$

- For columns in W_i^0, if we flip a 0 to a 1 in $w_{i.}$, in case the corresponding cell of a given column $h \in W_i^0$ is zero, no extra error is introduced as the multiplication

would be zero in any case. However if the corresponding cell is 1, then one new error is introduced. So the error in this case is simply:

$$\sum_{\mathbf{h}\in W_i^0} h_j \tag{2.23}$$

- For columns of $W_i^{\not\perp}$, we need to produce 1. Consequently, the initial error is

$$E_{\not\perp}^1 = \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.}, \mathbf{h}> -1)$$

and the new error will be

$$E_{\not\perp}^2 = \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.}, \mathbf{h}> -1 + h_j)$$

as flipping the j^{th} bit from 0 to 1 will introduce one new error if h_j is also equal to 1. As a result:

$$E_{\not\perp}^2 - E_{\not\perp}^1 = \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.}, \mathbf{h}> -1 + h_j) - \sum_{\mathbf{h}\in W_i^{\not\perp}} (<\mathbf{w}_{i.}, \mathbf{h}> -1) \tag{2.24}$$

$$= \sum_{\mathbf{h}\in W_i^{\not\perp}} <\mathbf{w}_{i.}, \mathbf{h}> -1 + h_j - <\mathbf{w}_{i.}, \mathbf{h}> +1 \tag{2.25}$$

$$= \sum_{\mathbf{h}\in W_i^{\not\perp}} h_j \tag{2.26}$$

- For columns of $W_i^{\perp}$, we may be able to decrease the error by flipping a bit from 0 to 1 as, initially, $w_{i.}$ is orthogonal to the columns of $W_i^{\perp}$, *i.e* the dot product produces zero while we expect a 1. As a result, if we flip the j^{th} cell to 1, then we can decrease the error only if the corresponding cell in $h \in W_i^{\perp}$ is also equal to one. Consequently the error change in this case is:

$$-\sum_{\mathbf{h}\in W_i^{\perp}} h_j \tag{2.27}$$

Summing (2.23), (2.26) and (2.27) yields Equation 2.3.

$\Delta E(i, j; 1 \to 0, L_1)$

- For W_i^0, as we need to produce a zero for each dot product, flipping a bit from 1 to 0 can only cause error decrease. This will happen if the corresponding cell in $h \in W_i^0$ is 1. Thus, the error change is:

$$-\sum_{\mathbf{h}\in W_i^0} h_j \tag{2.28}$$

- For columns in $W_i^{\not\perp}$, we again have two case: *(i)* when have $< \mathbf{w}_i, \mathbf{h} > h_j = 1$ in which, by the same token as before, we are introducing one error per column, which means

$$\sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j = 1}} 1 \tag{2.29}$$

  Case *(ii)* is when $(< \mathbf{w}_i, \mathbf{h} >) h_j > 1$, which means that $h_j = 1$ and $< \mathbf{w}_i, \mathbf{h} >$ is greater than 1. In this case, by flipping 1 to 0, we decrease the error one unit per column (as the corresponding cell, h_j, is equal to one for all columns in this set). So the difference in the error is given by

$$- \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} 1 \tag{2.30}$$

- For $W_i^{\perp}$, we do not introduce any new error by flipping a bit from 1 to 0 as the dot product is already zero and setting an element to zero does not change the result of this product.

Summing (2.28), (2.29) and (2.30) yields Equation 2.3.□

Theorem 2.1 provides a direct way to compute the gain associated to each possible flip of an element of $\mathbf{w}_{i.}$, which can be used in a *1-opt* local search procedure as defined in Algorithm 2 (in this algorithm, we use $\Delta E(i, j; L_p)$ which is equal to $\Delta E(i, j; 0 \rightarrow 1, L_p)$ if $w_{ij} = 0$ and to $\Delta E(i, j; 1 \rightarrow 0, L_p)$ otherwise). The following example illustrates how one can utilize Theorem 2.1 in practice.

Example 2.3 *Consider the following configuration where we aim at optimizing the i^{th} row of* $\mathbf{W}$ *(denoted by $w_{i.}$) having the corresponding row of* $\mathbf{X}$ *(denoted by $x_{i.}$) and* $\mathbf{H}$*:*

$$\mathbf{x}_{i.} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}, \mathbf{w}_{i.} = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \textit{and } \mathbf{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Using Def. 2.1, we have the following sets:

$$W_i^0 = \{\mathbf{h}_{.3}, \mathbf{h}_{.4}\} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad W_i^{\perp} = \{\mathbf{h}_{.5}\} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$W_i^{\not\perp} = \{\mathbf{h}_{.1}, \mathbf{h}_{.2}, \mathbf{h}_{.6}\} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

What we need to do is to compute the error change (gain) for zero cells and one cells in $w_{i.}$ separately. Let $\mathbb{G} = (g_1^{0\to1},\ g_2^{1\to0},\ g_3^{1\to0})$ denote the gain for flipping the bits where the subscripts show the index of corresponding cell in $w_{i.}$ and the superscripts show whether we should use the zero-to-one formulations in Theorem 2.1 or the one-to-zero formulations. Having vector $\mathbb{G}$ computed, we can decide which bit to flip in order to maximize the improvement. We consider both the L_2 and the L_1 cases:

- *L_2: we obtain $g_1^{0\to1} = +8$ using the Eq. (2.1). Similarly, if we use Eq. (2.2) for $g_2^{1\to0}$ and $g_3^{1\to0}$ we obtain -3 and -4 respectively. As a result the gain vector is $\mathbb{G} = (+8,\ -3,\ -4)$ which means that the best bit to flip is the last one as it has the highest negative gain.*

- *L_1: in this case we use Eq. (2.3) for the first cell and Eq. (2.4) for the second and the third one which gives us the following gain vector $\mathbb{G} = (+2,\ -1,\ -2)$. In this case, we select the third bit to flip as it leads to the best improvement comparing to the other bits.*

The procedure expressed in Algorithm 2 can be naturally extended to *p-opt* local search, either by adopting a *greedy* approach, in which case one applies p times a *1-opt* local search, as done in [Merz and Freisleben, 2002], or by finding the p flips that maximize the gain. In both cases, the gain of flipping several bits is the sum of the gain of the individual flips. However, in the latter approach, the complexity for selecting the best bits to flip is in $O(K^p)$, which prevents its application in most practical cases (in contrast, the complexity of the *p-opt* greedy approach to select the p bits is $O(pK)$). In the remainder, and as done in many studies, when we talk about the *p-opt* local search, we refer to the *p-opt* greedy approach. Lastly, Algorithm 2 (and its *p-opt* extension) can also be adapted to cover the formulation of the BMF problem with the binary reconstruction constraint by accepting a flip only if the result is a binary vector[1].

As mentioned previously in this chapter, the complexity of updating one row (say row i) of $\mathbf{W}$ through the UBQP approach is $O(KN)$. A standard procedure, denoted as *1-opt-Standard* hereafter, which compute the gain through the multiplication of $\mathbf{w}_{i.}$ and $\mathbf{H}$ would in fact have a complexity of $O(K^2N)$ as it needs to multiply $w_{i.}$, which is of size K, by $\mathbf{H}$ to find the gain of flipping one single cell ($O(KN)$ so far) and since this multiplications need to be done one time for each cell of $w_{i.}$, the total complexity will be $O(K^2N)$. Using the same reasoning, *1-opt-Standard* has a complexity of $O(K^2M)$

[1]This condition can be efficiently computed when checking the gains defined in Theorem 2.1.

Algorithm 2 *1-opt-BMF*

Input: Matrices **X**, **W** and **H**; $p = 1$ or 2

Output: Improved **W** and **H**

repeat
  for each row w_i in **W** **do**
    $j = \underset{1 \le j' \le k}{\operatorname{argmin}} \Delta E(i, j'; L_p)$
    if $\Delta E(i, j; L_p) < 0$ **then** flip W_{ij}
    end if
  end for
  for each column h^i in **H** **do**
    $j = \underset{1 \le j' \le k}{\operatorname{argmin}} \Delta E(i, j'; L_p)$
    if $\Delta E(i, j; L_p) < 0$ **then** flip H_{ji}
    end if
  end for
until no change in **W** and **H**

for updating one column of **H**. We show, via Theorem 2.2, that the *1-opt* procedure defined on the basis of Theorem 2.1 is more efficient. This result directly extends to *p-opt* local search as this latter is just a chain of *1-opt* procedures. Note that in Theorem 2.2, since we aim at finding the minimum gain that we are able to obtain using the proposed method, we consider updating a row of **W** in our calculations. The reason is that for *1-opt-UBQP* the complexity is the same for both case (updating one row of **W** and updating one column of **H**); however, since the complexity in the standard method is not the same for updating one row of **W** and updating one column of **H** ($O(K^2N)$ vs. $O(K^2M)$), we consider the faster one (K^2N) in order to obtain the *minimum gain* we can achieve with the proposed method. Consequently, Theorem 2.2 consider the case of updating one row of **W**.

Theorem 2.2 *We assume a row vector* $\mathbf{w}_{i.}$ *of* **W** *and a matrix* **H**. *Furthermore, let d be the density of* $x_{i.}$ *(the* i^{th} *row of* **X***) and let* τ *denote the proportion of columns l of* **H** *orthogonal to* $\mathbf{w}_{i.}$ *and such that* $x_{il} = 1$ *(thus* $\tau = |W_i^{\perp}|/N$*). Then, the gain in complexity for the 1-opt procedure based on Theorem 2.1 compared to both 1-opt-UBQP and 1-opt-Standard is at least:*

- $min\{(1-\tau)^{-1}, K\tau^{-1}\}$ *for the* L_2*-norm with respect to the 1-opt-UBQP;*
- $min\{K(1-\tau)^{-1}, K^2\tau^{-1}\}$ *for the* L_2*-norm with respect to 1-opt-Standard;*

- *$min\{K^2, K(d-\tau)^{-1}\}$ for the L_1-norm with respect to 1-opt-Standard.*

Proof First note that $0 \leq \tau \leq d \leq 1$ as the proportion of columns in $\mathbf{H}$ corresponding to 1s in $\mathbf{x}_{i.}$, *i.e.* d, is larger than the proportion of columns in $\mathbf{H}$ corresponding to 1s in $\mathbf{x}_{i.}$ and orthogonal to $\mathbf{w}_{i.}$, *i.e.* τ. From the definitions of d and τ, one has: $|W^0| = (1-d)N$, $|W^{\perp}| = \tau N$, $|W^{\not\perp}| = (d-\tau)N$. If we look at it in the matrix-wise way, we have the following matrices:

1. $\mathbf{W}_i^0$ which is of size $K \times (1-d)N$
2. $\mathbf{W}_i^{\perp}$ which is of size $K \times \tau N$
3. $\mathbf{W}_0^{\not\perp}$ which is of size $K \times (d-\tau)N$.

In order to study the minimum gain we obtain, we need to find the complexity of both L_1 and L_2 cases. Since we have two formulas for each of these methods (one for flipping from 0 to 1 and the other for flipping from 1 to 0), we need to compute, for each norm, the complexity of each formula and take the maximum as the complexity. For Equation 2.1, we have three parts in our complexity analysis which have been shown in Table 2.1. According to this table, if we sum all the complexities, we will

Table 2.1: Complexity analysis of Equation 2.1

Sub-equation	**Operations needed**	**Complexity**
$\sum_{\mathbf{h}\in W_i^0}(1+2<\mathbf{w}_{i.},\mathbf{h}>)h_j$	dot product between $w_{i.}$ and $\mathbf{W}_i^0$	$O\big(KN(1-d)\big)$
$\sum_{\mathbf{h}\in W_i^{\not\perp}}(2<\mathbf{w}_{i.},\mathbf{h}>-1)h_j$	dot product between $w_{i.}$ and $\mathbf{W}_i^{\not\perp}$	$O\big(KN(d-\tau)\big)$
$\sum_{\mathbf{h}\in W_i^{\perp}} h_j$	summation over a row of $W_i^{\perp}$	$O(\tau N)$

have the following total complexity for Equation 2.1:

$$O\big(KN(1-d)+KN(d-\tau)+\tau N\big) = O\big(max\{KN(1-\tau),\tau N\}\big) \tag{2.31}$$

For Equation 2.2, we have two main parts to consider in the complexity analysis. One should note that although the equation contains three parts, for two of them we need the same operation and, as a result, we can consider them as one single entity. Table 2.2 illustrates these entities: according to Table 2.2, the total complexity of Equation 2.2 amount to

$$O\big(KN(1-d)+KN(d-\tau)\big) = O\big(KN(1-\tau)\big) \tag{2.32}$$

Table 2.2: Complexity analysis of Equation 2.2

Sub-equation	**Operations needed**	**Complexity**
$\sum_{\mathbf{h}\in W_i^0}(1-2<\mathbf{w}_{i.},\mathbf{h}>)h_j$	dot product between $w_{i.}$ and $\mathbf{W}_i^0$	$O\big(KN(1-d)\big)$
$\sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j=1}} 1 \quad + \quad \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} 3-2<\mathbf{w}_{i.},\mathbf{h}>$	dot product between $w_{i.}$ and $\mathbf{W}_i^{\not\perp}$	$O\big(KN(d-\tau)\big)$

Therefor, according to (2.31) and (2.32), the proposed method in case of L_2 norm has a theoretical complexity of $O\big(max\{KN(1-\tau),\tau N\}\big)$. Now if we divide the complexity *1-opt-UBQP*, *i.e.* $O(KN)$, by this quantity, the minimum gain we obtain will be $min\{(1-\tau)^{-1},K\tau^{-1}\}$, and if we divide that by the complexity of the standard method, *i.e.* $O(K^2N)$, we have a minimum gain of $min\{K(1-\tau)^{-1},K^2\tau^{-1}\}$.

For the L_1 case, we have two equations: 2.3 and 2.4. We can consider Table 2.3 for Equation 2.3. Based on that, the total complexity of Equation 2.3 is:

Table 2.3: Complexity analysis of Equation 2.3

Sub-equation	**Operations needed**	**Complexity**
$\sum_{\mathbf{h}\in W_i^0} h_j$	summation over one row of $\mathbf{W}_i^0$	$O\big(N(1-d)\big)$
$\sum_{\mathbf{h}\in W_i^{\not\perp}} h_j$	summation over one row of $\mathbf{W}_i^{\not\perp}$	$O\big(N(d-\tau)\big)$
$\sum_{\mathbf{h}\in W_i^{\perp}} h_j$	summation over one row of $\mathbf{W}_i^{\perp}$	$O(\tau N)$

$$O\big(N(1-d)+N(d-\tau)+\tau N\big)=O(N) \tag{2.33}$$

For equation 2.4, we have basically two parts to study as shown in Table 2.4. Com-

Table 2.4: Complexity analysis of Equation 2.4

Sub-equation	**Operations needed**	**Complexity**
$-\sum_{\mathbf{h}\in W_i^0} h_j$	summation over one row of $\mathbf{W}_i^0$	$O\big(N(1-d)\big)$
$\sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j=1}} 1 - \sum_{\substack{\mathbf{h}\in W_i^{\not\perp} \\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}} 1$	dot product between $w_{i.}$ and $\mathbf{W}_i^{\not\perp}$	$O\big(KN(d-\tau)\big)$

paring (2.33) and complexities in Table 2.4, one can see that the total complexity of the proposed method in case of L_1 norm is $O\big(max\{N,KN(d-\tau)\}\big)$ as $N\geq N(1-d)$. Now if we divide the complexity of the standard method, $O(K^2N)$, by this quantity, the minimum gain we obtain will be $min\{K^2,K(d-\tau)^{-1}\}$ which establishes the theorem.□

One important remark here is that in the proposed method for L_2 norm, the condition $\tau N > KN(1-\tau)$ will almost never occur as it corresponds to values of τ greater than 0.50 and to matrices where $d \geq \tau > 0.50$, *i.e.* to extremely dense matrices. As a result, in practice, the complexity of the proposed L_2 method is $O\big(KN(1-\tau)\big)$ and, consequently, the gain will be $(1-\tau)^{-1}$ compared to *1-opt-UBQP* and $K(1-\tau)^{-1}$ compared to *1-opt-Standard*.

Another remark is that with the proposed method (Theorem 2.1), we do not need to perform all the dot products (the most computationally expensive operations in these formulations) for all cells of $w_{i.}$ as the dot products are done between $w_{i.}$ and the sets defined in Def. 2.1; none of these elements change while we are computing the gain for a given vector. Accordingly, one does not need to iterate over each cell which is the key point of efficiency of this method.

According to Theorem 2.2, one can note that the gain obtained by the proposed method is independent of K for the solutions of L_2-norm methods while there is a dependency between K and the gain obtained by the proposed method for the solutions of L_1-norm methods.

2.3 EXPERIMENTS

We have evaluated our local search heuristic on several methods used to solve the BMF problem and on several collections.

2.3.1 METHODS

We have retained here the most widely used methods to solve the BMF problem, all explained in detail in the previous chapter:

1. Projected NMF;
2. Projected SVD;
3. Proximus;
4. CBMF;
5. UBMF.

For projected NMF and SVD, we use the grid search, discussed in Chapter 1 and illustrated in Algorithm 1, with $\alpha = 0.05$ in order to find the best possible binary results (see *e.g.* [Miettinen et al., 2008]). To perform NMF and SVD, we used Matlab built-in functions and for the rest we used our own Matlab implementations.

As mentioned before, we refer to the method proposed in this chapter as *1-opt-BMF*, to the standard implementation as *1-opt-Standard* and to the *1-opt* local search associated with UBQP as *1-opt-UBQP*. One should note that the *1-opt-Standard* approach benefits from highly efficient block algorithms available in LAPACK (incorporated in Matlab) which are, in average, significantly faster than a simple, naive multiplication.

In addition, for CBMF, we also evaluate the swap heuristic described in [Kanungo et al., 2002] and discussed in Section 1.2.3 as well as a combination of this heuristic with *1-opt*. As mentioned in that section, the swap procedures, mainly applied to K-means-like methods, is to replace a center (after all centers become stable) with one data point and redo the clustering to see if the new solution is better than the previous one. Since CBMF is based on the K-means approach, we can apply the swap procedure which consists of randomly replacing a center (a column of $\mathbf{W}$) after stabilization with a random column of $\mathbf{X}$ and redo the $K-$means clustering (the CBMF decomposition in our case). This procedure is repeated a certain amount of times or until a time budget is exhausted. Obviously, one can still apply a *1-opt* heuristic on top of the swap technique in order to further improve the results.

The swap heuristic will be referred to as SCBMF (S stands for "swap") and the combination, which corresponds to the application of *1-opt* on the results of SCMBF, as *1-opt*-SCBMF. There are different criteria to choose the number of swaps in SCBMF. For example, one can select a fixed number of swaps or a time limit. To be fair in our comparison, we have adopted here the following strategy: for each dataset we first run the proposed *1-opt* algorithm (which is in general faster than the other approaches as we will observe shortly) on CBMF and measure its running time; then, we let the SCBMF procedure run the same amount of time to improve CBMF results. Like that, one can observe which method performs best under a given time budget.

The main difference between SCBMF and *1-opt* local search procedures is that the former tries to find better solutions by exploring different parts of the search space whereas the latter tries to find a better solution in a close neighborhood of the current solution. Observing the respective behaviors of these different approaches is thus interesting.

Lastly, as SCBMF proceeds via random selections, for each dataset, we run the SCBMF ten times and report the average error of these ten runs. In this case, a difference is deemed statistically significant if it was significant on the majority of the 10 runs. We use here the Wilcoxon sum-rank test at the 1% significance level to assess whether differences in the results are significant or not.

2.3.2 General Settings

In our experiments, we let the rank K vary in the set {1, 20, 40, 60, 80, 100}. As mentioned before, the BMF problem can be solved with or without the binary reconstruction constraint, with the L_1-norm or with the L_2-norm. Accordingly, four classes of decomposition methods can be considered (Table 2.5). All the experiments are

Table 2.5: Different problem classes

Approach/Norm	L_1	L_2
Constrained	CBMF	PROXIMUS
Unconstrained	UBMF	NMF, SVD

done on a Linux machine with an Intel Xeon CPU E5-2630 with 6 cores @ 2.30Ghz and 32Gb of memory. In order to be fair, unless it is explicitly mentioned, no multiprocessing or multithreading is done in our experiments.

2.3.3 Datasets

We examined both real world datasets and synthetic ones, all available from the frequent itemset mining repository (FIMI)[2]. The main reason that we use frequent itemset mining datasets is that they are necessarily binary which suits our experimental setting. Table 2.6 shows the sets we used in addition to some of their characteristics. The first part of the table shows the real datasets and the second part shows the synthetic ones. Note that in our experiments, we have removed all empty rows and empty columns from the data.

[2]Publicly available at `http://fimi.ua.ac.be/data/` (last visited 03-Jun-2015)

Table 2.6: Datasets used in the experiments.

Name	# Rows	# Columns	Density (%)
Mushroom	8124	119	19.32
Connect	67557	129	33.33
Accidents	340183	468	7.22
Pumsb	49046	2113	3.50
T10I4D100K	100000	870	1.16
T40I10D100K	100000	942	4.20

2.3.4 TIME COMPARISON

We compare here *1-opt-BMF* with *1-opt-Standard* and *1-opt-UBQP* according to the running time. One should note that these three approaches yield the same solution, namely the one corresponding to the best improvement in a neighborhood of size 1 and thus, their only difference is the running time. Note that *1-opt-BMF* and *1-opt-Standard* can be applied to both L_1 and L_2 decomposition methods, whereas *1-opt-UBQP* can only be applied on the L_2 decomposition approaches (NMF, SVD and *Proximus*). As a result, for L_2 methods, we compare *1-opt-BMF* with both *1-opt-Standard* and *1-opt-UBQP* while for L_1 methods we can only perform one comparison: *1-opt-BMF* with *1-opt-Standard*.

To further illustrate the speed of the different methods, we display in Figure 2.1-2.4 the **ratio**: execution time of *1-opt-Standard* divided by execution time of *1-opt-BMF* and execution time of *1-opt-UBQP* divided by execution time of *1-opt-BMF*. An additional line, labeled "Ratio=1", is added to make it simpler to compare different methods: *1-opt-BMF* is faster when the curve is above 1, and slower otherwise. In the following, we will discuss these results with respect to different datasets.

2.3.4.1 Projected SVD and NMF

Figure 2.1 illustrates the time comparison between *1-opt-UBQP* and *1-opt-BMF* on the left as well as the comparison between *1-opt-Standard* and *1-opt-BMF* on the right. The first issue to note is that, generally speaking, *1-opt-UBQP* is faster than *1-opt-Standard* as in the latter the ratio reaches up to 4.7 while in the former it reaches to 1.8 in the best case.

Another point on this figure is that by increasing the value of K the ratio increases

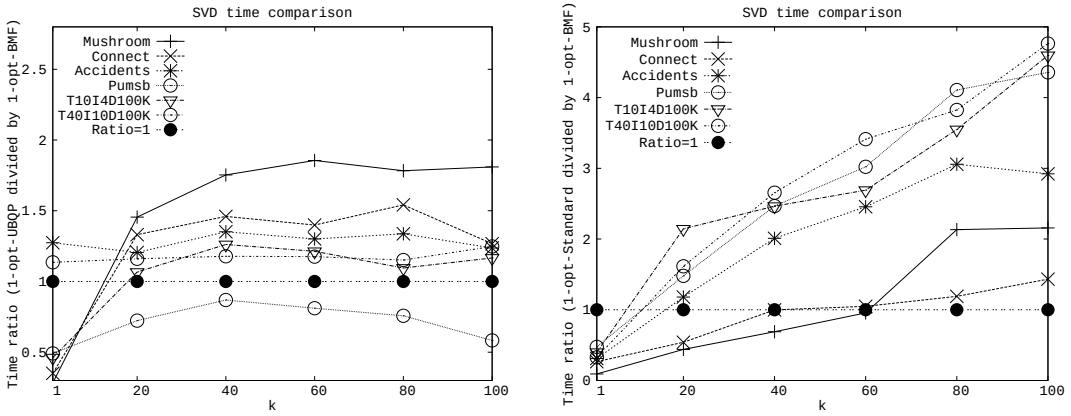

Figure 2.1: Efficiency of *1-opt-BMF* compared to *1-opt-UBQP* (left) and *1-opt-Standard* (right) for SVD

significantly with respect to the standard method. In case of UBQP, the curves become almost stable for larger values of K. As discussed in the Theorem 2.2, the gain of *1-opt-Standard* is directly influenced by K, unlike the *1-opt-UBQP*.

As one can see on the right hand side of this figure, the curves are under the fixed line (ratio=1) mostly for `Mushroom` and `Connect`; these two sets are the smallest sets in our experiments. As the *1-opt-BMF* has an overhead of finding the sets defined in Def. 2.1, for small datasets this overhead is more *visible* with respect to the update part. However, even if these sets, for $K \geq 60$, one can observe that the *1-opt-BMF* becomes faster. For sufficiently large datasets (as `Accidents` or `T40I10D100K`) and sufficiently large values of K (*e.g.* $K = 100$), *1-opt-BMF* can be nearly 5 times faster than the standard approach.

On the left hand side, we only see one dataset (`Pumsb`) being always below the fixed line. The main reason here is that this dataset is not only very sparse ($d = 3.5\%$), but also has a small value of τ ($0.02 \leq \tau \leq 0.03$ for $K = 20$ for instance) which makes $1-\tau$ very close to 1. According to Theorem 2.2, a small value of $1-\tau$ results in a small gain, and since we have an overhead for computing the sets defined in Def. 2.1, the *1-opt-BMF* cannot be more efficient than the *1-opt-UBQP* in this case. However, in the other datasets, we observe the efficiency of the *1-opt-BMF* with respect to *1-opt-UBQP*.

In Figure 2.2, one can observe the experiments using NMF. In that figure, we can see the same points that we saw for the SVD case. We can also see the same issue for `Pumsb` when the *1-opt-UBQP* is concerned. Additionally, the *semi-constant* line for ratios can also be seen in this case after a certain value of K (the figure on the left side) while in the *1-opt-standard* case (the figure on the right side) the ratio grows as

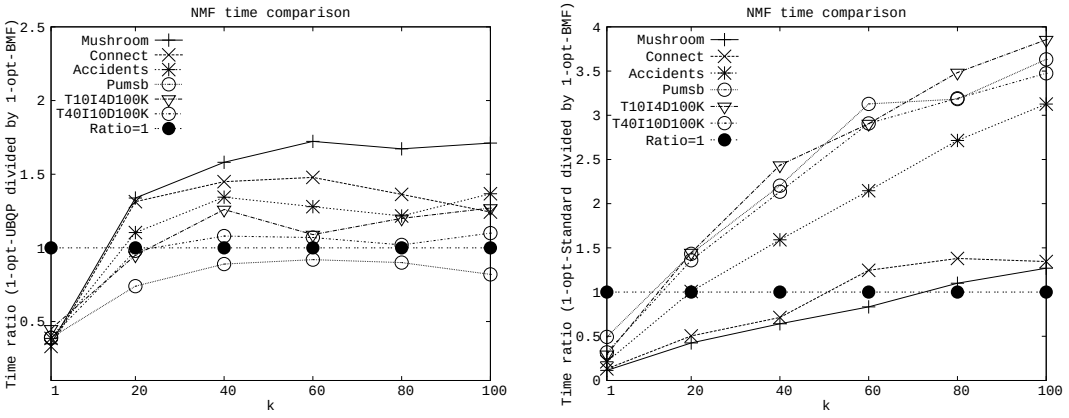

Figure 2.2: Efficiency of *1-opt-BMF* compared to *1-opt-UBQP* (left) and *1-opt-Standard* (right) for NMF

the value of K increases. This is in line with theoretical analysis provided in Theorem 2.2.

2.3.4.2 Proximus

First of all, one should note that in Proximus, one may end up in less number of latent factors than what s/he had in mind initially. The reason is that, according to the way Proximus solves the problem, in each iteration, a set of columns are eliminated (the *covered* columns are eliminated in order to ensure the orthogonality which, in turn, ensures the binary reconstruction). This may result in early termination of the algorithm if we run out of columns before reaching the desired value of K. This case has happened for two datasets in our experiments: `Mushroom` and `Connect`. Since they have few columns and they are very dense, the columns are covered very quickly. As a result, we only have 53 and 54 latent factors in `Mushroom` and `Connect` respectively. Consequently, in Figure 2.3, for $K \geq 60$, we report the same results corresponding to the maximum number of factors obtained.

As one can observe in Figure 2.3, for the *1-opt-Standard*, we see similar behavior as the other L_2-norm methods (SVD and NMF). The situation with respect to *1-opt-UBQP* is also similar to those of SVD and NMF. When applied to *Proximus*, *1-opt-BMF* is faster on 5 collections out of 6, *i.e.* all except `Pumsb` for the same reason as mentioned before. Here again we see that *1-opt-BMF* gets significantly faster than *1-opt-Standard* as the value of K increases.

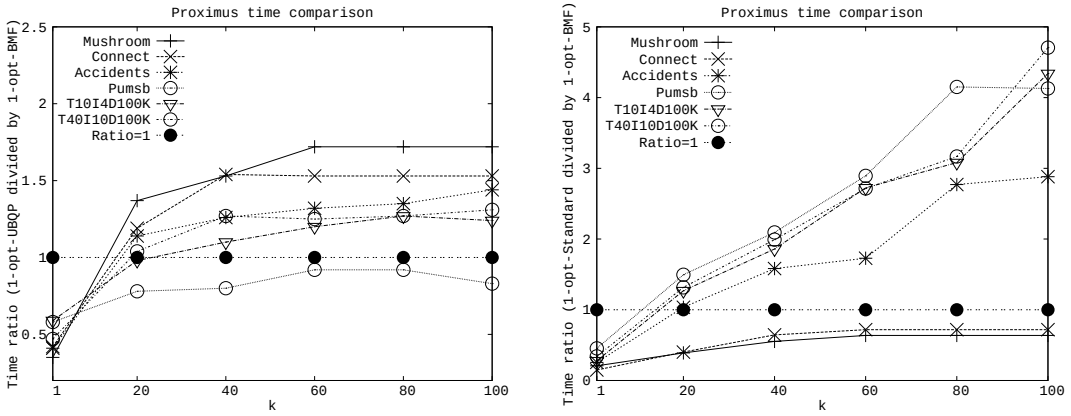

Figure 2.3: Efficiency of *1-opt-BMF* compared to *1-opt-UBQP* (left) and *1-opt-Standard* (right) for Proximus

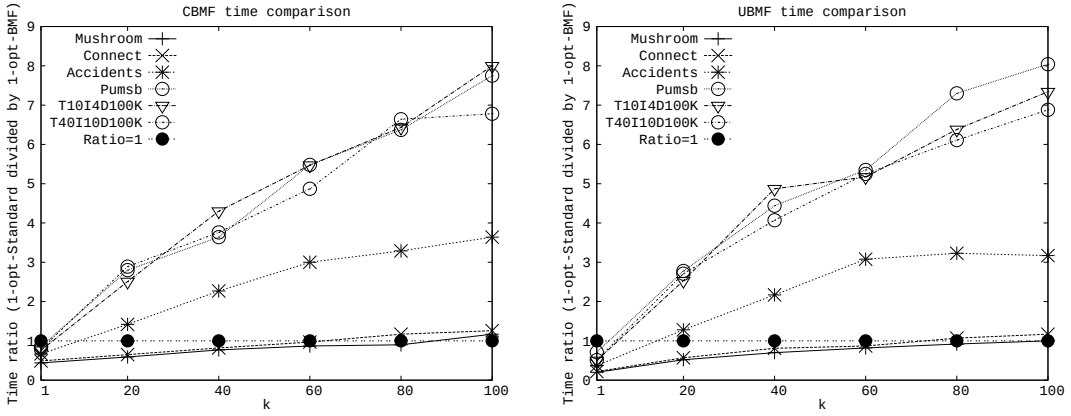

Figure 2.4: Efficiency of *1-opt-BMF* compared to *1-opt-Standard* for CBMF (left) and UBMF (right)

2.3.4.3 CBMF and UBMF

For the L_1-norm methods, we can only apply the *1-opt-Standard* (Figure 2.4). Here, as we have seen before, except the smallest sets (`Mushroom` and `Connect`), the *1-opt-BMF* is up to 8 times faster than the standard method where this efficiency is more considerable with larger values of K which is in line with the theoretical analysis provided before.

All in all however, *1-opt-BMF* is in general faster than *1-opt-Standard* and *1-opt-UBQP* where the difference is much higher for the former than for the latter. Furthermore, unlike *1-opt-UBQP*, *1-opt-BMF* can be applied to all the BMF methods we know of.

2.3.5 IMPACT OF *1-opt-BMF* ON L_2 METHODS

As discussed before, all the previously mentioned methods yield the same solution. Here, we would like to examine the significance of the improvement brought by a *1-opt* method. Figure 2.5 shows the effectiveness of any *1-opt* strategy when it is applied to (projected) SVD, (projected) NMF and *Proximus*. As one can note, projected NMF performs generally better than projected SVD, with a reconstruction error (measured with the L_2-norm) significantly lower. Projected NMF also yields a lower reconstruction error than *Proximus*, but *Proximus* provides a solution that satisfies the binary reconstruction constraint, which is not the case for NMF.

To assess whether the improvements obtained with *1-opt* are significant, we computed the *p-value* of the Wilcoxon rank-sum test (which is widely used to compare a series of bits) at the 1% significance level. In almost all L_2-norm cases with $K = 1$ there is no improvement with *1-opt*. This can be explained by the fact that the first dominant factor in the datasets is more easily approximated by the different methods than the first K factors. This observation also explains why the gain with *Proximus*, which solves a series of rank 1 problems, is not as important as with the other methods. For $K > 1$ however, in all sets and with all methods, the improvement obtained with the *1-opt* is statistically significant. In particular, for NMF and SVD, the improvement can be up to 61% for real sets, and up to 12 % for synthetic sets.

2.3.6 IMPACT OF *1-opt-BMF* ON L_1 METHODS

Table 2.7 illustrates the improvement that *1-opt* can make over the L_1 methods, namely CBMF, SCBMF and UBMF. As mentioned before, SCBMF is provided the same amount of time as *1-opt-BMF*. As one can note, the case of $K = 1$ is removed from the table since, as mentioned above, no local improvement can be made for the first factor.

The table shows the normalized errors in percentage. Significant improvements are shown in bold (measured again using Wilcoxon rank-sum test at the 1% significance level). Note that for CBMF (resp. UBMF), the *1-opt* error is shown in bold if it is significantly better than the standard CBMF (resp. UBMF). SCBMF's error is shown in bold if it is significantly better than standard CBMF. For 1-opt-SCBMF, the error is in bold if it is significantly better than SCBMF.

The first point to note is that all proposed heuristics have difficulties improving the error value over synthetic datasets. In those datasets, which do not contain any

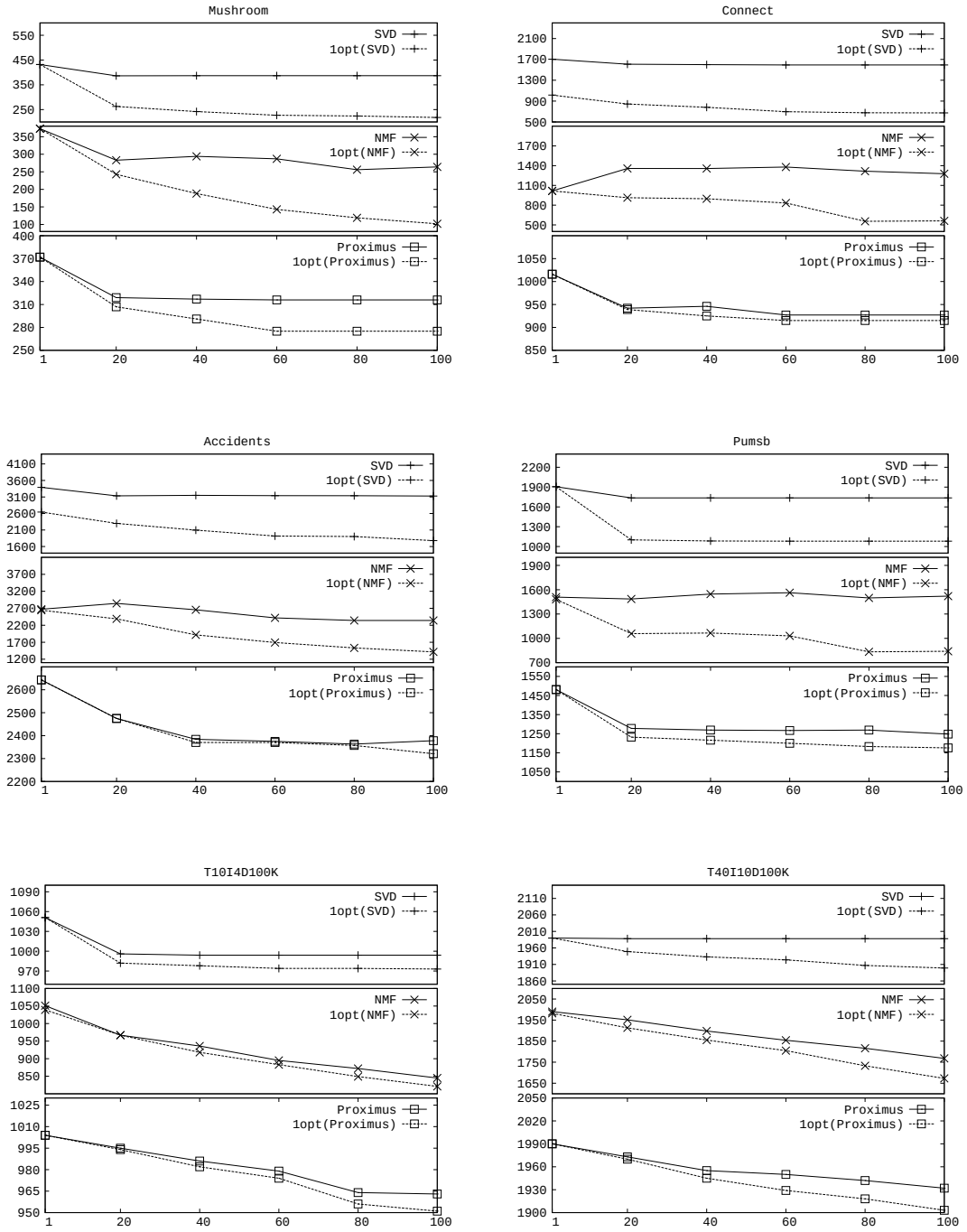

Figure 2.5: Improvement of L_2-norm with the *1-opt* strategy applied on SVD, NMF and *Proximus*. The horizontal axis represents the value of K and the vertical axis shows the L_2-norm.

structure, all the methods yield roughly the same results.

Over all real datasets, a *1-opt* local search can often bring significant improvement in the error value. The most spectacular case is the `Mushroom` dataset, where for $K = 80$ the error value is decreased by more than 18%. This dataset has a large

Table 2.7: Effectiveness of the *1-opt* on L_1 methods. Numbers are the normalized error in percentage. Statistically significant improvements are shown in boldface.

Dataset	k	CBMF		SCBMF	*1-opt*-SCBMF	UBMF	
		Standard	*1-opt*			Standard	*1-opt*
Mushroom	20	7.5032	**7.2662**	7.0481	**6.7332**	6.8685	6.8685
	40	4.9601	**4.4301**	**4.1346**	**3.6278**	4.4537	4.4537
	60	2.5769	**2.2591**	2.4711	**2.1682**	2.1958	2.1958
	80	1.4323	**1.1655**	1.3449	**1.1608**	1.0042	1.0042
	100	0.1917	0.1313	0.1835	**0.1500**	0.1377	0.1377
Connect	20	7.9585	**7.8659**	**7.4513**	**7.3004**	7.3202	7.3202
	40	4.9624	**4.9077**	**4.5270**	**4.4728**	4.1145	4.1145
	60	2.6073	2.5850	**2.4071**	2.3849	1.7336	1.7336
	80	1.7881	**1.6420**	**1.3584**	**1.3551**	0.8995	0.8995
	100	0.3537	0.3537	0.3537	0.3537	0.3537	**0.3321**
Accidents	20	3.1373	3.1373	3.1373	3.1373	2.9923	2.9923
	40	2.7191	2.7191	2.7191	2.7191	2.4352	2.4352
	60	2.1906	**2.1493**	2.1900	**2.1495**	1.9766	1.9766
	80	1.6426	**1.6360**	**1.6321**	**1.6255**	1.4654	1.4653
	100	1.1345	**1.1014**	1.1342	**1.1018**	0.9504	0.9504
Pumsb	20	1.3253	**1.2895**	1.3214	**1.2782**	1.2177	1.2176
	40	0.9608	**0.9310**	0.9608	**0.9311**	0.8903	0.8903
	60	0.8719	**0.8647**	0.8719	**0.8647**	0.8201	0.8200
	80	0.8763	**0.8467**	0.8740	**0.8491**	0.7638	0.7637
	100	0.7560	**0.7470**	0.7560	**0.7470**	0.7010	0.7010
T10I4D100K	20	1.1158	1.1158	**1.1148**	1.1148	1.1158	**1.1097**
	40	1.0745	1.0745	1.0745	1.0745	1.0734	1.0734
	60	1.0432	1.0432	**1.0413**	1.0413	1.0403	1.0403
	80	0.9806	0.9804	0.9803	0.9802	0.9757	0.9757
	100	0.9441	0.9435	0.9438	0.9432	0.9321	0.9321
T40I10D100K	20	4.0715	4.0715	4.0715	4.0715	4.0715	**4.0563**
	40	3.9340	3.9340	3.9340	3.9340	3.9321	3.9321
	60	3.8203	3.8203	**3.8169**	3.8169	3.7926	3.7926
	80	3.6850	3.6850	3.6850	3.6850	3.6382	3.6382
	100	3.5602	**3.5596**	**3.5560**	3.5555	3.5062	3.5062

number of patterns spanning few lines [Uno et al., 2005], making it more difficult for BMF algorithms to compute a good approximation for high values of K. These results show that although a *1-opt* heuristic explores a small neighborhood of the current solution, it can help improve the decomposition on such difficult cases.

`Connect` is a dense dataset with a strong structure [Uno et al., 2005]: this structure is mostly captured by classical CBMF for small values of K, with only small improvement by *1-opt* (1%). Here the improvement of a local search only pays off for $K = 80$ (8%). On the other hand, SCBMF has significantly better results (from 6% to 24% improvement in error value) than CBMF: this means that CBMF was stuck in a region that was not optimal. The swap heuristic of SCBMF is the only one of this comparison that can get out of this local optima and find a better solution. SCBMF's solutions can be further refined by applying the *1-opt* heuristic, showing the ability of *1-opt* to improve an already good solution whatever its position in the search space.

`Accidents` is a different type of dataset: despite its huge size it has relatively short transactions (length 33 in average) and contains many patterns differing on only a few columns, a structure difficult to capture with low values of K. CBMF captures well the bulk of this structure (as shown by the lack of improvement from SCBMF) and its solution is hard to improve for low K values (no improvement by *1-opt*). However for higher values of K, the potential of the *1-opt* local search heuristic appears, as it provides a significant improvement for $K \geq 60$.

`Pumsb` is the sparsest real dataset of this experiment. This sparsity, combined with the higher number of columns, prevents SCBMF to find a better solution in the allocated time. However, the dataset (originating from US Census data) has some structure, and here also the *1-opt* method can significantly improve the solution found by CBMF (at best 3% for $K = 40$).

In unconstrained case, i.e. UBMF, it becomes much harder to improve the results. Nevertheless, there are still a few cases where the *1-opt* approach can provide a significantly better approximation.

2.4 CLOSING REMARKS

We have addressed in this chapter the problem of rank K Binary Matrix Factorization (BMF) which aims at approximating a binary matrix by the product of two binary matrices of lower rank. Several researchers have addressed this problem, focusing on either approximations of rank 1 or higher, using either the L_1 or L_2-norms for measuring the quality of the approximation, through variants of the alternating strategy typically used for non-negative matrix factorization [Lee and Seung, 2000].

We first reviewed several local search strategies that can be used to improve the

BMF solutions obtained by previously proposed methods. Then we introduced, as our second contribution [Mirisaee et al., 2015b,Mirisaee et al., 2015c], a new local search dedicated to the BMF problem and studied its complexity with respect to other local search approaches.

The experiments, conducted with several state-of-the-art methods, on several collections, have confirmed that the *1-opt* local search procedure proposed in this chapter is in general faster than the previously proposed ones. They have also shown that this procedure significantly improves the solutions found by the state-of-the-art L_2-norm methods: projected NMF, projected SVD and *Proximus*.

For the L_1-norm methods, the experimental results show that the proposed local search procedure improves the results of the state-of-the-art methods solving the complete BMF problem, with the binary reconstruction constraint, on the real datasets, which are more structured than the synthetic ones. These methods include CBMF and its "swapped" version. The situation is more contrasted with UBMF, which solves the BMF problem without the binary reconstruction constraint. *1-opt* local search heuristic do not seem to be able to improve this state-of-the-art method in many cases.

3 Application to the mining of representative frequent itemsets

3.1 Introduction

In previous chapters, we have mentioned that matrix decomposition techniques have been widely used in data mining applications as they are able to capture the main structure of the data and provide us with latent factors and important pattern found within the data. Then we focused on one particular case of matrix decomposition, namely Binary Matrix Factorization (BMF), and showed its relation to other problems. In addition, we discussed the local search heuristics for BMF and proposed a new heuristic which can improve the solutions significantly faster than other state-of-the-art techniques. An extensive set of experiments was conducted and different datasets have been considered to show the efficiency of the proposed method.

In this chapter, we focus on frequent itemset mining (one sub-problem of a more general domain called *pattern mining*), a problem which has been of interest of many researchers and industries [Han et al., 2007]. In particular, we aim at finding a set of representative itemsets which can not only provide a good view of the entire frequent itemsets, but also is a lot less numerous, hence the term *representative*. As mentioned previously, matrix decomposition techniques can capture the important and latent information form datasets; accordingly, in this chapter, we propose to rely on this method to find representative itemsets. As in itemset mining problem the input is a binary matrix, we can simply use the BMF technique discussed in the first two chapters. In the following, we will first introduce the general form of itemset mining problem and then discuss the importance of *representative* frequent itemsets.

Frequent pattern mining is a major domain of pattern mining. Its goal is to automatically extract frequently occurring *patterns* in large datasets. The fundamental

problem of frequent pattern mining is to mine frequent *itemset* [Agrawal and Srikant, 1994], i.e. sets of items frequently occurring together in transactional databases (defined formally in the following section). This problem originated from the analysis of commercial databases and was later extended to many domains.

To better understand the importance of the frequent itemset mining problem, consider the following scenario: suppose that Tom runs a supermarket in Michigan, in which he makes the decisions for the arrangement of products. After a while he realizes that some products have been bought together quite frequently, like wine and cheese or bread and butter. In such case, depending on the marketing strategy, he may either rearrange the products in the store such that those items are located close to each other (so that customers find them quickly) or far from each other (so that customers travel all the way in the store and buy more products on the way). Accordingly, it is very important for him to find the products bought together frequently.

The frequent itemset mining technique is used in many applications today such as chemistry [Inokuchi et al., 2000], bioinformatics [Wang et al., 2005], trace mining [Zou et al., 2010], web data analysis [Han et al., 2000] and gene expression data [Alves et al., 2010]. Frequent pattern mining algorithms have to explore a huge combinatorial space of patterns. This leads to two main problems: first, on real databases, mining algorithms usually need a large amount of time to output the set of frequent patterns. This hinders interactive analysis of data, that is however needed in most knowledge discovery tasks. Second, the set of frequent patterns is often huge: millions of frequent patterns are commonplace. This is overwhelming for the analyst, and it further prevents from having real insights on the data that appear in some frequent patterns.

There has been a large body of research on condensed representations of frequent patterns such as *closed* frequent patterns [Pasquier et al., 1999, **?**], that can usually output one order of magnitude less patterns without loosing information. However, one order of magnitude reduction is often not enough when dealing with millions of patterns, hence a current trend of research focuses on discovering a small set of patterns that would represent most of the information contained in the set of all frequent patterns, and that would be small enough to be managed by the analyst [Vreeken et al., 2011, Guns et al., 2010].

In this chapter, we present our third contribution [Mirisaee et al., 2014] by introducing a new method for computing a reduced set of representative frequent patterns, that is both computationally efficient and produces very few high quality frequent pat-

terns. This method is based on constrained binary matrix factorization which has been widely used in many contexts recently and has been explained previously. As mentioned in Chapter 1, the general goal of matrix decomposition is to factorize a matrix into the product of two or more smaller matrices. The Constrained Binary Matrix Factorization (CBMF) problem, as a special case of matrix decomposition, was described previously and an efficient local heuristic for that has been introduced (Chapter 2). In this chapter, we use the materials and techniques provided in Chapter 2 in order to mine *representative* frequent itemsets. Exploiting this tool, the detailed contributions of this chapter are the following:

- We theoretically study the characteristics of a matrix decomposition method adapted to frequent itemset mining, and define the notions of *strong decomposition* and *approximately strong decomposition*.
- We prove that extracting frequent itemsets from the factors of the decomposition (which are matrices of reduced dimensionality) guarantees to find existing frequent itemsets with a given support threshold and size.
- We show, through a detailed experimental study, that the *representative* frequent itemsets output by this method represent a significant portion of the set of all frequent itemsets according to existing metrics [Gupta et al., 2000], while being up to nine orders of magnitude less numerous.

In the reminder of this chapter, we first explain formally the problem of frequent itemset mining followed by a review of the state-of-the-art and the major studies conducted in this domain. Then we introduce a framework which provides the theoretical requirements for linking the matrix decomposition problem to frequent itemset mining. Next, we design a set of experiments in which we use some evaluation metrics from the literature in order to examine the quality of the obtained representative itemsets. Finally, some closing remarks are discussed in the last section.

3.2 FREQUENT ITEMSET MINING

Let I be a set of items. An **itemset** is a subset of I. A transaction database (TDB) is a set of **transactions** D where each transaction $t \in D$ is an itemset of I. We will denote by m the size of a transaction database, i.e. its number of transactions. For an itemset

$P \subseteq I$ and a transaction database D, the **support** of P in D, denoted by $support(P,D)$, is the number of transactions of D including P:

$$support(P,D) = |\{t \mid t \in D \wedge P \subseteq t\}|$$

The **frequency** of P is

$$frequency(P,D) = \frac{support(P,D)}{m}$$

Furthermore, P is **frequent** if its support is greater than a given minimum support threshold $minsup$:

$$frequency(P,D) \geq minsup$$

The following example illustrates these definitions.

Example 3.1 *Table 3.1 shows a transactional database in which* tid *denotes the transaction identification (or simply its ID). In this example, {cheesr, wine, chips} is an itemset of size three (since there are three items in the set) and of support value two since it has appeared in two transactions: the first and the last transactions (*t_1 *and* t_5*). Since this TDB contains five transactions, the frequency of this itemset is* $\frac{2}{5}$.

Table 3.1: A Transactional DataBase (TDB)

tid	**contents**
1	{cheese, wine, chips, bread}
2	{bread, butter, beer, nuts}
3	{cheese, bread, orange, soap}
4	{chips, bread, beer, nuts, orange}
5	{cheese, wine, chips}

3.3 STATE OF THE ART

In this section, we first consider the major studies on frequent itemset mining followed by the studies in the recent domain of finding an informative subset of the frequent patterns.

Frequent itemset mining and association rule learning could be seen frequently together in the literature. Association rule learning is defined as a set of techniques which aim at finding interesting relations in big datasets. These rules could be found in the same way as clustering rules [Witten et al., 2011]. Association rule mining is about finding some implications within the data. For instance, if a client buys bread and cheese, how likely is that s/he buys also some chips? Looking at this problem, one can easily see that it is very similar to the frequent itemset mining problem. Consider the following example:

Example 3.2 *Having the TDB shown in Table 3.1, we are interested in the following phenomenon: if a client buys cheese, how likely is it that s/he buys also wine? We denote it as* $cheese \Rightarrow wine$*. To examine that, we need two elements: the support of the itemset {cheese, wine} which is, in this case, called the support of the association rule and also the confidence of this rule. As the name explains, the confidence of a rule tells us how likely it is to have such a rule. As a result, the confidence of this rule is the support of the {cheese, wine} (the left hand side and the right hand side of the rule) divided by the support of {cheese} (the left hand side). As one can observe, to obtain the association rules, we need to compute the frequent itemsets.*

Perhaps one of the most known studies in this domain has been conducted by Agrawal and Srikant [Agrawal and Srikant, 1994]. This study provides a fast algorithm for solving the association rule mining which is from one to three orders of magnitude faster than the state-of-the-art algorithms of the time. Their algorithm uses a seed set in order to generate new potentially large (frequent) itemsets called candidate itemsets and then counts actual support of these candidate sets. In each step, sufficiently large itemsets become seeds for the next step. These algorithms are called Apriori and AprioriTid and make use of anti-monotony property. The level-wise process used in these algorithms avoids considering the transactions in the database.

Few years later, Agrawal et. al. [Agrawal and Shafer, 1996] considered a set of parallel algorithms for association rule mining and analyzed the trade-off between computation, memory usage, synchronization and the use of problem-specific information. They provided a parallel algorithm to tackle the problem of finding all frequent itemsets. Then a second parallel algorithm handles the problem of generating the association rules. Based on this framework, the authors suggest three algorithms: *count distribution*, *data distribution* and *candidate distribution*. The first one tries to

avoid communication between processors. The second one is based not on the exchange of data tuples between processes but on their counts so that processors can work independently while reading the data. As both of these approaches need process synchronization at the end of each pass (which may waste processors' time if the workload is not well balanced), the third algorithm tries to tackle this issue with minimizing the dependency between processes.

Another successful study in this domain is [Bayardo Jr, 1998] in which the authors present an algorithm which scales linearly in the number of maximal patterns, no matter how long the patterns are. This algorithm, called *Max-Miner*, uses the set enumeration tree search presented in [Rymon, 1992] in order to expand sets over a finite item domain. The proposed algorithm in that study provides a pruning system which is not only based on subset infrequency (like Apriori algorithm) but also based on superset infrequency. Using that, Max-Miner is able to reduce the search space and perform the mining task several orders of magnitude faster with respect to the Apriori method.

Han et. al. introduced, in [Han et al., 2000], a new approach for mining frequent itemsets, which avoids the generate-and-test approach used in Apriori-like algorithms. This method is called *FP-growth* and is based on a frequent pattern tree (FP-tree) structure in order to store compressed information about frequent patterns. The efficiency of this approach is based on three points: compressing the database (to avoid database scans), pattern fragment growth (to avoid candidate generation) and a divide-and-conquer method to decompose the mining task. Focusing on two bottlenecks of the Apriori method, *i.e.* huge number of candidate sets and repeated database scans, FP-growth provides a better way of mining frequent itemsets which is about one order of magnitude faster than the Apriori method and scales for both short and long frequent patterns.

So far, we have seen some major studies in frequent itemset mining domain. However, classical methods, which mine all frequent patterns, produce a large number of frequent itemsets. This makes the analysis task very difficult, if not impossible. Having millions or billions of frequent patterns may not be ideal for a market analyst who needs to decide about the future decisions of a company. As a result, a new trend of research became of interest very quickly: how to find fewer itemsets which are not only frequent but also informative and non-redundant. Like that, the analyst, for example, can simply study the data without being overwhelmed with a tremendous amount of patterns. Consequently, we will first review the major studies in the recent

domain of finding an informative subset of the frequent patterns before providing the new BMF-based representative itemset mining procedure.

One of the most known ways to provide a reduced number of itemsets without loosing information is to mine *closed* itemsets [Pasquier et al., 1999,**?**]. Intuitively, an itemset P is closed if non of its proper supersets has the same frequency as P. One of the first algorithms proposed to handle this problem is developed in [Pei et al., 2000] and is called *CLOSET*. This algorithm, which is based on *FP-tree*, finds closed itemsets by developing a single prefix path compression method and exploring a partition-based projection mechanism. The authors then illustrated the efficiency and speed of the algorithm by comparing it to other closed itemset mining algorithms.

LCM (Linear time Close itemset Miner) is another known algorithm in the literature of itemset mining which is basically based on prefix preserving closure extension [Uno et al., 2005,Uno et al., 2004,Bayardo Jr, 1998]. Initially, LCM did not have a procedure to reduce the database and, as a result, was not efficient in dense datasets; however, the reduction techniques then added to the algorithm on the later versions in order to perform a fast checking on closedness of the itemsets [Uno et al., 2005]. Experiments on LCM show that it is significantly faster than other methods, particularly when sparse datasets are evaluated. In the latest version of LCM [Uno et al., 2005], three types of data structures used in the domain (bitmap, prefix tree and array lists) are combined in order to provide the best performance.

In order to output a small set of frequent patterns, some studies explicitly define a targeted number of patterns, and criteria that have to be verified by the set of patterns. Guns et al. [Guns et al., 2010], for example, present an exhaustive approach, where the criteria to be satisfied are expressed through different constraints. The main idea in this study is to produce a set of k related patterns instead of mining all individual patterns. A number of constraints have been designed in order to find such set at the global level rather than the local level. Using constraint programming techniques, an exhaustive search is performed which, according to the authors, has many limitations since it might be computationally very expensive. A family of constraints has been introduced in order to find the k-pattern set including individual pattern constraints (based on [Fürnkranz and Flach, 2005]), redundancy constraints, discriminative constraints and a set of different coverage constraints.

Other approaches, such as [Bringmann and Zimmermann, 2007], make use of redundancy-based heuristics to determine the set of patterns. The algorithm proposed in [Bringmann and Zimmermann, 2007] iterated over the set of patterns and, in each

iteration, a pattern is evaluated for its level of *goodness*. This technique is considered as a post-processing which is performed on the set of frequent itemsets.

Other studies rely on compression, either over all the frequent patterns [Afrati et al., 2004], or over the dataset [Vreeken et al., 2011]. In the first case, frequent itemsets that best approximate the set of all frequent itemsets are computed. This technique requires a pre-generated set of frequent itemsets and, using that as the input, an approximated collection of frequent patterns can be generated. In the second case, the authors exploit the Minimal Description Length (MDL) principle to compute which set of patterns best compresses the database. The method studied in [Vreeken et al., 2011] is called KRIMP and is based on a code table: a table which maps each itemset to a prefix code which is determined based on the usage of the itemset. The idea of the coding is that more frequent itemsets have shorter lengths since the concern here is to have the best compression. The KRIMP algorithm starts with a standard code table based on all singletons (itemsets of size one), and then adds one frequent itemset at the time (mined beforehand and ordered with respect to a procedure based on size and support) and recomputes the code table. If the new code table provides a better compression, then the itemset is kept in the set, discarded otherwise.

All of the previously cited studies consider the set of all frequent patterns in order to compute the reduced subset of patterns. The approach in [Vreeken et al., 2011] is a two pass approach, that first has to compute all the frequent patterns and then select a subset of them, which is computationally inefficient. On the other hand, the approach in [Guns et al., 2010] computes, in one pass, a subset of frequent patterns that optimize a given criterion. However, to do so, it has to explore a complex search space over the sets of patterns, which is also computationally expensive.

The approach presented here, however, is different from the previous studies in that we rely on main components of the data provided by matrix decomposition techniques in order to find representative frequent itemsets. One should note that the proposed approach is also different form studies like [Krajca et al., 2011] which use the Boolean decomposition and closed itemsets to reduce the dimension. Contrary to other pattern set discovery methods, by moving the burden of reducing the number of patterns from the pattern search space to the input matrix itself, one can design mining algorithms that are computationally efficient and that yield good results.

3.4 THEORETICAL ANALYSIS

In order to provide an efficient BMF-based method for the frequent itemset mining task, we first need to establish the theoretical link between these problems, and to do that, we first define two kinds of decompositions: *strong decomposition* and *approximately strong decomposition*. Then, based on these definitions, we show how frequent itemsets can be mined using matrix decomposition factors.

3.4.1 NOTATIONS

We first define the matrix notations used in this chapter. For a given matrix $\mathbf{X}_{m\times n}$, $\mathbf{X}_{ij}$ denotes the element of the matrix located in row i and column j. $\mathbf{X}_{i\bullet}$ denotes the summation of row i, *i.e.*

$$\mathbf{X}_{i\bullet} = \sum_{k=1}^{n} \mathbf{X}_{ik}$$

$\mathbf{X}^{\bullet j}$ denotes the summation of the j^{th} column of $\mathbf{X}$ *i.e.*

$$\mathbf{X}^{\bullet j} = \sum_{k=1}^{m} \mathbf{X}_{kj}$$

The row vector taken from column-wise summation of $\mathbf{X}$ is denoted by $\mathbf{X}^{\bullet}$. Similarly, the column vector obtained from row-wise summation of $\mathbf{X}$ is shown by $\mathbf{X}_{\bullet}$. The i^{th} row of $\mathbf{X}$ is denoted by $\mathbf{X}_i$ and the j^{th} column of $\mathbf{X}$ is denoted by $\mathbf{X}^j$. The following example, illustrates the notations that we use in this study.

Example 3.3 *Consider the following matrix,* $\mathbf{X}$*. In this matrix,* $\mathbf{X}_{34} = 0$*,* $\mathbf{X}_{1\bullet} = 4$*,* $\mathbf{X}^{\bullet 4} = 1$*,* $\mathbf{X}^{\bullet}$*,* $\mathbf{X}_{\bullet}$*,* $\mathbf{X}_2$ *and* $\mathbf{X}^5$ *is shown below:*

$$X = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$X^{\bullet} = \begin{pmatrix} 3 & 3 & 3 & 1 & 1 & 0 & 0 \end{pmatrix} \quad X_{\bullet} = \begin{pmatrix} 4 \\ 4 \\ 3 \end{pmatrix}$$

$$X_2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \quad X^5 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

We now intuitively explain how itemsets can be found in the factors of a matrix decomposition. Consider a decomposition of an input matrix $\mathbf{X}$ with k latent factors: $\mathbf{X}_{m\times n} \approx \mathbf{W}_{m\times k} \times \mathbf{H}_{k\times n}$. Each of the k columns of $\mathbf{W}$ can be mapped to a set of columns of $\mathbf{X}$ (through the product with $\mathbf{H}$). As columns of $\mathbf{X}$ correspond to *items*, columns of $\mathbf{W}$ correspond to *packet of items*. On the other hand, in $\mathbf{H}$, one observes the relation between each packet of items of $\mathbf{W}$ and each original item of $\mathbf{X}$. Using this setting, one can easily reconstruct itemsets of $\mathbf{X}$. The following example illustrates this idea.

Example 3.4 *Consider the following decomposition of the form* $\mathbf{X} = \mathbf{W} \times \mathbf{H}$

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The matrix on the left ($\mathbf{X}$*) contains three itemsets shown in three different colors. The same colors are used in the matrices on the right hand side to highlight the packet of items (in* $\mathbf{W}$*) and the associations of the items to each packet (in* $\mathbf{H}$*). The columns of* $\mathbf{W}$ *represent packets of items. The first packet has a support of 3 and the second and the third have a support of 2. In order to find the items associated to each packet, we need to look at the rows of* $\mathbf{H}$ *to see which items are assigned to each packet. Items one, two and three are in the first packet, item four and five in the second and item six in the third packet.*

Based on this example, one can see that this procedure provides a direct way of finding patterns in the dataset. However, we still need to establish a theoretical link between BMF and frequent itemset mining so that we can interpret the notions used in itemset mining domain (such as minimum support value) using BMF factors. To establish such link, we define the notion of *valid transaction matrix*, that will be the input of the proposed method. One should note that any binary matrix can be easily transformed into a valid transaction matrix.

Definition 3.1 ***(Valid transaction matrix)*** *A matrix* $\mathbf{X}_{m\times n}$ *is called a valid transaction matrix if it is binary and there is no rows or columns with all elements equal to zero. More formally,* $\mathbf{X}_{m\times n}$ *is a valid transaction matrix if:*
(1) $\forall i, j,\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ \mathbf{X}_{ij} \in \{0, 1\}$
(2) $\forall i, j,\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ \mathbf{X}_{i\bullet} > 0,\ \mathbf{X}^{\bullet j} > 0$

Now we consider different binary decompositions of a valid transaction matrix

3.4.2 STRONG DECOMPOSITION

A strong decomposition is simply an exact decomposition, *i.e.* a decomposition which perfectly reconstructs the input and produces no error in the approximation.

Definition 3.2 ***(Strong decomposition)*** *A matrix* $\mathbf{X}_{m\times n}$ *is strongly decomposable if:*
(1) it is a valid transaction matrix
(2) $\exists k,\ 1 \leq k \leq n$ *such that* $\mathbf{X}_{m\times n} = W_{m\times k} \times H_{k\times n}$
(3) $\mathbf{W} \in \{0,1\}$, $\mathbf{H} \in \{0,1\}$, $\mathbf{W} \times H \in \{0,1\}$

Condition (3) states that the factors $\mathbf{W}$ and $\mathbf{H}$ as well as their product ($\mathbf{W} \times \mathbf{H}$) are binary. The notion of strong decomposition furthermore implies that one can find factors that are all relevant, in the following sense.

Property 3.1 *If* $\mathbf{X}$ *can be decomposed strongly, then there is a strong decomposition of the form* $\mathbf{X}_{m\times n} = \mathbf{W}_{m\times k} \times \mathbf{H}_{k\times n}$ *such that there is no columns in* $\mathbf{W}$ *and no rows in* $\mathbf{H}$ *with all elements equal to zero. More formally, for all* t, $1 \leq t \leq k$, $\mathbf{W}^{\bullet t} > 0$ *and* $\mathbf{H}_{t\bullet} > 0$.

Proof. If $\mathbf{X}$ is strongly decomposed and $\mathbf{W}$ (resp. $\mathbf{H}$) contains one or more fully-zero columns (resp. rows), then one can remove the zero columns of $\mathbf{W}$ and their corresponding rows in $\mathbf{H}$ (resp. zero rows of $\mathbf{H}$ and their corresponding columns in $\mathbf{W}$), and still obtain, through the product of the simplified matrices, a strong decomposition of $\mathbf{X}$ with k' latent factors such that $k' < k$.□
The following example illustrates Prop. 3.1 where one can simply remove the second column of $\mathbf{W}$ as well as the second row of $\mathbf{H}$ (shown in red) and obtain exactly the same approximation:

Example 3.5

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$$

Using the above definitions, we are now ready to provide the following theorem which describes a first link between the itemsets mined from $\mathbf{W}$ and $\mathbf{H}$ and itemsets in $\mathbf{X}$.

Theorem 3.1 *Let* $\mathbf{X}_{m\times n} = \mathbf{W}_{m\times k} \times \mathbf{H}_{k\times n}$ *be a strong decomposition of* $\mathbf{X}$*. If a packet of items with support value* f *is found in* $\mathbf{W}$ *(i.e.* $\exists p,\ 1 \leq p \leq k,\ \mathbf{W}^{\bullet p} = f$*), then there is at least one itemset with support value of at least* f *in* $\mathbf{X}$*.*

Proof. Suppose that packet p, $1 \leq p \leq k$, has support value f in $\mathbf{W}$, i.e $\sum_{i=1}^{m} W_{ip} = f$. Because of the definition of *strong decomposition* and of Prop. 3.1, there exists at least one q, $1 \leq q \leq n$, such that $\mathbf{H}_{pq} = 1$. Then, for all i, $1 \leq i \leq m$, we have:

$\mathbf{X}_{iq} = \sum_{j=1}^{k} W_{ij}H_{jq} \geq W_{ip}H_{pq}$

And thus: $\sum_{i=1}^{m} X_{iq} \geq \sum_{i=1}^{m} W_{ip}H_{pq} = H_{pq}\sum_{i=1}^{m} W_{ip}$

So finally: $\sum_{i=1}^{m} X_{iq} \geq f$

which establishes the theorem.□

In simpler terms, Theorem 3.1 shows that, in a strong decomposition, if there exists a frequent *packet of items* in $\mathbf{W}$ with support value of f, then there is a corresponding frequent itemset with support value of at least f in $\mathbf{X}$ (see Example 3.4).

An important consequence of Theorem 3.1 is regarding the size of the captured itemset. In this theorem, for a p such that $\mathbf{W}^{\bullet p} = f$, if only one q is found such that $\mathbf{H}_{pq} = 1$, then there is an itemset of size one (a singleton), which is not very useful. However, it is easy to prove that if several such q's are found, then an itemset, and not a singleton, in $\mathbf{X}$ is captured. This is expressed in the following corollary.

Corollary 3.1 *If* $\mathbf{X}_{m\times n} = \mathbf{W}_{m\times k} \times \mathbf{H}_{k\times n}$ *is a strong decomposition of* $\mathbf{X}$*, and there exists a packet of items* p*,* $1 \leq p \leq k$ *of size* l *in* $\mathbf{H}$ *(i.e.* $\mathbf{H}_{p\bullet} = l$*) with support value* f *in* $\mathbf{W}$ *(i.e.* $\mathbf{W}^{\bullet p} = f$*), then there exists an itemset of size at least* l *and support value of at least* f *in* $\mathbf{X}$*.*

The following example, illustrates the Corollary 3.1.

Example 3.6 *Consider the following decomposition of the form* $\mathbf{X} = \mathbf{W} \times \mathbf{H}$ *where itemsets are colored in matching colors* $\mathbf{X}$*,* $\mathbf{W}$ *and* $\mathbf{H}$*.*

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

As one can see, the itemset in red has support value of 2 and length 3, and it is captured by the first packet of items (first column of **W***, shown in red) where the support of the packet is also 2. The items associated to this packet can be seen in the first row of* **H** *(also in red) where one can observe that the first three items are assigned to this packet. As a result, this factor retrieves the red itemset in* **X** *with its exact size and support value.*

The second itemset of **X***, shown in blue, can be captured by the second packet of items in* **W***,* i.e. *the second column, also in blue. This packet has a support value of 2 and the itemset associated to it is an itemset of size 2, found via the second row of* **H** *represented also in blue.*

Corollary 3.1 can be directly derived from Theorem 3.1 (we will provide soon a proof of a more complex version of this Corollary in Theorem 3.3). This result is very important since it shows that one can systematically mine frequent itemsets of any size using the factors of the decomposition, and as matrix decomposition provides the main structure of the data, we can expect them to be representative itemsets (we will elaborate this point in the experimental section via the quality metrics available in the literature).

As one can see, once a transaction matrix has been strongly decomposed into latent, binary factors, we can efficiently obtain some representative frequent itemsets from the decomposition without reloading the data into memory. In practice, however, a strong decomposition of a transaction matrix may not exist. Accordingly, we now consider a more realistic decomposition, called *approximately strong decomposition*, and reformulate the materials provided so far.

3.4.3 APPROXIMATELY STRONG DECOMPOSITION

A direct extension of strong decomposition is to no longer assume that the decomposition is perfect, i.e. has no error. This can be simply done by adding an error term to the reconstruction of **X** from the latent factors.

Definition 3.3 ***(Approximately strong decomposition)*** *A matrix* $\mathbf{X}_{m\times n}$ *can be decomposed approximately strongly if:*
(1) it is a valid transaction matrix
(2) $\exists k,\ 1 \leq k \leq n$ *such that* $\mathbf{X}_{m\times n} = W_{m\times k} \times H_{k\times n} + \epsilon_{m\times n}$
(3) $\mathbf{W} \in \{0,1\}^{m\times k}$, $\mathbf{H} \in \{0,1\}^{k\times n}$ *and* $\mathbf{W} \times H \in \{0,1\}^{m\times n}$

Note that having values of $\mathbf{X}$, $\mathbf{W}$, $\mathbf{H}$ and $\mathbf{W} \times \mathbf{H}$ in $\{0,1\}$ implies $\epsilon \in \{-1,0,1\}^{m \times n}$. It should be also noted that Prop. 3.1 holds for the approximate decomposition as well. Based on Def. 3.3, now we can rewrite Theorem 3.1.

Theorem 3.2 *Let $\mathbf{X}_{m\times n} = \mathbf{W}_{m\times k} \times \mathbf{H}_{k\times n} + \epsilon_{m\times n}$ be an approximately strong decomposition of $\mathbf{X}$, and let ϵ_{max} be the maximum of absolute values of $\epsilon^{\bullet}$. If a frequent packet of items with support value of f is found in $\mathbf{W}$ (i.e. $\exists p,\ 1 \le p \le k,\ \mathbf{W}^{\bullet p} = f$), then there is at least one itemset with support value of at least $f - \epsilon_{max}$ in $\mathbf{X}$.*

Proof Suppose that packet p, $1 \le p \le k$, has support value of f in $\mathbf{W}$, i.e $\sum_{i=1}^{m} \mathbf{W}_{ip} = f$. Because of the definition of *approximately strong decomposition* and of Prop. 3.1, there exists at least one q, $1 \le q \le n$, such that $\mathbf{H}_{pq} = 1$. Then, for all i, $1 \le i \le m$, we have:

$$\mathbf{X}_{iq} = \sum_{j=1}^{k} W_{ij}H_{jq} + \epsilon_{iq} \ge W_{ip}H_{pq} + \epsilon_{iq} \quad \text{and thus:}$$

$$\sum_{i=1}^{m} X_{iq} \ge \sum_{i=1}^{m} W_{ip}H_{pq} + \sum_{i=1}^{m} \epsilon_{iq} = H_{pq}\sum_{i=1}^{m} W_{ip} + \sum_{i=1}^{m} \epsilon_{iq} = f + \sum_{i=1}^{m} \epsilon_{iq} \quad \text{so finally:}$$

$$\sum_{i=1}^{m} X_{iq} \ge f - \epsilon_{max}$$

which establishes the theorem.□

Considering itemsets of a given size with approximately strong decompositions is slightly more difficult than with strong decompositions. The following theorem extends the corollary of Theorem 3.1 (Corollary 3.1) to the case of approximately strong decompositions.

Theorem 3.3 *Let $\mathbf{X}_{m\times n} = \mathbf{W}_{m\times k} \times \mathbf{H}_{k\times n} + \epsilon_{m\times n}$ be an approximately strong decomposition of $\mathbf{X}$, and $\gamma(a,b) = (\mathbf{W}^{a} \cdot \epsilon^{b})$ where $\cdot$ represents dot product ($a \in \{1,\cdots,k\}$, $b \in \{1,\cdots,n\}$). Also let $\gamma(p) = \min_{1\le j\le n} \mathbf{H}_{pj} \times \gamma(p,j)$, $\forall p,\ 1 \le p \le k$. Now, if there exists a packet of items p, $1 \le p \le k$, of size l (e.g. $\{q_1, q_2, \cdots, q_l\}$) in $\mathbf{H}$ (i.e. $\mathbf{H}_{pq_1} = \mathbf{H}_{pq_2} = \cdots = \mathbf{H}_{pq_l} = 1$ and $\mathbf{H}_{p\bullet} = l$) with support value of f in $\mathbf{W}$ (i.e. $\mathbf{W}^{\bullet p} = f$), then there exists an itemset of size at most l and with support value of at least $f + \gamma(p)$ in $\mathbf{X}$. Furthermore, if $\gamma(p) = 0$, then $\{q_1, q_2, \cdots, q_l\}$ is an itemset of size l in $\mathbf{X}$ with support value of f.*

Proof. First note that $\gamma(a,b)$ corresponds to the reconstruction error for item b in packet a, and that $\gamma(p)$ corresponds to the minimum reconstruction error of all items belonging to packet p. Without loss of generality, we assume that a frequent itemset

of size l and support f occurs in the first f rows of $\mathbf{W}$ and first l columns of $\mathbf{H}$ (see the following illustration).

$$W = \begin{array}{c} \\ 1 \\ \vdots \\ f \\ \vdots \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{ccc} 1 & \cdots & k \end{array} \\ \begin{pmatrix} 1 & & \\ & & \\ & & \\ 1 & & \\ 0 & & \\ \vdots & & \\ 0 & & \end{pmatrix} \end{array} \qquad H = \begin{array}{c} \\ 1 \\ \vdots \\ k \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & \cdots & l & l+1 & \cdots & n \end{array} \\ \begin{pmatrix} 1 & \cdots & 1 & 0 & \cdots & 0 \\ & & & & & \\ & & & & & \end{pmatrix} \end{array}$$

We thus have:

$$1^{st}\ item : \sum_{i=1}^{f} X_{i1} = \sum_{i=1}^{f} W_{i1}H_{11} + \sum_{i=1}^{f} \epsilon_{i1}$$
$$\vdots$$
$$l^{th}\ item : \sum_{i=1}^{f} X_{il} = \sum_{i=1}^{f} W_{i1}H_{1l} + \sum_{i=1}^{f} \epsilon_{il}$$

which can be rewritten:

$$1^{st}\ item : \sum_{i=1}^{f} X_{i1} = \sum_{i'=1}^{m} W_{i'1}H_{11} + \sum_{i'=1}^{m} W_{i'1}\epsilon_{i'1}$$
$$\vdots$$
$$l^{th}\ item : \sum_{i=1}^{f} X_{il} = \sum_{i=1}^{m} W_{i1}H_{1l} + \sum_{i'=1}^{m} W_{i'1}\epsilon_{i'l}$$

Furthermore, according to the definition of $\gamma(p)$ for any packet p, we have:

$$1^{st}\ item : \sum_{i=1}^{f} X_{i1} = f + \gamma(1,1) \geq f + \gamma(1)$$
$$\vdots$$
$$l^{th}\ item : \sum_{i=1}^{f} X_{il} = f + \gamma(1,l) \geq f + \gamma(1)$$

which shows that there is an itemset included in $\{i_1, \cdots, i_l\}$ with support of at least $f + \gamma(1)$. Furthermore, if $\gamma(1) = 0$, then $\{i_1, \cdots, i_l\}$ is an itemset of size l and support value f in $\mathbf{X}$.□

The following example illustrates how Theorem 3.3 (which is the general form of Theorem 3.2) works in practice.

Example 3.7 *Consider the following Approximately strong decomposition of the form* $\mathbf{X} = \mathbf{W} \times \mathbf{H} + \epsilon$*:*

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

According to the decomposition shown above and Theorem 3.3, we have the following matrix for $\mathbf{H} \circ (\mathbf{W}^a.\epsilon^b)$, $\forall 1 \leq a \leq k$, $1 \leq b \leq n$ *:*

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \circ \begin{pmatrix} 0 & 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

where $\circ$ *denotes the Hadamard product (element-wise product) and* $.$ *denotes the dot product. Accordingly, the* γ *value is -1 for both packets,* i.e $\gamma(1) = -1$ *and* $\gamma(2) = -1$ *as the minimum value in both rows are -1. According to the decomposition, the first packet,* i.e *the first column of* $\mathbf{W}$ *has support value of 2 and as* $\gamma(1) = -1$*, one can conclude that this column represents an itemset with support value of at least* $2 - 1 = 1$*. If we look at the first row of* $\mathbf{H}$*, which corresponds to this packet, we see that four items are allocated to this packet. As a result, this factor represents an itemset with support value of at least 1 and size of at most 4 (shown in red).*

The second packet, i.e *the second column of* $\mathbf{W}$*, has support value of 3 and, as* $\gamma(2) = -1$*, one can conclude that this packet represents an itemset with support value of at least* $3 - 1 = 2$ *and size of at most 2 as there are two items associated to this packet,* i.e. *the second row of* $\mathbf{H}$*. This itemset in addition to its corresponding row/column is shown in blue.*

Considering Theorem 3.3, one can easily and systematically mine representative itemsets using an approximately strong decomposition. Algorithm 3 shows this procedure, which is denoted as DIM: Decomposition Itemset Miner. Note that, according to Theorem 3.3, when $\gamma(p) = 0$, one is able to identify an itemset in $\mathbf{X}$ with its exact support and size.

3.5 IMPLEMENTATION

The previous development shows how one can find representative itemsets form a transaction matrix using the decomposition factors. Based on that, having a valid transaction matrix, one can easily apply the decomposition (to obtain the factors, i.e. $\mathbf{W}$ and $\mathbf{H}$) and start mining the itemsets. According to the input of Algorithm 3, we need a decomposition method to obtain high quality factors which also satisfies the binary reconstruction constraint.

We have discussed, in Chapter 2, different methods satisfying the conditions required for itemset mining. For the experimental section of this chapter, we used the

Algorithm 3 Mining representative itemsets with DIM

Input: **W**, **H**, vector γ, min-supp f, min-size l

Output: S : a set of itemsets with length of at most l and support of at least f

```
1: S ← ∅
2: for all rows r in H do
3:     if H_{r•} ≥ l then
4:         if W^{•r} + γ(r) ≥ f then
5:             P ← items corresponding available in r
6:             S ← S ∪ {P}
7:         end if
8:     end if
9: end for
10: return S
```

results provided therein as the input of DIM in order to systematically find representative itemests. One should note that we are not considering the *Boolean* matrix decomposition [Miettinen, 2008b] in this study since all our theoretical developments are based on classical matrix multiplication.

3.6 EXPERIMENTS

In this section we evaluate the proposed method, DIM, based on the quality of produced itemsets and the mining efficiency through a set of experiments on different datasets. The qualitative experiments compare the set of itemsets output by DIM with the set of all closed frequent itemsets output by LCM algorithm [Uno et al., 2005]. The efficiency experiments compare the mining time of DIM and LCM. For LCM, we use its authors C implementation, while DIM is implemented in Matlab, which provides efficient matrix operations. The experiments are conducted on a Linux machine with an Intel Xeon E-2630 with 6 cores @ 2.30 Ghz with 32 GB of RAM. No multiprocessing or multithreading has been applied on DIM or LCM.

The datasets used in the experiments are publicly available datasets from the FIMI repository [1]. Following the methodology presented in [Uno et al., 2005], we choose `pumsb`, `accidents` and `T40I10D100k` as representatives of *structured dense*, *dense* and *sparse* datasets respectively. Although `T40I10D100k` is of size 100K-by-942 and

[1] `http://fimi.ua.ac.be/data/` (last visited 03-Jun-2015)

is sparse, if we mine it with very small support values, we have a huge number of itemsets. `Accidents` is of size 340K-by-468 and each transaction contains a lot of items and, thus, it is dense. Since it is not structured, the number of frequent itemsets and closed itemsets are almost equal for any minimum support value, which makes it very difficult to mine. `Pumsb` has 49K transactions and 2K items and each transaction is rather dense. If we mine that with small minimum support values, then the number of closed itemsets will be much smaller than the number of frequent itemsets.

3.6.1 EFFICIENCY EVALUATION

Contrary to other frequent itemset mining techniques, the complexity of DIM does not depend on the minimum support threshold, i.e. once a decomposition is obtained, the mining process is quite fast and straightforward. This experiment may be considered unfair, as LCM aims at finding all closed frequent itemsets, while the proposed approach only computes a small set of representative frequent itemsets. DIM might thus be compared with approaches computing a representative subset of the frequent itemsets. However, as discussed in the related work (Section 3.3), most existing approaches to identify a representative subset of the frequent itemsets are two-pass approaches, that first compute all (closed) frequent itemsets and then compute a subset of these frequent itemsets. The existing one-pass approach [Guns et al., 2010] is more general than the proposed method and does an exhaustive search on a huge search space. Its time complexity is thus much higher than that of DIM. Therefore, their running time is higher than the one of LCM alone, and comparing running time of the proposed method to that of LCM is indeed unfair, but to our disadvantage.

Figure 3.1 presents the mining time with a varying minimum support for both LCM and DIM. As expected, DIM's running time is constant. Note that, in DIM, while the decomposition is done, the mining phase is quite straightforward and fast. Accordingly, one can see a constant line in Fig. 3.1 as the time consuming phase of DIM is the decomposition. In practice, the decomposition has only to be done once, and then it can be exploited by Algorithm 3 for any support value we desire. It can be observed that for high support values LCM is faster than DIM. However, for lower support values, LCM suffers from the combinatorial explosion of the number of results, and needs much more computation time than DIM. This point is more crucial in structured dense datasets, like `pumsb`, where even with high support values, itemset mining task is very difficult. In dense datasets, like `accidents`, LCM becomes very

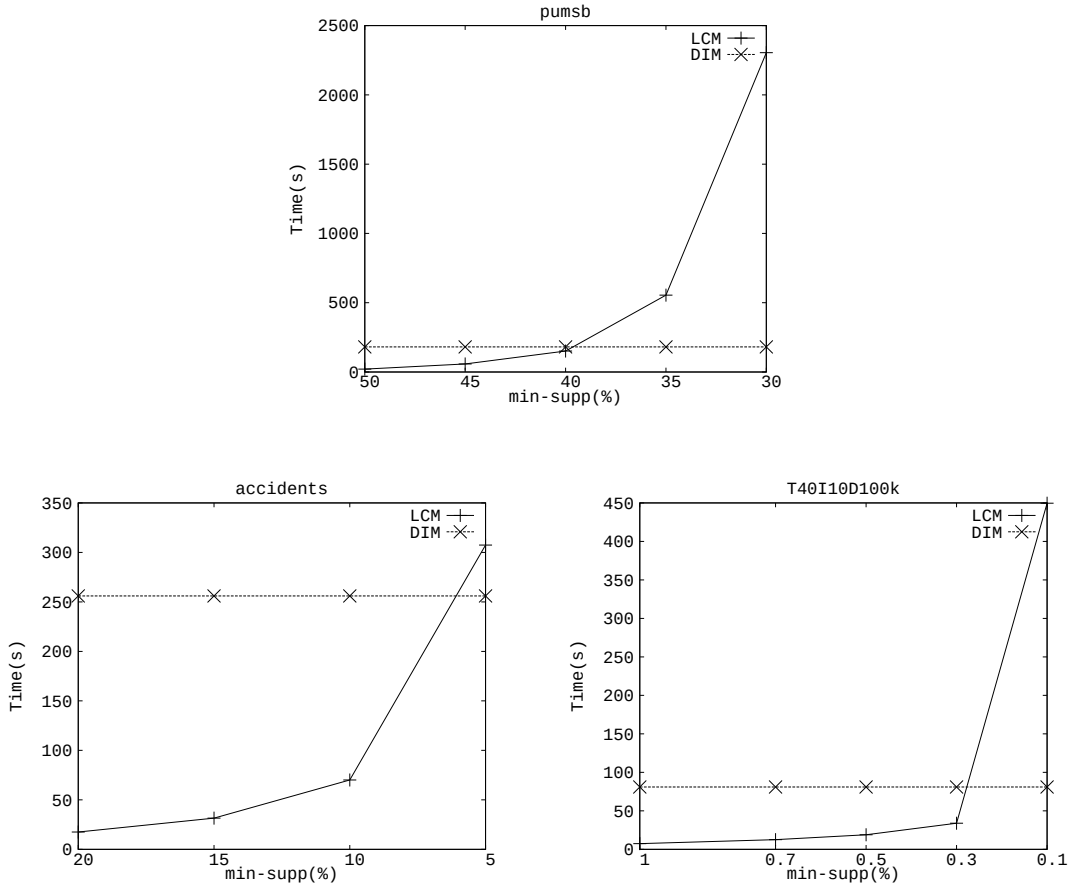

Figure 3.1: Time comparison of LCM and DIM, varying the min-supp for `pumsb`, `accidents` and `T40I10D100k`

slow for support values smaller than 10%. In sparse datasets, like `T40I10D100k`, it is easy to find the itemsets with high support values. However, as these datasets contain quite huge itemsets with small support values, LCM spends a long time to find closed itemsets.

3.6.2 QUALITATIVE EVALUATION

DIM mines very few itemsets and the efficiency experiments have shown that for low support values this approach could output a solution faster than state-of-the-art algorithms. It thus remains to evaluate the quality of the itemsets found with respect to the complete set of closed frequent itemsets.

For this, we rely on metrics presented in [Gupta et al., 2000] and used for evaluating the quality of a subset of frequent itemsets with respect to the entire set of frequent itemsets. Following those metrics, we denote the entire set of closed itemsets (output

by LCM) as *base-itemsets* and itemsets outputs by DIM as *found-itemsets*. The following two metrics evaluate the quality of found-itemsets with respect to base-itemsets.

Recoverability: this metric measures how well a collection of found-itemsets can cover base-itemsets. Consequently, this metric is similar to *recall*. For a base-itemset B_i, recoverability is calculated as follows: among all found-itemsets F_j we look for the one which has the maximum number of items in common with B_i, namely F^i_{max}. The recoverability of B_i is defined as:

$$recoverability(B_i) = \frac{|B_i \cap F^i_{max}|}{|B_i|} \tag{3.1}$$

The total recoverability is a weighted average (bigger base-itemsets contribute more than smaller ones) of the recoverability of all base-itemsets.

Precision: As we can see, having one single large found-itemset (possibly including all items) results in a recoverability of 1 for all base-itemsets. Therefore, we need another metric to penalize such cases. Spuriousness is another metric to evaluate the quality of found-itemsets. For a found-itemset F_i, spuriousness is defined as follows: among all base-itemsets, we find the one which has the maximum number of items in common with F_i, namely B^i_{max}. Then spuriousness of F_i is defined as follows:

$$spuriousness(F_i) = \frac{|F_i - F_i \cap B^i_{max}|}{|F_i|}$$

In this case also the total spuriousness is computed by a weighted average (bigger found-itemsets contribute more than smaller ones) of the spuriousness of each found-itemset. Precision of an itemset is then defined as $1 - spuriousness$. The following example illustrates these metrics:

Example 3.8 *Consider $\{A, J\}$ and $\{A, B, C\}$ as found-itemsets and the table 3.2 as a set of base-itemsets:*

Table 3.2: Base-itemsets

#1	#2	#3	#4	#5	#6
{A,B}	{A,B,D}	{A,B,C,D}	{A,D,E,F}	{A,F,G,H,I}	{A,C,D,F,I}

All the base-itemset can be best recovered by the second found-itemset ($\{A, B, C\}$). All items of # 1, two items of # 2, all items of # 3, two items of # 4, one item of # 5 and two items of # 6 are covered. As a result we have the following recoverability for each base-itemset:

$$Recoverability(\#1) = \frac{2}{2} \quad Recoverability(\#2) = \frac{2}{3} \quad Recoverability(\#3) = \frac{3}{3}$$

$$Recoverability(\#4) = \frac{2}{4} \quad Recoverability(\#5) = \frac{1}{5} \quad Recoverability(\#6) = \frac{2}{5}$$

Consequently, the total recoverability of this set of found-itemsets is a weighted average over the above mentioned values:

$$\frac{\frac{2}{2} \times 2 + \frac{2}{3} \times 3 + \frac{3}{3} \times 3 + \frac{2}{4} \times 4 + \frac{1}{5} \times 5 + \frac{2}{5} \times 5}{2+3+3+4+5+5} = 0.47$$

For spuriousness, all the base-itemsets can only match one item of the first found-itemset ($\{A, J\}$); however, the second found-itemset ($\{A, B, C\}$) can be completely matched with # 3. Accordingly, the spuriousness of this set of found-itemsets (using a similar weighted average as shown above) is

$$\frac{\frac{1}{2} \times 2 + \frac{0}{3} \times 3}{2+3} = 0.20$$

and as a result, the precision of these found-itemsets is $1 - 0.2 = 0.8$.

A summary of the results is presented in Table 3.3. Number of (closed) frequent

Table 3.3: Comparing LCM closed itemsets and DIM itemsets

Dataset	min-supp	#(LCM)	#(DIM)	Prec.	Recov.
Pumsb	40%	44.7M	7	71%	72%
	20%	7.49B	10	34%*	76%*
	10%	≫10B	15	27%*	94%*
	5%	≫10B	17	25%*	94%*
	1%	≫10B	20	22%*	94%*
accidents	10%	9.97M	9	75%	67%
	5%	64.8M	13	69%	68%
	1%	1.62B	15	40%*	53%*
T40I10D100k	1%	64481	1	88%	2.8%
	0.5 %	1.27M	7	84%	9.5%
	0.3 %	3.56M	8	87%	10%
	0.1%	18.4M	9	87%	18%

itemsets for both LCM and DIM as well as the precision and recoverability of DIM itemsets are shown in this table. The minimum size for the itemsets is set to 2 since singletons are not of interest. The number of latent factors (k) is set to 30 since larger values of k results in uninteresting itemsets (i.e. itemsets with large error value). To

increase the readability, large numbers are suffixed by M (millions) or B (billions). Cells with a star (*) report an estimate: LCM outputs too many items, resulting in output file of hundreds of Gigabytes and hard disk saturation. In these cases only a uniform random sample of 5 million itemsets was used to compute the metrics.

Two points in this table are of particular interest. First, whereas the number of closed frequent itemsets can get over billions of itemsets (in some cases it is even hard to count them, see `pumsb` for example), the number of frequent itemsets output by DIM is always very small. Thus, the frequent patterns output by DIM can be quickly examined by an analyst, which is not the case for all closed frequent itemsets produced by LCM.

Second, despite being so few, DIM frequent itemsets exhibit significant recoverability values. This fact becomes more important when the precision is also high, like `accidents` with minimum support of 10% and `pumsb` with minimum support of 40%. In these cases, not only we are able to recover a large amount of base-itemsets, but also we can guarantee that these itemsets are not long, uninteresting ones (according to the precision value). In other words, using DIM itemsets, an analyst can have a general, precise view of the entire data without being overwhelmed by millions or billions itemsets. For instance, in case of `accidents` with minimum support of 10%, we can look at 9 representative itemsets produced by DIM (instead of looking at 9.97M itemsets of LCM) and recover 67% of all closed itemsets.This is an important contribution: it experimentally shows that by looking at DIM patterns, an analyst will have a significant glimpse of the knowledge that can be extracted from the dataset, *in a very short amount of human analysis time*.

The case for sparse datasets like `T40I10D100k` is different. For this dataset DIM extracts few itemsets with very high precision values but a relatively low recoverability (18% for a support of 0.1%). This result is in line with the different nature of `T40I10D100k` compared to `pumsb` and `accidents`: `T40I10D100k` is a synthetic, sparse dataset and it does not exhibit the same strong "pattern structure" as the two other datasets. This can be confirmed with LCM results: there are few patterns comparatively to the other datasets, and they appear at very low support values. In such a case, the proposed method based on matrix decomposition is not the most adapted to get a representative view of the complete set of itemsets of LCM, as matrix decomposition techniques are likely to remove small groups of ones from the matrix for the sake of approximation. This results in losing some information that appears in the complete output of LCM. On the other hand, the itemsets captured by DIM are captured

with an extremely high precision; for instance, for a support of 0.1% the analyst will have to analyze only 9 itemsets, and get a good idea of the content of about one fifth of the complete number of itemsets.

Since DIM provides few itemsets, one may wonder if the amount of information provided by these itemsets can be also obtained by taking the top p (here, the term *top* is considered with respect to the support value, *i.e.* the p itemsets having the maximum support values) itemsets produced by LCM, where p is equal to the number of itemsets produced by DIM. Table 3.4 gives recoverability for the top p itemsets of LCM for each dataset (precision is by definition 100% in this case). For instance, in the `accidents` dataset with minimum support of 10%, DIM outputs 9 itemsets, so we take the top 9 itemsets from the results of LCM, and compute the recoverability, which is 13%. The recoverability of the top p itemsets of LCM is more than 4 times lower than those of DIM in real datasets. In the synthetic dataset, however, the improvement is marginal. These results, in general, confirms that DIM itemsets convey more information than traditional top p itemsets techniques.

Dataset	p	min-supp	Recov. (LCM)	Recov. (DIM)
`pumsb`	7	40%	15%	72%
`accidents`	9	10%	13%	67%
`T40I10D100k`	9	0.1%	12%	18%

Table 3.4: Comparing top-p LCM closed itemsets with DIM itemsets

3.7 CLOSING REMARKS

In this chapter, we have examined the problem of finding representative itemsets through decomposition of the input matrix. Using theoretical analysis, we have shown that matrix decomposition can help us mining representative frequent itemsets as it extracts the main information of the data.

The experiments have shown that although for high support values the proposed approach is less time efficient than state-of-the-art algorithms, for low support values, it is much faster than those algorithms. This approach is also highly scalable with respect to other algorithms according to the fact that the classical itemset mining approaches become very slow since they need to explore a huge combinatorial space as the number of items increases.

The proposed method only finds a handful of representative itemsets, but the experiments show that these itemsets convey a significant portion of the information of all frequent itemsets. We advocate that contrary to classical methods, the mining time of the proposed method is *well invested*: what is the utility of a very fast algorithm that outputs millions of patterns that an analyst will take hours to make some sense of? With the proposed method, the computer does most of the work, and the analyst is presented with concise, reliable and manageable information.

II Instance Selection

4 REPRESENTATIVE OBJECTS FOR LARGE DATASETS

4.1 INTRODUCTION

Dealing with a large number of instances is a major problem in many scientific areas such as data mining or data analysis. In big data, selecting most representative data points is not only of interest for analysts but also simplifies the visualization. In many applications, such as text mining for instance, many data points are similar (or correlated) and could be described by one representative data point [Xin et al., 2005, Zhou et al., 2009]. For instance, in [Zhou et al., 2009], the authors consider the problem of designing a personalized recommendation algorithm and try to provide a new technique based on higher order correlations so that the redundant correlations are eliminated. The main idea in that paper is that similarities between the same attributes (potentially, the attributes with higher weights) among data instances may introduce strong correlations. The following example illustrates the idea of representative object selection in the frequent itemset mining domain (see Chapter 3 for more detail on frequent itemset mining).

Example 4.1 *Consider the following set of frequent itemsets mined by an arbitrary mining algorithm:*

$$\{I_1, I_3, I_4, I_5\} \quad \{I_1, I_3, I_4, I_6\} \quad \{I_1, I_4, I_5, I_6\} \quad \{I_3, I_4, I_5, I_6\} \quad \{I_1, I_3, I_4, I_5, I_6\}$$

One can observe that this set of itemsets can be well represented by the last itemset as they are all (highly) correlated and the last itemset can cover the others.

Another example of such cases is the protein datasets where one need to select a set of non-redundant data points in order to apply statistical analysis of protein sequence-structure relations [Hobohm et al., 1992]. In this domain, one is interested

in selecting the datasets which are similar as the set of datasets considered in this domain could be very similar and, as a result, redundant.

Many datasets in data mining could be seen as matrices where rows represent the objects (instances) and columns represent the features. In such cases, we are interested in selecting a few objects of this matrix such that we can have a global view of the entire data. These objects are called representative and are considered in many interesting applications [Leroy et al., 2015]. For instance, a biologist might be interested in finding a set of genes in a gene expression data which are responsible for a particular disease. Classic approaches such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) fail to find the answer to this question [Deshpande and Rademacher, 2010, Mahoney and Drineas, 2009].

A simple solution to this problem is to perform a clustering on the dataset and consider the centroid of clusters as representative objects. However, there are some drawbacks in such approaches. Firstly, in real world cases, one does not know the structure of the data in advance. Accordingly, it might be very difficult to choose the right number of clusters [Milligan and Cooper, 1985, Ben-Hur and Guyon, 2003]. Secondly, one data center per cluster may not be able to provide a good global view of the data. Consider a complex, high-dimensional dataset with millions of instances and only very few clusters; certainly, a huge dataset cannot be well represented with only several data points; on the other hand, clustering the data with a large number of clusters gives the chance to non-representative instances to be selected as representatives [Daszykowski et al., 2002]. Thirdly, in some applications (as we will see in the experimental section), the data is not clusterable, *i.e.* it contains only one type of data (all the genes involved in a given disease, all the news containing a given topic, etc). Lastly, the distribution of the data points plays a very important role in the process of data selection. Picking only the central points of the clusters is not able to handle this issue appropriately.

As an example, consider Fig. 4.1 where among more than 12K instances, 40 centroids of K-medoids have been displayed. Although these centroids can cover two *arms* of the data, the one on the top is not very well covered as only two representatives have been selected form there. We will see, in the experimental section, what this dataset represents and show how we can find other representatives with an alternative technique proposed later in this chapter.

As our fourth contribution, in this chapter, we propose a method which selects a handful of instances which can provide a global view of the data [Mirisaee et al.,

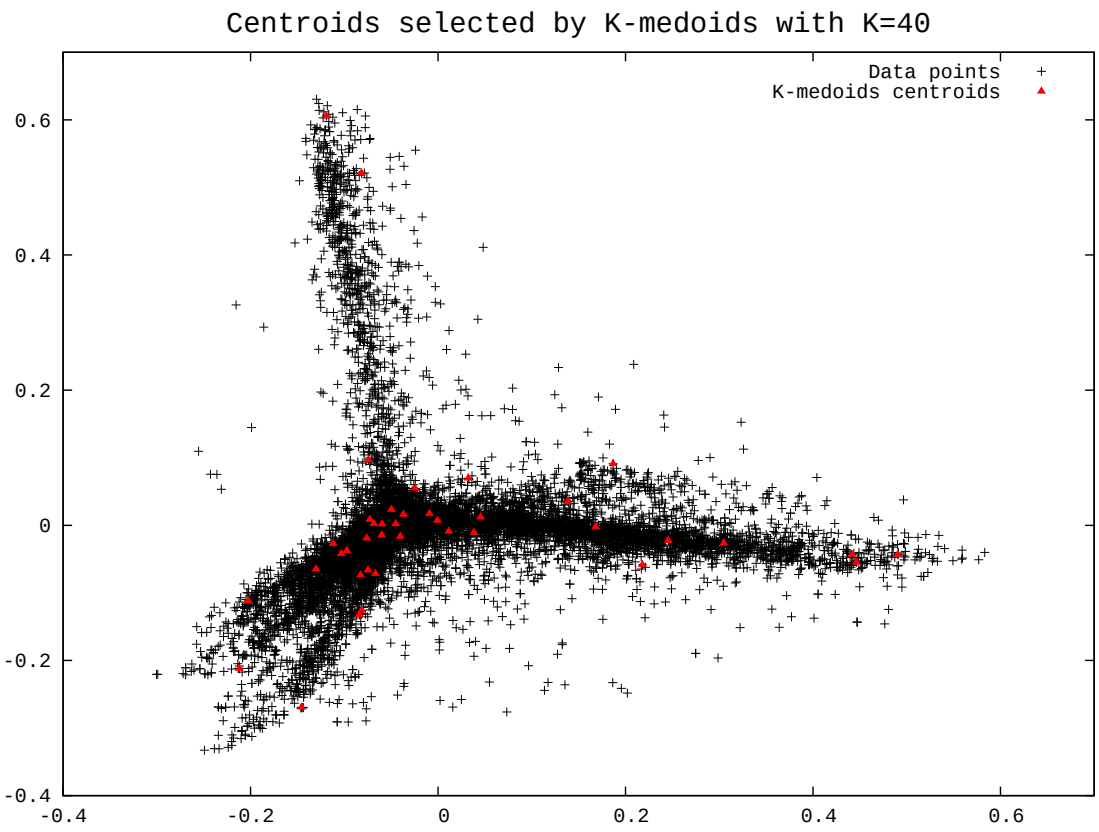

Figure 4.1: Centeroids provided by K-medoids with $k = 40$ on a 2D data with more than 12K instances.

2015a]. This method can be considered as an alternative technique for what has already been proposed in this context. More importantly, as it is explained in the experimental section, the proposed method can perfectly play a complementary role for other state-of-the-art representative subset selection algorithms, like DBSCAN-based technique.

This chapter is organized as follows: Section 4.2 discusses the related work on the topic. Then the main problem, its three variants along with the complexity analysis is explained in Section 4.3. The solutions to these three variants are provided in Section 4.4. In Section 4.5, experiments are discussed on synthetic and real datasets and an extensive analysis on the results is provided. Lastly, Section 4.6 provides some closing remarks of the chapter.

4.2 RELATED WORK

As mentioned before, high dimensionality as well as large number of data instances inspired researchers to look for data simplification techniques (when a few dimensions are concerned, the problem is easy to solve [Chaudhuri et al., 1994]). In general, data simplification (or data reduction) can be obtained in different ways. For instance, one may use feature selection algorithms in order to reduce the number of columns in a dataset [Blum and Langley, 1997, Liu and Motoda, 1998]. Embedded, filter and wrapper approaches are some examples of the feature selection problem.

In [Blum and Langley, 1997], the authors consider the methods which focus on the problem of finding and removing irrelevant features and the problem of selecting relevant examples. The former consists of finding the features which can describe the concept and the methods to combine them in order to reduce the number of features. The latter, however, focuses on selecting examples which are able to provide some improvements in the learning process. This task is mainly performed in order to satisfy different objectives, most of them concerns clustering applications. For example, if the learning is computationally expensive, one can select a subset of examples in order to make the learning process faster. Another case is when the labeling costs a lot as it may need a large amount of human expert time.

Ideally, the instance selection procedure should be model independent [Huan and Motoda, 2002], *i.e.* having a model M and two data sets s and t, where s is the entire dataset and t is a subsample (a selection of objects) of that, we need $P(M_s) \approx P(M_t)$, where $P()$ denotes the performance of a model. As discussed in [Huan and Motoda, 2002], there are different techniques to address the object selection problem. Perhaps, the easiest way is sampling: selecting some data points at random (mostly without replacement) or using a probability distribution [Kivinen and Mannila, 1994]. Alternatively, one can perform an adaptive sampling where the next object to be selected depends on the objects that have already been selected. The advantage of this technique is that data characteristics are considered in the selection process. However, one drawback of this type of techniques is that they may introduce other complexities while used in different applications [Chen et al., 2002].

A different approach to select instances from datasets is to use the classification techniques. For instance, one can take the points which *matter* to the classifier. Instance-Based Learning methods are examples of such technique. In [Aha et al., 1991], for

instance, the authors discuss the problem of using only some specific part of the data to address the incremental learning task. In this paper, the nearest neighbor algorithm is extended and, in order to handle its memory requirements, their algorithm relies only on some specific examples with small sacrifices in classification accuracy.

In [DuMouchel et al., 1999], the authors introduce the *squashing* technique which could be seen as a lossy database compression method. The idea in that work is to scale down large datasets while keeping its micro-structures which are important for data analysis. Basically, squashing has three steps, namely grouping, momentizing and generating. In grouping, the data is partitioned into subregions. Then moments are calculated using Taylor series approximation, and next, for each region a set of squashed elements are created. These three steps can be individually optimized in order to find the most desirable results. Squashing is shown to be much more effective than a simple random data selection.

In a recent work, Mall et. al. proposed a KNN-based method in order to provide a subset of data that can be representative of the inherit structure of data [Mall et al., 2014]. To do that, the authors first convert the data to a KNN graph and then they use the FURS method [Mall et al., 2013], which takes the nodes from dense regions of the graph, to obtain a subset. Then the subsets are used in different learning tasks such as clustering and classification.

Our study is different from the above-mentioned ones in that they all try to extract some examples from the data in order to improve the modeling tasks (such as classification). However, we do not consider such cases and aim at presenting only few examples from the dataset in order to provide a top-view of the entire data.

Another trend of data simplification techniques is matrix decomposition methods such as Non-negative Matrix Factorization (NMF) [Lee and Seung, 1999a, Lee and Seung, 2000] which try to cluster the data in both dimensions (as discussed in detail in Chapter 1); *i.e.* cluster the objects and attributes at the same time. Other approaches like PCA try to find linearly uncorrelated variable among the data [Wold et al., 1987]. Perhaps the most known technique in matrix decomposition family is SVD [Golub and van Loan, 1996] which provides low rank approximation via singular values and singular vectors. In general, as discussed in Chapter 1, SVD, when applied to a given matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ is a factorization of the following form:

$$X = U \Sigma V^T \tag{4.1}$$

where $U \in R^{m \times m}$ and $V \in R^{n \times n}$ are orthogonal matrices, and their columns are eigen-

vectors of XX^T and X^TX, respectively. Σ is a diagonal matrix of size X and it contains singular values of X in descending order. The nearest rank-k matrix to X under both the Frobenius and the spectral norm is obtained by taking the largest k eigenvalues and their corresponding eigenvectors [Deshpande and Rademacher, 2010].

Many previous studies tried to address the problem of column subset selection (also refereed to as volume sampling which was originally defined in [Deshpande et al., 2006]) where the objective is to be as close as possible to the best rank-k approximation computed via SVD. Although SVD can be computed in $O(min\{mn^2, nm^2\})$, it is still too slow for many applications. Accordingly, there is a large body of papers exploring the approximation techniques for that. For instance, in [Boutsidis et al., 2009], the authors provide a method to select k columns of a given matrix, $\mathbf{X}$, such that the probability of having $||\mathbf{X} - P_C\mathbf{X}||_F \leq \Theta(k \log^{1/2} k)\,||\mathbf{X} - \mathbf{X}_k||_F$ is at least 0.8, where $P_C\mathbf{X}$ denotes the projection of $\mathbf{X}$ onto the span of columns of $\mathbf{C}$ and $\mathbf{X}_k$ denotes the best rank-k approximation obtained via SVD. Their algorithm consists of two steps: a randomized step and a deterministic step. In the randomized step $\Theta(k \log k)$ columns of $\mathbf{X}$ is chosen, and in the deterministic step a strong rank-revealing QR factorization [Gu and Eisenstat, 1996] is employed to select exactly k columns of $\mathbf{X}$.

Our study is also different from this family of research because we are not interested in finding the low rank approximation (rank-k approximation) but we would like to find some real data points selected from a huge amount of data.

Representative object selection has been widely used in other domains such as chemistry, biology and chemometrics [Dean and Lewis, 1999,Brown and Martin, 1996, Higgs et al., 1997, Agrafiotis, 1997]. One of the most known approaches has been discussed in [Kennard and Stone, 1969] and is very similar to the one presented in [Arthur and Vassilvitskii, 2007b] which is used to initialize K-means (which is basically the idea of the K-means++ algorithm). To select k representative objects, the idea of [Kennard and Stone, 1969] is to first select the most central data object and then select the one which has the maximum distance to the first one. The next data points are selected based on the maximum distance to all previously selected ones. This procedure is continued until k data points are selected. As we will see in the following sections, one of the variants of the proposed method is basically using a similar approach for initialization and, thus, will provide a better solution with respect to [Kennard and Stone, 1969].

Another similar, but more efficient, technique has been introduced by Clark [Clark, 1997] and is called OptiSim. Similar to [Kennard and Stone, 1969], OptiSim first

selects the most central data point. Then, it selects one object at random and checks its distance to all previously selected ones; if the (average) distance is more than a parameter ϵ, then the object is added to the set, discarded otherwise. This process is repeated until k items are added to the set. The main disadvantage of OptiSim is that it is not completely parameter-free as one needs to determine a proper value for ϵ. Moreover, in one round of selecting candidates, we may not find k object (as we select candidates at random) and, as a result, a *recycle bin* needs to be considered in order to reselect the items if such case happens.

In [Daszykowski et al., 2002], the authors propose a clustering-based approach (more particularly, based on DBSCAN) to tackle the problem of finding representative object. In this approach, each object is labeled as a *core*, *border* or *outlier*, and based on those labels, clusters are built. Finally, a K-means is applied on the clusters provided by DBSCAN and the centers (which are not necessarily the data points) are then selected as representatives of the dataset. We will, in the experimental section, compare the results of the proposed method with a method inspired from [Daszykowski et al., 2002] and show that in some cases they perform similarly and in others the perform as complementary techniques for data representation.

4.3 Problem Statement, Complexity Analysis

As mentioned before, we would like to find a set of instances from a large dataset which can represent the entire data and provide a global view of it. To do that, we introduce a general problem, called Instance Selection Problem (ISP), and is defined as follows:

Definition 4.1 ***(The general ISP problem)*** *Let $\mathscr{L} = \{X_1, \cdots, X_m\}$ be the set of rows of matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ with X_i the i^{th} row of $\mathbf{X}$, and d a distance function. Then the general ISP problem is to find a subset $\mathbb{S} \subset \mathscr{L}$ with $|\mathbb{S}| = k$ and $1 \leq i \leq m$ such that:*

$$\underset{\mathbb{S} \subset \mathscr{L}}{\operatorname{argmin}} \sum_{x \in \mathscr{L} \setminus \mathbb{S}} d(x, \mathbb{S}) \tag{4.2}$$

As mentioned before, many rows (*i.e.* data instances) might be correlated and could b represented by a single row. Accordingly, we define matrix $\mathbf{C}_{m \times m}$ as the pair-wise similarity matrix where $\mathbf{C}_{ij}$ represents the similarity between i^{th} and j^{th} row of $\mathbf{X}$. Without loss of generality, we assume that $\mathbf{C} \in [0, 1]$ where 1 denotes the maximum

similarity and 0 denotes the minimum similarity. Clearly $\mathbf{C}' = 1 - \mathbf{C}$ represents the pairwise dissimilarities between rows of $\mathbf{X}$. Based on that, we have three variations of the ISP problem, namely ISP_{min}, $ISP_{average}$ and ISP_{max}. The *min* case tries to find some data points such that the sum of distances between each point and the closest point from the set of selected ones is minimized (like classic clustering algorithms), while the *max* case finds the points which has the maximum dissimilarity with all others. The *average* case finds the points which are in average close to all other data points. In the following, we formalize these problems:

Definition 4.2 *Given a data matrix* $\mathbf{X}$*, the set* $\mathscr{L}$*, a pairwise dissimilarity matrix* $\mathbf{C}'$ *and an integer* k*, we have the following* ISP *problems:*

$$ISP_{min}: \operatorname*{argmin}_{\mathbb{S}\subset\mathscr{L}} \sum_{x\in\mathscr{L}\setminus\mathbb{S}} \min_{x'\in\mathbb{S}} C'(x,x') \tag{4.3}$$

$$ISP_{average}: \operatorname*{argmin}_{\mathbb{S}\subset\mathscr{L}} \frac{1}{|\mathbb{S}|} \sum_{x\in\mathscr{L}\setminus\mathbb{S}} \sum_{x'\in\mathbb{S}} C'(x,x') \tag{4.4}$$

$$ISP_{max}: \operatorname*{argmin}_{\mathbb{S}\subset\mathscr{L}} \sum_{x\in\mathscr{L}\setminus\mathbb{S}} \max_{x'\in\mathbb{S}} C'(x,x') \tag{4.5}$$

such that $|\mathbb{S}| = k$.

In order to study the complexity of the problem stated in Def. 4.2, we use two well-known problem: K-medoids and weighted max-cut problem.

K-medoids problem: given a set of n data points W and a distance function d, the objective is to find a set of medoids, T, such that $|T| = k$ and sum of distances between each data point and its closest medoid, T_i, is minimized. More formally:

$$\operatorname*{argmin}_{T\subset W} \sum_{x\in W} d(x,T) \quad \text{with} \quad d(x,T) = \min_{y\in T} d(x,y) \tag{4.6}$$

By comparing Eq. (4.3) and Eq. (4.6), one can see that ISP_{min} is NP-complete as it is a simple, polynomial transformation from the K-medoids problem. With the same reasoning, we can simply prove that the ISP_{max} is also NP-complete since ISP_{max} is similar to ISP_{min} with a different distance function, i.e. $d(x,T) = \max_{x'\in\mathbb{S}} C'(x,x')$.

In order to study the complexity of $ISP_{average}$, we use graph partitioning problems, in particular those introduced in [Ageev and Sviridenko, 1999] and [Feige and Langberg, 2001]. The problem which is of interest here is the Maximum Cut problem with Given Sizes of Parts (MCGS).

MCGS: given a graph $G = (V, E)$ and nonnegative weights for edges w_{ij}, MCGS problem aims at finding a set $U \in V$ such that $|U| = k$ and $\sum_{i \in U, j \notin U} w_{ij}$ is maximized. As discussed in [Feige and Langberg, 2001], this problem (which is referred to as Max-Cut$_k$ therein) is NP-complete. It is easy to observe that having an instance $G = (V, E)$ of the general Max-Cut problem, we can create the following instances of the MCGS problem in polynomial time: $\{G = (V, E), k = 1\}, \cdots, \{G = (V, E), k = [n/2]\}$ where $n = |V|$. If we are able to find a polynomial solution for these instances, then we can find a polynomial solution for the Max-Cut problem, which is a contradiction unless $P = NP$. It is also very easy to see that $ISP_{average}$ is clearly a MCGS problem and, as a result, is NP-complete.

4.4 SOLUTIONS

4.4.1 MIN AND MAX CASE

As mentioned, in the previous section, in order to solve the ISP_{max} and ISP_{min}, one can use the K-medoids solvers such as Partitioning Around Medoids (PAM) [Kaufman and Rousseeuw, 1990]. In this case, we can apply PAM by providing it with either the similarity matrix (the *max* case) or dissimilarity (the *min* case) matrix. Algorithm 4 describes the PAM approach used to solve ISP_{min} and ISP_{max}. Without loss of generality, we assume that rows describe the objects and columns describe the attributes. Therefore, we aim at selecting k rows of the input matrix X as representatives. We also assume that the $C \in [0, 1]$ and, as a result, $C' = 1 - C \in [0, 1]$. In terms of complexity, one should keep in mind that in any similarity based method, we first need to construct the similarity or the dissimilarity matrix. Given the matrix of pairwise distances between the data points, for ISP_{min} and ISP_{max}, as shown in Algorithm 4, we need to substitute each centroid with an object and redo the object assignment until stabilization. Let Λ be the number of iterations before stabilization of centroids; then, the complexity of Algorithm 4 will be $O(M^2K\Lambda)$ as we have a loop over all the medoids (K times) and all the non-medoids (M times) in order to do the reassignments (M times).

Algorithm 4 PAM algorithm for ISP_{min} and ISP_{max}

Input: Similarity matrix C or distance matrix C', k

Output: k representative instances

```
select k rows of X
compute the total cost
repeat
    for all medoid m do
        for all non-medoid m' do
            swap m with m' and compute total cost
        end for
        replace m with m' having the minimum cost
    end for
until no change observed in the medoids
return medoids
```

4.4.2 AVERAGE CASE

In case of $ISP_{average}$, we propose a greedy algorithm which can efficiently find a solution of the optimization problem (4.4). As one can observe, we can transform the minimization problem, *i.e.* Eq. (4.4), into a maximization problem by simply replacing C' with C. The proposed greedy algorithm, which works on the maximization version of $ISP_{average}$, picks the instances one by one in order to build the final solution set $\mathbb{S}$. To do that, suppose that we have a set of "already selected" items, $\mathbb{S}_{-1}$, and we want to add the next item, s_l, to this set in order to make a larger set $\mathbb{S}$ (*i.e* $\mathbb{S} = \mathbb{S}_{-1} \cup s_l$). One can exploit the following formulation knowing that we can simply omit the averaging coefficient in the optimization problem:

$$
\begin{aligned}
&\underset{s_l \in \mathscr{L} \setminus \mathbb{S}_{-1}}{\operatorname{argmax}} \sum_{x' \in \mathscr{L} \setminus \mathbb{S}} \sum_{s \in \mathbb{S}} C(x', s) = \\
&\underset{s_l \in \mathscr{L} \setminus \mathbb{S}_{-1}}{\operatorname{argmax}} \sum_{x' \in \mathscr{L} \setminus \mathbb{S}} \Big(\sum_{s \in \mathbb{S}_{-1}} C(x', s) + C(x', s_l) \Big) = \\
&\underset{s_l \in \mathscr{L} \setminus \mathbb{S}_{-1}}{\operatorname{argmax}} \sum_{x' \in \mathscr{L} \setminus \mathbb{S}} \sum_{s \in \mathbb{S}_{-1}} C(x', s) + \sum_{x' \in \mathscr{L} \setminus \mathbb{S}} C(x', s_l) = \\
&\underset{s_l \in \mathscr{L} \setminus \mathbb{S}_{-1}}{\operatorname{argmax}} \underbrace{\sum_{\substack{x' \in \mathscr{L} \setminus \mathbb{S}_{-1} \\ x' \neq s_l}} \sum_{s \in \mathbb{S}_{-1}} C(x', s)}_{\alpha} + \underbrace{\sum_{x' \in \mathscr{L} \setminus \mathbb{S}} C(x', s_l)}_{\beta}
\end{aligned}
\tag{4.7}
$$

Algorithm 5 Greedy approach for $ISP_{average}$

Input: Similarity matrix C, k

Output: k representative instances

1: $\mathbb{S} \leftarrow \emptyset$
2: $\mathbb{S} \leftarrow \mathbb{S} \cup \underset{i}{\text{argmax}} \sum_{i=1}^{M} C_{ij}$
3: **repeat**
4: **for all** $x \in \mathbf{X} \backslash \mathbb{S}$ **do**
5: $\mathbb{A}(x) = \sum_{i=1}^{|\mathbb{S}|} C(x, \mathbb{S}_i)$
6: $\mathbb{B}(x) = 0$
7: **for all** $x' \in \mathbf{X} \backslash \mathbb{S}$, $x' \neq x$ **do**
8: $\mathbb{B}(x) += C(x, x')$
9: **end for**
10: $\mathbb{C}(x) = \mathbb{B}(x) - \mathbb{A}(x)$
11: **end for**
12: $\mathbb{S} \leftarrow \mathbb{S} \cup \text{argmax}\ \mathbb{C}(x_i)$
13: **until** $|\mathbb{S}| < k$
14: **return** $\mathbb{S}$

We can rewrite α as the following:

$$\alpha = \underbrace{\sum_{x' \in \mathscr{L} \backslash \mathbb{S}_{-1}} \sum_{s \in \mathbb{S}_{-1}} C(x', s)}_{does\ not\ depend\ on\ s_l} \underbrace{- \sum_{s \in \mathbb{S}_{-1}} C(s, s_l)}_{\gamma} \tag{4.8}$$

Accordingly, the terms to be maximized are β and γ:

$$\underset{s_l \in \mathscr{L} \backslash \mathbb{S}_{-1}}{\text{argmax}} \sum_{x' \in \mathscr{L} \backslash \mathbb{S}} C(x', s_l) - \sum_{s \in \mathbb{S}_{-1}} C(s, s_l) \tag{4.9}$$

Eq. (4.9) suggests, for the next point to be selected, the following greedy approach: in order to find the best point to add to the set, one can assign a score to all points and pick the point with the highest score. For each point, this score is simply computed as the sum of similarities it has with all non-selected points (β) minus sum of similarities it has with all already-selected points (γ). Algorithm 5 provides such greedy procedure. Note that first point added to the set is the one that is the most similar to all other points (line 2). As far as the complexity is concerned, given the pairwise similarity matrix, C, for each non-selected object, we need to consider its distance to all elements of $\mathbb{S}$ which results in a complexity of $O(MK)$. This said, the greedy approach is much faster than Algorithm 4.

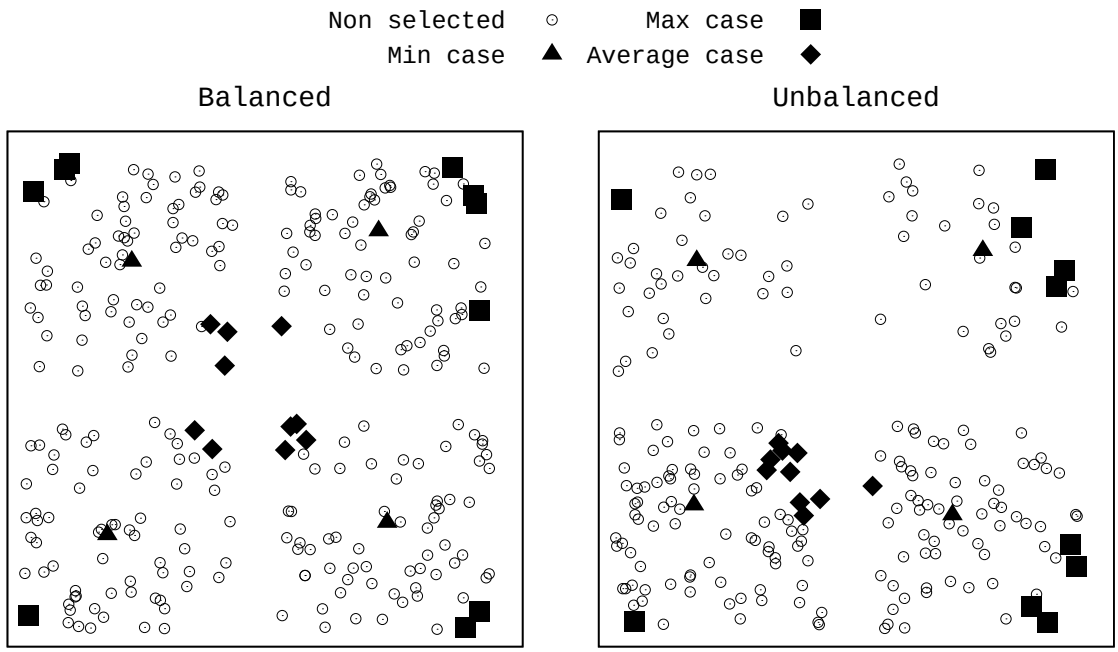

Figure 4.2: Data points selected by different *ISP* methods on synthetic data.

4.5 EXPERIMENTS

4.5.1 ILLUSTRATION ON SYNTHETIC DATA

As mentioned before, given a set of data points, we aim at selecting few instances in order to have a view of the entire data. Here, we first show how the algorithms discussed in Section 4.4 behave on a small, synthetic dataset. To better visualize the effect of *ISP* methods, we have shown in Figure 4.2, some random points distributed in four visual clusters each of which located in one of the quadrants of the 2D plane. Then, we considered 10 points to be selected for the *max*, 10 points for the *average* case, and 4 points for the *min* case as there are 4 clusters in the set. Figure 4.2 shows two cases for the synthetic data: balanced and unbalanced, where in the unbalanced case the two bottom regions contain more data points compared to those of top (and bottom-left more than bottom-right).

As one can see in the balanced case, the *average* method selects the most central instances, *i.e.* the points which are on the inner frontier of the clusters and are very close to other clusters. In this case, the points are distributed among all clusters while in the unbalanced case the points tend to be more biased to the clusters having more instances (the bottom-left cluster). One should note that these points are *not* simply the points the most similar to others (in which case it will be very easy to find them),

but are the points which are similar to other points *and* are dissimilar to each other (see Eq. (4.9)). The points having this property are called "central" points hereafter.

The *max* method, as we expect, selects the data points which are on the outer frontier of the clusters which denotes the fact that they are the least similar instances compared to their own cluster as well as the other clusters. The *min* case, which is a classical K-medoids approach, selects the centroids of each cluster.

This simple example shows how different ISP methods can select objects from different regions to provide a global view of the data. In the following, we will illustrate the effectiveness of the three ISP methods on Reuters data and will explain how the proposed algorithms behave on this dataset.

4.5.2 REUTERS DATA

We have visually shown, on synthetic data, how different solutions of the ISP problem work. However, we still need to study them on real world data. To do that, we use the Reuters dataset [Lewis et al., 2004] which is publicly available for research purposes. This dataset is called RCV1 and contains Reuters' English news collected from 20/08/1996 to 19/08/1997 where each entry is associated with at least one topic. There are around 800K news and more than 90 topics. To conduct our experiments, we adopted the following strategy: we first found the most popular topics, *i.e.* the topics which had more than 200 news containing only and only that topic. We then picked four popular topics and extracted all the news having that topic (and potentially other topics) in their metadata. Next, we removed the stop words, lemmatized the text and removed punctuation and digits. Finally, we filtered the terms which appeared less than 5 times. Table 4.1 illustrates the chosen topics as well as some information about them after the preprocessing step. We used normalized tf-idf as

Code	**Description**	**# Docs**	**# Terms**
C11	Strategy/Plans	24325	35142
C31	Markets/Marketing	40506	42002
E512	Merchandise trade	12634	19037
GDIP	International relations	37739	43167

Table 4.1: Topics used in the experiments.

the weighting and the cosine function as the similarity metric as they are widely used

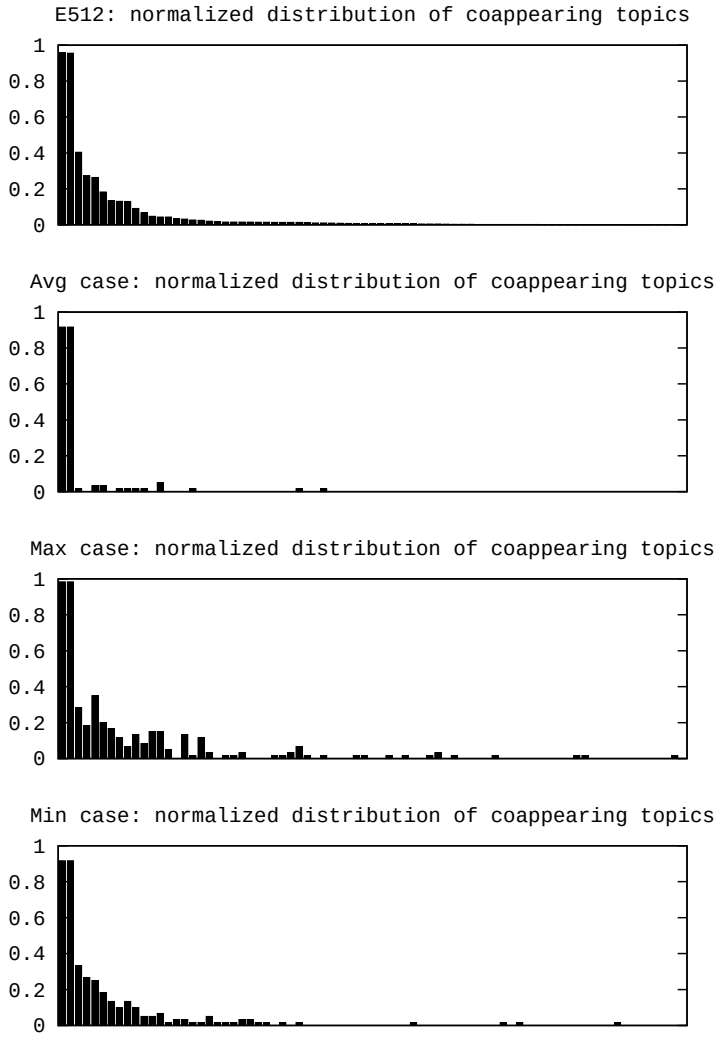

Figure 4.3: Comparison of the normalized distribution of topics coappearing with E512. The horizontal axis represents the topics (not labeled to increase the readability) and the vertical axis shows the values. Topics with less than five documents have been removed from the distributions.

in such contexts [Bilenko et al., 2003, Zhang et al., 2011]. In the following, we will discuss the data points selected by each ISP algorithm and explain how they can represent the entire data. To do that, we fix the number of documents to be selected to 60 (*i.e* 180 documents in total). The main reason is that if we select less documents, although ISP works well, other approaches (explained shortly) may not provide a good representation of the entire data.

We have already seen, thought Fig. 4.2, how ISP behaves on a small, synthetic data. Here, to illustrate the effect of each ISP technique on real data, we show the

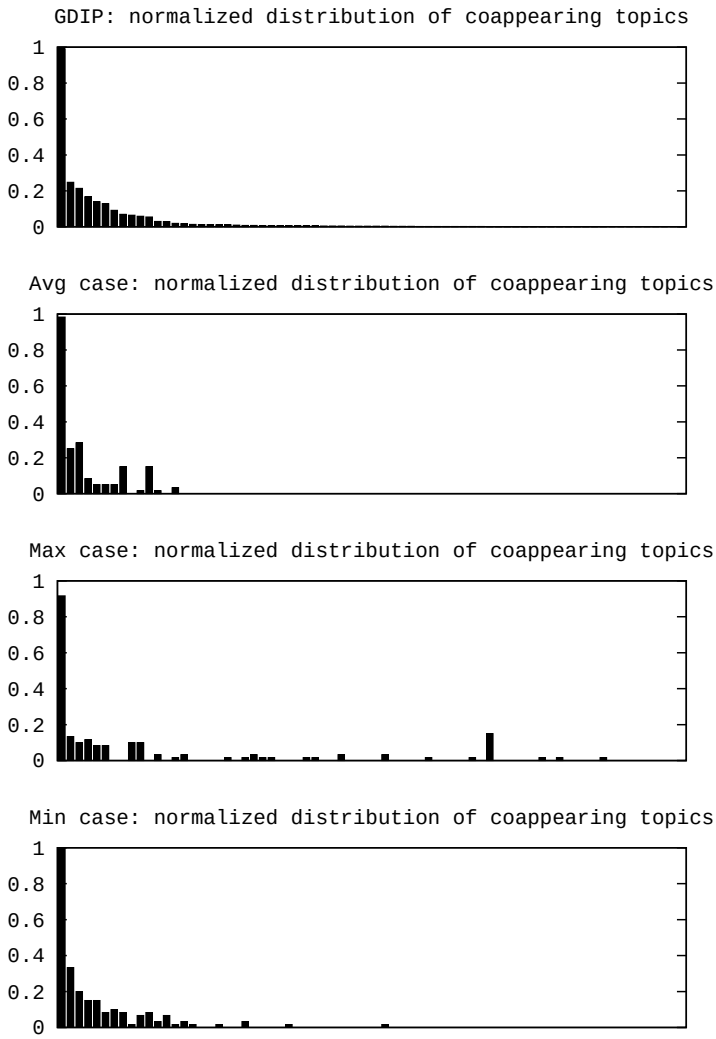

Figure 4.4: Comparison of the normalized distribution of topics coappearing with GDIP. The horizontal axis represents the topics (not labeled to increase the readability) and the vertical axis shows the values. Topics with less than five documents have been removed from the distributions.

normalized distribution of topics of the documents selected by each ISP method as well as that of the original set for two topics, namely E512 and GDIP, in Figure 4.3 and 4.4 respectively. Note that the topics are sorted (in descending order) based on their frequency of appearance in the original set and this order has been preserved for the ISP plots. We have also filtered the topics with less than 5 documents in order to avoid the very long tail in the distributions.

As one can see in Fig. 4.3, the *average* method selects the data points with a strong focus on the two most popular topics (E51 ECAT). The *max* case, however, has a long,

visible tail (*i.e.* selects the documents containing less frequent topics in the original set) as it tends to select the objects with a wide variety of topics. As it can be seen, the topics provided by the objects selected by the *max* technique are a lot less frequently seen in the original set (one can easily observe that by comparing the corresponding bars in the *max* case and in the first plot). This confirms that the ISP_{max} selects the documents which are far from others and contain less frequent topics. The *min* case is expected be located somewhere between the two others: it should, ideally, selects the documents containing not solely the "most popular" topics, but also some documents with "less popular" topics. That is the fact that one can see in Fig. 4.3 when it is compared to other approaches as well as to the distribution of original set. With the *min* approach, the distribution values of the first 5 "most frequent" topics is at least 0.2 and we still see some other topics further on the tail of distribution. Compared to the distribution of the original set, one can observe that *min* case is also able to highlight the topics which are in the medium level of "popularity" in the data. On the other hand, unlike the *max* case, it does not provide a vast range of topics including less frequent topics. This indicates that the *min* case has found data points situated between the *max* and the *average* case.

For GDIP, similar to E512, one can see that the documents of the *average* case focus on the most popular topics; that is why one observes the bars only on the left hand side of the second plot in Fig. 4.4. The documents of the *max* case covers, as usual, many topics while the *min* case falls between the two other cases.

To further evaluate the representativity of the documents selected by the ISP methods, we designed an approach inspired by [Daszykowski et al., 2002] as the method therein has shown better performance with respect to other methods such as [Kennard and Stone, 1969] and [Clark, 1997], and compared ISP to that.

In [Daszykowski et al., 2002], the authors propose to apply the DBSCAN on the dataset to find some clusters in the data and, then, run K-means within each cluster of DBSCAN in order to select the representatives (centers of K-means). The number of cluster in K-means is either a fixed number or is chosen proportional to the size of each DBSCAN cluster. The rational is that DBSCAN is able to capture the clusters with complex structures. Accordingly, once these structures are captured, we can apply a "center-based" technique, like $K-$means, to find the representatives (centers of the K-means in this case).

Inspiring from that, we propose the following experimental settings to evaluate the representatives selected by ISP: we first find a large set of candidates using DBSCAN

Topic	# Clusters	# Candidates (cores)	p	ϵ
C11	38	1869	25	0.40
C31	74	1850	10	0.05
E512	13	1804	40	0.40
GDIP	29	1926	30	0.15

Table 4.2: DBSCAN part of the DBPAM and DBISP.

and then use two different methods to select the representatives from them: *(i)* similar to [Daszykowski et al., 2002], we apply K-medoids (here we use PAM, *i.e* Algorithm 4) to select desired number of representatives (180) out of those candidates, and *(ii)* we apply the ISP techniques on the candidates and, then, compare the results. The main reason of using K-medoids is that, unlike K-means, the provided centroids are data points.

To select the candidates, we propose to run the DBSCAN with a grid search on different values of p and ϵ in order to find a large set of cores as candidates ($5 \leq p \leq 300$ with a step size of 5 and $0.05 \leq \epsilon \leq 0.70$ with a step size of 0.05). Here, we set the size of candidates to 10 time the size of representatives; *i.e.* 1800. One should note that it might not be possible to obtain exactly 1800 cores; accordingly, we used the results in which at least 1800 and at most 2000 cores are provided. In case where there were several similar possibilities (close number of cores), we selected the one with maximum number of clusters as it can better represent the data. Once we have the set of candidates, we compare the representatives selected by PAM with those selected by ISP. These two techniques are denoted as DBPAM and DBISP in our experiments as they are the combination DBSCAN with PAM and ISP respectively. Following this strategy, one can see a summary of the DBSCAN part of the DBPAM and DBISP in Table 4.2.

2D Illustrations

For sake of visualization, we used multidimensional scaling on the normalized tf-idf matrix and used the first two components to plot the figures. One can alternatively perform a PCA for this purpose [Daszykowski et al., 2002]. Fig. 4.5-4.8 show the representatives selected by each method for all four topics. To better illustrate the effects of DBPAM and DBISP, in these figures, we show each method separately as well as the combination of both methods. In the following, we see how these two techniques can

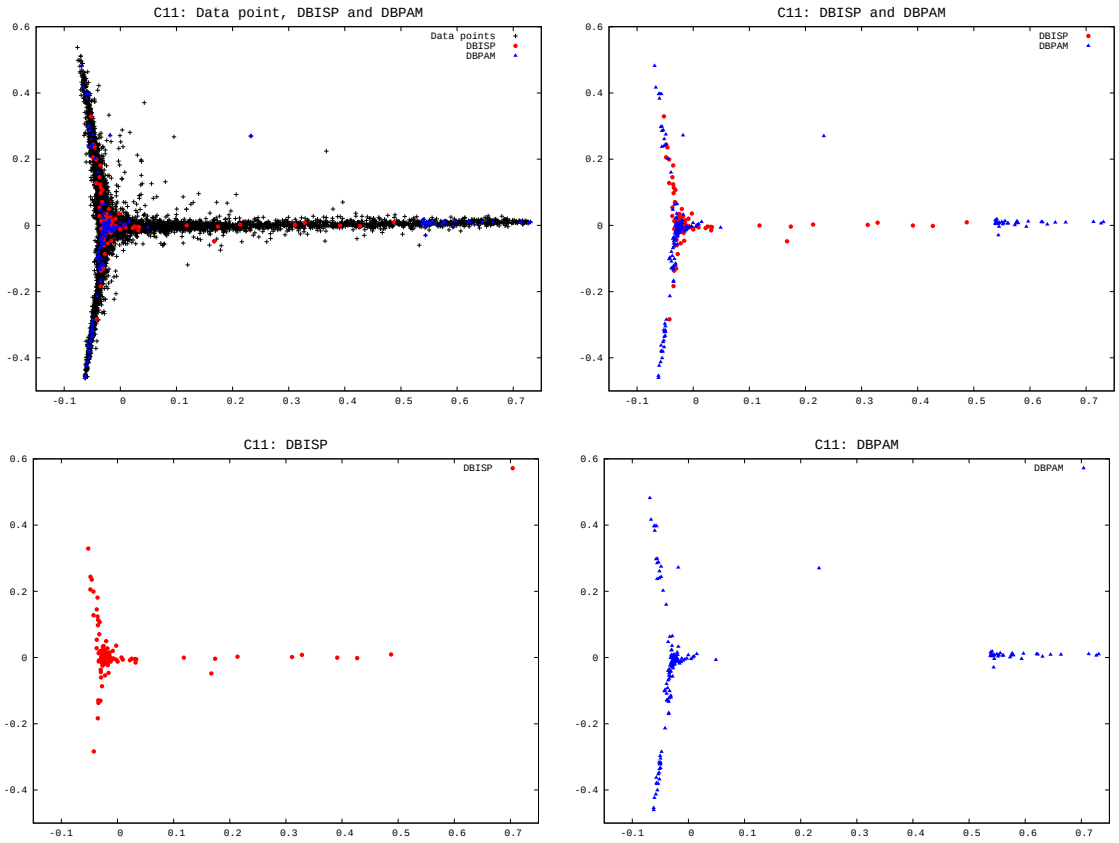

Figure 4.5: Comparison between DBPAM and DBISP on C11.

represent the data and why they should be considered as complementary methods.

C11: this case is a very good illustration for the complementariness of DBISP and DBPAM. As one can note in Fig. 4.5, DBPAM has difficulties covering the horizontal arm (see the huge white space in the center of DBPAM's figure). However, DBISP recovers this area with some representatives. On the other hand, DBISP covers only the middle of the vertical arm while DBPAM provides a good representation of the extreme parts of this arm. Accordingly, they act like two pieces of a puzzle which, together, provide a decent picture of the data.

C31: as one can note in Fig. 4.6, DBPAM covers well the region on the left and on the bottom; however, the region on the right is not very well covered as only some points have been selected on the rightmost part of this region. Nevertheless, DBISP has covered this area with several data points, making it a complement to DBPAM. As one can note in the figure, for C31, DBISP alone can also provide a very good representation of the data shape.

E512: in this case also, the two approaches seem to perform complementary as combining them will provide a shape similar to the original data as it can be observed in Fig. 4.7. As one can see, DBPAM fails to find sufficient data points on the top "arm"

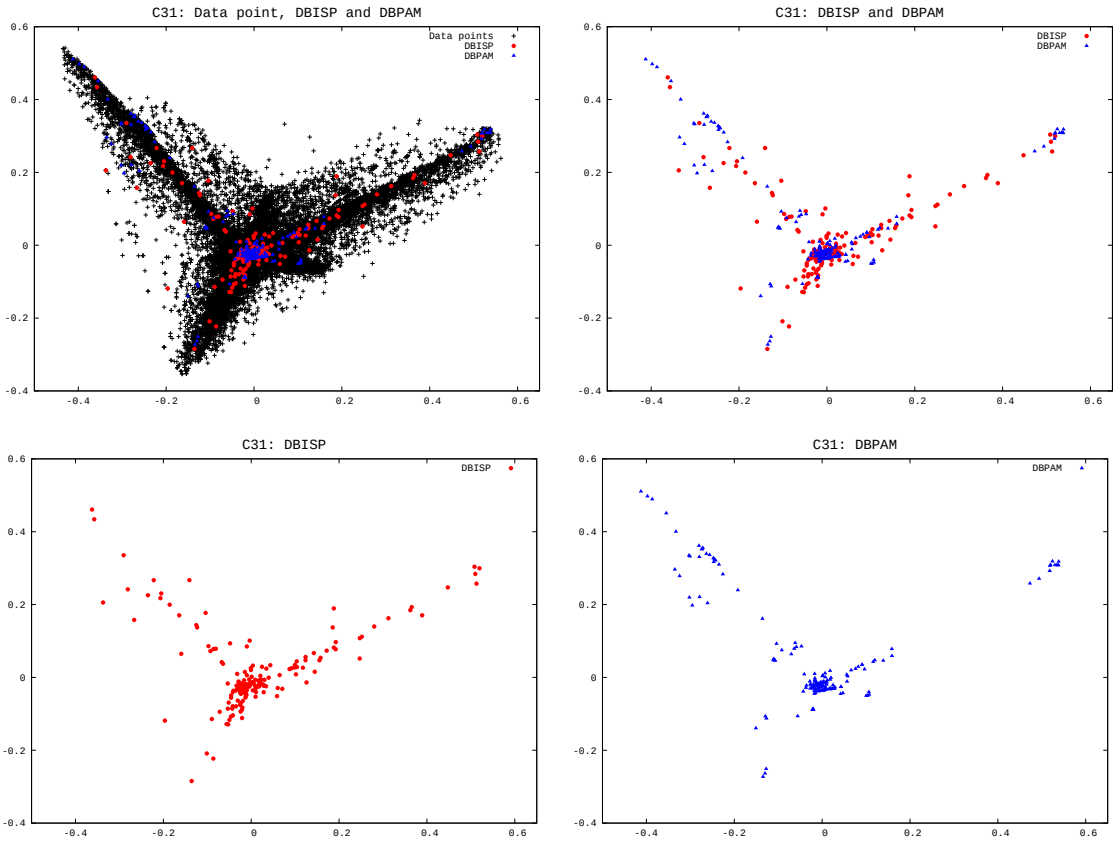

Figure 4.6: Comparison between DBPAM and DBISP on C31.

of the data; it only finds some points on the topmost area. However, DBISP seems to recover the missed region by selecting a few point located from the center to the top (*i.e.* covering this arm). On the other hand, the left-bottom part is better covered by DBPAM, which makes these two methods complementary.

GDIP: for this topic, both methods perform similarly on the arm located on bottom as well as the one located on the right hand side of the data. Nevertheless, DBPAM cannot cover the top arm properly; again, three points are selected in this arm and they are located in the far end region. However, DBISP can recover this missing area with several points (see Fig. 4.8), and, together with DBPAM, provides a good representation of the data shape.

Topic Analysis

So far, we have considered the effectiveness of DBISP and its complementariness with respect to the available methods (in particular DBPAM). In the following, we will consider the representativeness of the DBISP instances by considering each ISP method separately.

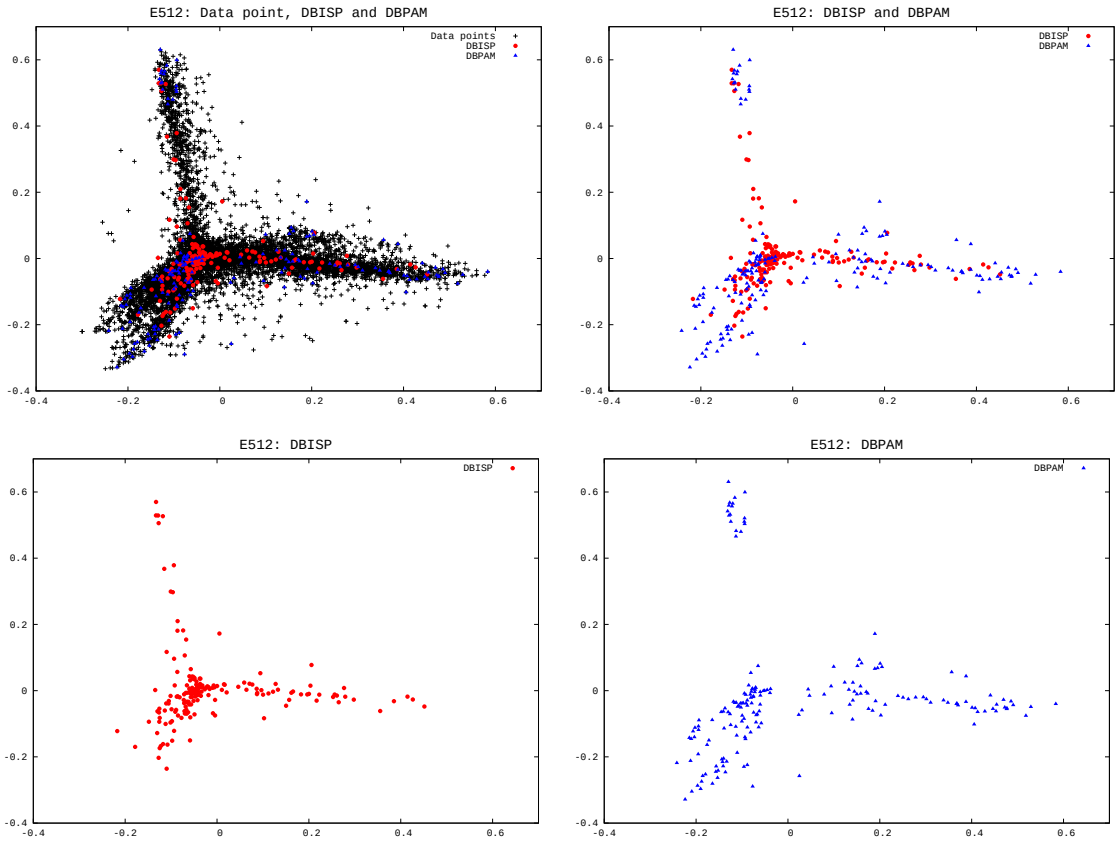

Figure 4.7: Comparison between DBPAM and DBISP on E512.

E512: firstly, one should note that the most coappearing topics for E512 is E51 and ECAT (95.5% of co-appearance). Considering that, one can see the effect of $ISP_{average}$: only 3 documents (out of 60) selected by this method does not contain these two topics. This is basically in line with the reasoning provided before: the average case finds the documents which have a lot in common with other documents. Consequently, we observe that the topics of the $ISP_{average}$ data points are widely seen among the dataset.

In the *max* case, the selected documents contain much more topics compared to the average or the *min* case. They also contain the topics which do not appear in the other approaches. For instance, one of the documents selected by the *max* approach contains C18 (*ownership changes*) which only appears with E512 in 39 documents (0.3%) and cannot be seen in the documents selected by the *min* or by the *average* approach. Another example of such cases is C24 (*capacity/facilities*) which coappears with E512 in 1.2% document. One should note that the documents selected by the *max* case are not outliers (as they are in a way filtered by DBSCAN beforehand), but are the points determining the shape of the data as they are located in the outer frontier (see Eq. (4.5)). In other words, they are the documents containing *less frequent* topics.

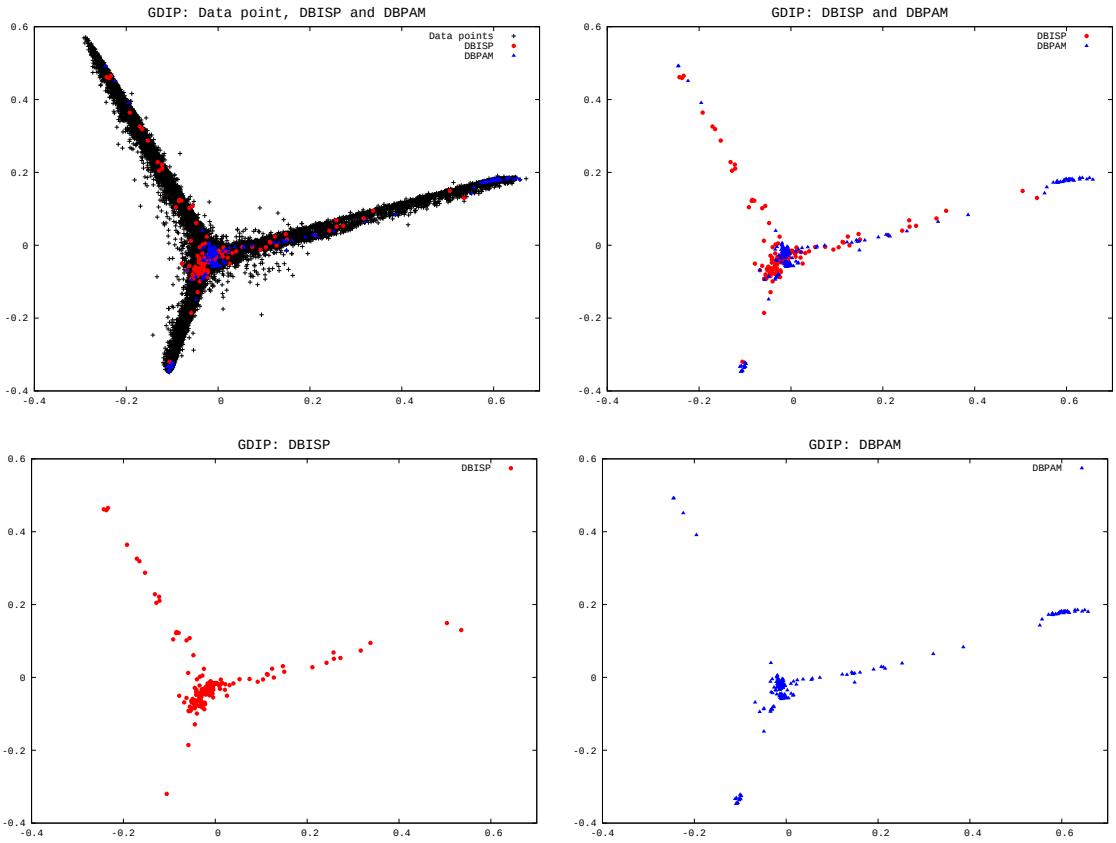

Figure 4.8: Comparison between DBPAM and DBISP on GDIP.

The documents of the *min* case, which are ideally the central points of the *sub-parts* of the data, are expected to be in between the *max* case and the *average* case. Accordingly, we observed that 90% of the data points selected by the *min* approach contain the frequent topics (E51 ECAT). In addition, in this case, there are also topics which are less common in the dataset but not as uncommon as the *max* case topics. For instance, one can observe the topic G15 (*European community*) which coappears with 14% of documents or GDIP (*international relations*) which coappears with 27.3% of the documents of E512. These results confirm that the *min* case selects the points which are neither in the outer border of the data nor the very central space but are in the center of different *sub-parts* of the data.

C31: the documents selected by the *max* case contain the topics which could not be found in those of the *min* or the *average* method. GJOB (*Labor issues*) and E41 (*Employment/Labor*) are two examples of such topics. To illustrate, one can observe that these two topics coappear with C31 in only 0.8% of documents in the entire set. Again, this is what expect from the *max* case: selecting the documents which are on the outer frontier of the data and potentially contain a wider range of topics. In the *min* case, we observe that the selected documents not only contain the most coappear-

ing topic, *i.e.* CCAT (*Corporate/Industrial*), but also contain other less likely topics. For instance C15 (*performance*) which coappears in 11% of documents and C151 (*accounts/earning*) which coappears in 4.2% documents. However, unlike the *max* case, the *min* case documents do not contain less-frequently-distributed topics. For the *average* case, as it selects the most central objects of the set, 40% of the documents contain only one coappearing topics: CCAT (*corporate/industrial*). This indicates that the selected documents are those focusing on the very main topics: 98.2% of C31 documents contain also CCAT; as a result, one can conclude that these documents are the very central ones in the dataset.

C11: In this case also, some less frequent topics appear in the *max* case. For example, we see C23 (*research/development*) which is highly rare to coappear with C11 (1.8% of co-appearance). In addition, in the *min* case, we see that the topics of the selected documents are the topics which are more likely to appear with C11. For instance, the documents selected by ISP_{min} contains the C15 (*performance*) with a co-appearance of 9.6%. C152 (*comment/forecasts*) is another example of this type which appeared with C11 in 7.5% of documents. For the *average* case, we observe that 28% of the documents selected by $ISP_{average}$ are the documents with only and only one coappearing topic: CCAT (*corporate/industrial*). As we expect, CCAT is a "always there" topic when we are considering *strategy/plans* (C11). This could be simply seen by looking at the proportion of C11 documents containing CCAT (97.4%). In the $ISP_{average}$ case, among 60 selected documents, there is only one document that does not contain CCAT. This point shows that the *average* method has chosen the most central documents in the dataset.

GDIP: perhaps this topic is a good example where the *max* approach can detect "not-easy-to-find" documents. In this topic, while the conclusion on the *min* and *average* methods are similar to the other topics, the conclusion on the *max* case is more contrasted. In the representatives selected by ISP_{max}, we find several topics which not only do not coappear with GDIP in *min* and *average* results, but also are too infrequent in the entire set. These topics coappear with GDIP in less than 1% of documents, which means that the documents containing these topics are infrequent and, as a result, hard to detect. However, the *max* approach was able to detect at least one document of each topic. Maybe the topic G151 (*EC international market*) is a good example here: only 166 documents contain G151 as a coappearing topic with GDIP and ISP_{max} is able to detect one of them. Another example is G154 (*EC monetary/economic*) which coappears with GDIP in 341 documents and ISP_{max} can detect two of them.

4.6 CLOSING REMARKS

In this chapter, as our fourth contribution [Mirisaee et al., 2015a], we have studied a new, alternative approach to select representative objects from different regions of big, high-dimensional datasets in order to provide a top, global view of the data. To do that, we defined the Instance Selection Problem (ISP) and its three variants. We have then shown how each of these variants can select representatives from different regions of the data. To show the behavior of the proposed approach, we first used a small, synthetic data for illustration purpose. We have then designed a set of experiments to show that using the DBISP technique, which is a combination DBSCAN and ISP, we can obtain good representatives on data. We then evaluated our results on Reuters dataset by extracting all documents of four popular topics and applying the DBISP on them. Inspired by the state-of-the-art techniques, we designed a method called DBPAM and then compared the results of DPISP with that of DBPAM. We argue that in some cases and some data regions DBISP provides a better representation of the data while in most case these two approaches should be considered as complementary techniques: where DBPAM is unable to cover a region of the data, DBISP does it and *vice versa*.

CONCLUSION

The goal of this thesis was to explore the problem of finding interesting and representative itemsets and items in large, high-dimensional datasets. To achieve this objective, we first studied the main challenges in this domain: the data is large in terms of number of objects/instances and number of attributes/features and may also contain noise and missing values. These factors make the mining process much more difficult. As a result, a wide range of data studies have been conducted to address these problems. One major technique in this area is data simplification. The idea of data simplification is to find better ways to represent the data such that better pieces of information can be extracted or better understanding of the data is provided. For instance, one can name feature selection algorithms as a data simplification technique where we are interested in selecting a subset of features that best represents the data.

As the data is usually represented via matrices, where rows are the instances and columns are the attributes, one important data simplification technique is matrix decomposition. Matrix decomposition is a method which factorizes a matrix into two or more smaller matrices, called factors, such the the multiplication of the factors approximates the original data. Matrix decomposition can extract the main information of the data and capture the latent factors. This technique has been widely used over the past few years and found a large number of applications such as text mining and image processing. Depending on the application, one may use different decomposition techniques. For instance, when the input matrix is nonnegative and the factors are also supposed to be nonnegative, one may use Non-negative Matrix Factorization (NMF).

Following this line of thought, we studied one particular case of matrix decomposition called Binary Matrix Factorization (BMF) in which the input and the factors are binary; the reconstructed matrix can be optionally binary as well. We first, in Chapter 1, went through the decomposition techniques available in the state-of-the-art before formalizing the BMF problem. We then presented the available methods solving the

BMF problem. As our first contribution, we established a link between the BMF problem and the Unconstrained Binary Quadratic Programming (UBQP) problem. This link allowed us to use the heuristics developed for UBQP, which we called *1-opt-UBQP*, in the case of BMF.

Then in Chapter 2, as our second contribution, we presented a new, efficient heuristic, called *1-opt-BMF*, to improve the BMF solutions. We first provided a theoretical analysis on the efficiency of this method with respect to *1-opt-UBQP* and the standard approach, called *1-opt-Standard*. Then, we have shown experimentally that the proposed heuristic is up to 8 times faster than the other methods and brings statistically significant improvement in many cases.

In Chapter 3, we discussed our third contribution where we introduce a new technique, called Decomposition Itemset Miner (DIM), in order to mine representative itemsets. Since in frequent itemset mining domain, the classical methods provide a huge number of itemsets, which are extremely hard to be analyzed by an analyst, the techniques providing good, representative itemsets have been of interest recently. As matrix decomposition techniques are able to capture the main structure and the latent factors of the data, we proposed to rely on these techniques in order to find representative itemsets. We first studied the existing methods in the literature, from classical algorithms to those aiming at finding a reduced number of itemsets. Then, we proposed to rely on BMF in order to mine representative itemsets and, to do that, we first established the theoretical link between BMF and frequent itemset mining. Once established, we designed a set of experiments to show the efficiency of the proposed method, as well as the quality of the itemsets produced. These experiments shows the itemsets found by DIM can well recover the data and are mostly precise.

Continuing on the problem of finding representative objects, we examined the problem of finding a subset of instances form datasets which can well represent different regions of the dataset. To do that, in Chapter 4, we formalized the Instance Selection Problem (ISP) and provided three variants of this problem, namely $ISP_{average}$, ISP_{min} and ISP_{max}. We then showed that all these three variants are NP-complete before providing solutions to them. For the $ISP_{average}$, we exploited the objective function to provide a greedy approach which selects one object at the time, and for the two others we used the classical PAM algorithm. Then, inspired by the state-of-the-art methods, we designed a technique, called DBPAM, which is a combination of DBSCAN and PAM, and compared that the another approach called DBISP, a combination DBSCAN and ISP. In our experiments, we showed that DBISP can be considered as

a complementary method for the existing technique: where other techniques cannot cover a region, DBISP does it and *vice versa*.

This thesis opens several research directions. Firstly, complexity of the BMF problem is still unknown, even for rank 1. In the literature, the complexity of some similar problems, such as weighted rank-1 BMF, has been considered; however, to the best of our knowledge, no complexity analysis has been presented for BMF. Although this problem is conjectured to be NP-hard in many studies, no formal proof of this fact has been proposed so far.

Another future research direction would be new methods for using BMF for frequent itemset mining problem. In this study, we presented the theoretical requirements and proved that BMF can be used for this application. However, one can further exploit the decomposition factors by characterizing the itemsets found by this method. For instance "Which part of the dataset do they mostly cover?" is an interesting question that can be further studied to better understand the usefulness of BMF in frequent itemset mining. Another possibility is to exploit *all* the itemsets provided in the second decomposition factor (**H** in this document). According to the theoretical analysis provided here, many itemsets are dropped due to the fact that the error parameter does not allow us to exploit them. However, this does not mean that they are totally useless. Using the properties of BMF, one can further study the characteristics of the rows of **H** to better recover the representative itemsets. Another interesting direction is to develop the theory of using *boolean* decomposition in order to mine representative itemsets.

As another interesting future work, one can further study different formulations of finding representative objects. In this document, to find representative objects, we have provided three problems, two of which rely on clustering methods. Another alternative would be using decomposition techniques and exploit the latent factors provided by those techniques. For instance, as matrix decomposition techniques can be considered as a 2-mode clustering method (also known as co-clustering or biclustering), one can consider the factors of the decomposition as the clusters of objects and the clusters of attributes. For instance, using NMF in a document-term matrix, each latent factor could be considered as a topic [Xu et al., 2003]; following this line of thought, can we select some documents from each topic and consider them as representatives? If so, which documents should be selected form each topic? What characteristics should they have?

PUBLICATIONS

1. Hamid Mirisaee, Ahlame Douzal, and Alexandre Termier. Finding Representative Instances from Dataset. Accepted to be published in the Proc. of IEEE International Conference on Data Science and Advanced Analytic (DSAA'15), 2015.

2. Vincent Leroy, Sihem Amer-Yahia, Eric Gaussier, and Hamid Mirisaee, Building Representative Composite Items, Accepted to be published in the Proc. of the 24th ACM International Conference on Information and Knowledge Management (CIKM'15), 2015

3. Hamid Mirisaee, Eric Gaussier, and Alexandre Termier. Efficient Local Search for L1 and L2 Binary Matrix Factorization. Accepted to be published in the Journal of Intelligent Data Analysis (IDA), 2015.

4. Hamid Mirisaee, Eric Gaussier, and Alexandre Termier. Local Heuristic Search for Binary Matrix Factorization. In Proceedings of the 29th AAAI conference on Artificial Intelligence (AAAI'15), 2015.

5. Hamid Mirisaee, Eric Gaussier, and Alexandre Termier. Itemset Approximation Using Constrained Binary Matrix Factorization. In Proceedings of IEEE International Conference on Data Science and Advanced Analytic (DSAA'14), 2014.

6. Francine Chen and Hamid Mirisaee. Do Topic-dependent Models Improve Microblog Sentiment Estimation?, In Proceedings of The 8th International AAAI Conference on Weblogs and Social Media (ICWSM'14), AAAI, 2014.

BIBLIOGRAPHY

[Afrati et al., 2004] Afrati, F., Gionis, A., and Mannila, H. (2004). Approximating a collection of frequent sets. In *KDD 04*, pages 12–19, Seattle, Washington, USA.

[Ageev and Sviridenko, 1999] Ageev, A. A. and Sviridenko, M. I. (1999). Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *Integer Programming and Combinatorial Optimization*, pages 17–30. Springer.

[Agrafiotis, 1997] Agrafiotis, D. K. (1997). Stochastic algorithms for maximizing molecular diversity. *Journal of chemical information and computer sciences*, 37(5):841–851.

[Agrawal and Shafer, 1996] Agrawal, R. and Shafer, J. C. (1996). Parallel mining of association rules. *IEEE Transactions on knowledge and Data Engineering*, 8(6):962–969.

[Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. of the 20th International Conference on Very Large Databases (VLDB 94)*, Chile.

[Aha et al., 1991] Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine learning*, 6(1):37–66.

[Alves et al., 2010] Alves, R., Rodriguez-Baena, D. S., and Aguilar-Ruiz, J. S. (2010). Gene association analysis: a survey of frequent pattern mining from gene expression data. *Briefings in Bioinformatics*, 11:210–224.

[Andrews and Patterson III, 1976] Andrews, H. C. and Patterson III, C. (1976). Singular value decomposition (svd) image coding. *Communications, IEEE Transactions on*, 24(4):425–432.

[Arthur and Vassilvitskii, 2007a] Arthur, D. and Vassilvitskii, S. (2007a). K-means++: The advantages of careful seeding. In *Proceedings of the* 18^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*.

[Arthur and Vassilvitskii, 2007b] Arthur, D. and Vassilvitskii, S. (2007b). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

[Bayardo Jr, 1998] Bayardo Jr, R. J. (1998). Efficiently mining long patterns from databases. In *ACM Sigmod Record*, volume 27, pages 85–93. ACM.

[Beasley, 1998] Beasley, J. E. (1998). Heuristic algorithms for the unconstrained binary quadratic programming problem. *London, UK: Management School, Imperial College*, 4.

[Ben-Hur and Guyon, 2003] Ben-Hur, A. and Guyon, I. (2003). Detecting stable clusters using principal component analysis. In *Functional Genomics*, pages 159–182. Springer.

[Bertsekas, 1976] Bertsekas, P. (1976). On the goldstein-levitin-polyak gradient projection method. *IEEE Transactions on Automatic Control*, 21:174–184.

[Bilenko et al., 2003] Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., and Fienberg, S. (2003). Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23.

[Billionnet and Sutter, 1994] Billionnet, A. and Sutter, A. (1994). Minimization of a quadratic pseudo-boolean function. *European Journal of Operational Research*, 78(1):106–115.

[Bittner, 1973] Bittner, E. (1973). *The functions of the police in modern society*. National Institute of Mental Health, Center for Studies of Crime and Delinquency.

[Blum and Langley, 1997] Blum, A. L. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271.

[Borgne and Bontempi, 2007] Borgne, A. M. Y. L. and Bontempi, G. (2007). New routes from minimal approximation error to principal components. *Neural Processing Letters*, 27(3):19–207.

[Boutsidis et al., 2009] Boutsidis, C., Mahoney, M. W., and Drineas, P. (2009). An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977. Society for Industrial and Applied Mathematics.

[Bringmann and Zimmermann, 2007] Bringmann, B. and Zimmermann, A. (2007). The chosen few: On identifying valuable patterns. In *ICDM*, pages 63–72.

[Brown and Martin, 1996] Brown, R. D. and Martin, Y. C. (1996). Use of structure-activity data to compare structure-based clustering methods and descriptors for use in compound selection. *Journal of Chemical Information and Computer Sciences*, 36(3):572–584.

[Cataldi et al., 2010] Cataldi, M., Di Caro, L., and Schifanella, C. (2010). Emerging topic detection on twitter based on temporal and social terms evaluation. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, page 4. ACM.

[Chaudhuri et al., 1994] Chaudhuri, D., Murthy, C., and Chaudhuri, B. (1994). Finding a subset of representative points in a data set. *IEEE transactions on systems, man, and cybernetics*, 24(9):1416–1424.

[Chen et al., 2002] Chen, B., Haas, P., and Scheuermann, P. (2002). A new two-phase sampling based algorithm for discovering association rules. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–468. ACM.

[Chen and Mirisaee, 2014] Chen, F. and Mirisaee, S. H. (2014). Do topic-dependent models improve microblog sentiment estimation? *Proceedings of ICWSM. AAAI*.

[Clark, 1997] Clark, R. D. (1997). Optisim: an extended dissimilarity selection method for finding diverse representative subsets. *Journal of Chemical Information and Computer Sciences*, 37(6):1181–1188.

[Daszykowski et al., 2002] Daszykowski, M., Walczak, B., and Massart, D. (2002). Representative subset selection. *Analytica Chimica Acta*, 468(1):91–103.

[Dean and Lewis, 1999] Dean, P. M. and Lewis, R. A. (1999). *Molecular diversity in drug design*. Springer.

[DeAngelis et al., 1995] DeAngelis, G. C., Ohzawa, I., and Freeman, R. D. (1995). Receptive-field dynamics in the central visual pathways. *Trends in neurosciences*, 18(10):451–458.

[Dendrinos et al., 1991] Dendrinos, M., Bakamidis, S., and Carayannis, G. (1991). Speech enhancement from noise: A regenerative approach. *Speech Communication*, 10(1):45–57.

[Deshpande and Rademacher, 2010] Deshpande, A. and Rademacher, L. (2010). Efficient volume sampling for row/column subset selection. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 329–338. IEEE.

[Deshpande et al., 2006] Deshpande, A., Rademacher, L., Vempala, S., and Wang, G. (2006). Matrix approximation and projective clustering via volume sampling. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126. ACM.

[Drineas et al., 2004] Drineas, P., Frieze, A., Kannan, R., Vempala, S., and Vinay, V. (2004). Clustering large graphs via the singular value decomposition. *Machine learning*, 56(1-3):9–33.

[DuMouchel et al., 1999] DuMouchel, W., Volinsky, C., Johnson, T., Cortes, C., and Pregibon, D. (1999). Squashing flat files flatter. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 6–15. ACM.

[Feige and Langberg, 2001] Feige, U. and Langberg, M. (2001). Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2):174–211.

[Fürnkranz and Flach, 2005] Fürnkranz, J. and Flach, P. A. (2005). Roc 'n'rule learning—towards a better understanding of covering algorithms. *Machine Learning*, 58:39–77.

[Glover F., 1998] Glover F., Kochenberger G. A., A. B. (1998). Adaptive memory Tabu search for binary quadratic programs. *Management Science*, 44.

[Golub and van Loan, 1996] Golub, G. and van Loan, C. (1996). *Matrix computations*, volume 3. Johns Hopkins University Press.

[Gu and Eisenstat, 1996] Gu, M. and Eisenstat, S. C. (1996). Efficient algorithms for computing a strong rank-revealing qr factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869.

[Guns et al., 2010] Guns, T., Nijssen, S., and Raedt, L. D. (2010). k-pattern set mining under constraints. Technical Report CW 596, KU Leuven.

[Gupta et al., 2000] Gupta, R., Fang, G., Field, B., Steinbach, M., and Kumar, V. (2000). Quantitative evaluation of approximate frequent pattern mining algorithms. In *KDD 08*, Las Vegas, Nevada, USA.

[Han et al., 2007] Han, J., Cheng, H., Xin, D., and Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86.

[Han et al., 2000] Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM.

[Hasewaga et al., 1993] Hasewaga, S., Imai, H., Inaba, M., Katoh, N., and Nakano, J. (1993). Efficient algorithms for vairance-based k-clustering. In *Proceedings of the first Pacific conference on Computer Graphics Applications*.

[Helmberg and Rendl, 1998] Helmberg, C. and Rendl, F. (1998). Solving quadratic (0, 1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82(3):291–315.

[Higgs et al., 1997] Higgs, R. E., Bemis, K. G., Watson, I. A., and Wikel, J. H. (1997). Experimental designs for selecting molecules from large chemical databases. *Journal of chemical information and computer sciences*, 37(5):861–870.

[Hobohm et al., 1992] Hobohm, U., Scharf, M., Schneider, R., and Sander, C. (1992). Selection of representative protein data sets. *Protein Science*, 1(3):409–417.

[Holmes et al., 2007] Holmes, M., Gray, A., and Isbell, C. (2007). Fast svd for large-scale matrices. In *Workshop on Efficient Machine Learning at NIPS*.

[Huan and Motoda, 2002] Huan, L. and Motoda, H. (2002). On issues of instance selection. *Data Mining and Knowledge Discovery*, 6(2):115–130.

[Inokuchi et al., 2000] Inokuchi, A., Washio, T., and Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23.

[Jiang et al., 2014] Jiang, P., Peng, J., Heath, M., and Yang, R. (2014). A clustering approach to constrained binary matrix factorization. In *Data Mining and Knowledge Discovery for Big Data*, pages 281–303. Springer.

[Johnson et al., 1988] Johnson, D. S., Papadimitriou, C. H., and Yannakakis, M. (1988). How easy is local search? *Journal of computer and system sciences*, 37(1):79–100.

[Jolliffe, 2002] Jolliffe, T. (2002). *Principal Component Analysis*. Springer, 2nd edition.

[Junco et al., 2013] Junco, R., Elavsky, C. M., and Heiberger, G. (2013). Putting twitter to the test: Assessing outcomes for student collaboration, engagement and success. *British Journal of Educational Technology*, 44(2):273–287.

[Kanungo et al., 2002] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. (2002). A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18. ACM.

[Katayama et al., 2000] Katayama, K., Tani, M., and Narihisa, H. (2000). Solving large binary quadratic programming problems by effective genetic local search algorithm. In *GECCO*, pages 643–650.

[Kaufman and Rousseeuw, 1990] Kaufman, L. and Rousseeuw, P. J. (1990). Partitioning around medoids (program pam). *Finding groups in data: an introduction to cluster analysis*, pages 68–125.

[Kennard and Stone, 1969] Kennard, R. W. and Stone, L. A. (1969). Computer aided design of experiments. *Technometrics*, 11(1):137–148.

[Kernighan and Lin, 1970] Kernighan, B. and Lin, S. (1970). An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell Systems Technical Journal*, 49(2).

[Kivinen and Mannila, 1994] Kivinen, J. and Mannila, H. (1994). The power of sampling in knowledge discovery. In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 77–85. ACM.

[Koren et al., 2009a] Koren, Y., Bell, R., and Volinsky, C. (2009a). Matrix factorization techniques for recommender systems. *Computer*, pages 30–37.

[Koren et al., 2009b] Koren, Y., Bell, R., and Volinsky, C. (2009b). Matrix factorization techniques for recommender systems. *IEEE Computer Society*, 42(8):30–37.

[Koyuturk and Grama, 2003] Koyuturk, M. and Grama, A. (2003). Proximus: A framework for analyzing very high-dimensional discrete-attributed datasets. *ACM SIGKDD*, pages 147–156.

[Koyuturk et al., 2005] Koyuturk, M., Grama, A., and Ramakrsihnan, N. (2005). Compression, clustering, and pattern discovery in very-high-dimensional discrete-attribute data sets. *IEEE Trans. Knowledge Data Eng*, 17:447–461.

[Krajca et al., 2011] Krajca, P., Outrata, J., and Vychodil, V. (2011). Using frequent closed itemsets for data dimensionality reduction. In *ICDM*.

[Langville et al., 2014] Langville, A. N., Meyer, C. D., Albright, R., Cox, J., and Duling, D. (2014). Algorithms, initializations, and convergence for the nonnegative matrix factorization. *arXiv preprint arXiv:1407.7299*.

[Lee and Seung, 1999a] Lee, D. and Seung, H. (1999a). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791.

[Lee and Seung, 2001] Lee, D. and Seung, H. (2001). Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562.

[Lee and Seung, 2000] Lee, D. and Seung, S. (2000). Algorithms for non-negative matrix factorization. In *In NIPS*, pages 556–562. MIT Press.

[Lee and Seung, 1999b] Lee, D. D. and Seung, H. S. (1999b). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791.

[Leroy et al., 2015] Leroy, V., Amer-Yahia, Sihem adn Gaussier, E., and Mirisaee, H. (Accepted to be published in the Proceedings of CIKM'15, 2015). Building representative composite items.

[Lewis et al., 2004] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397.

[Li, 2005] Li, T. (2005). A general model for clustering binary data. *ACM SIGKDD*, pages 188–197.

[Lin, 2005] Lin, C. (2005). Projected gradient methods for non-negative matrix factorization. *Technical report, National Taiwan University*.

[Lin, 2007] Lin, C.-J. (2007). Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779.

[Lin and Kernighan, 1973] Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.

[Liu and Motoda, 1998] Liu, H. and Motoda, H. (1998). *Feature extraction, construction and selection: A data mining perspective*. Springer.

[Mahoney and Drineas, 2009] Mahoney, M. W. and Drineas, P. (2009). Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702.

[Mairal et al., 2010] Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2010). Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11:19–60.

[Mall et al., 2014] Mall, R., Jumutc, V., Langone, R., and Suykens, J. A. (2014). Representative subsets for big data learning using k-nn graphs. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 37–42. IEEE.

[Mall et al., 2013] Mall, R., Langone, R., and Suykens, J. A. (2013). Furs: Fast and unique representative subset selection retaining large-scale community structure. *Social Network Analysis and Mining*, 3(4):1075–1095.

[Meedsa et al., 2006] Meedsa, E., Ghahramani, Z., Neal, R., and Roweis, S. (2006). Modeling dyadic data with binary latent factors. *NIPS*, pages 977–984.

[Mehrotra, 1997] Mehrotra, A. (1997). Cardinality constrained boolean quadratic polytope. *Discrete Applied Mathematics*, 79(1):137–154.

[Merz and Freisleben, 2002] Merz, P. and Freisleben, B. (2002). Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics*, 8(2):197–2013.

[Miettinen, 2008a] Miettinen, P. (2008a). The boolean column and column-row matrix decompositions. *Data Mining and Knowledge Discovery*, 17(1):39–56.

[Miettinen, 2008b] Miettinen, P. (2008b). The boolean column and column-row matrix decompositions. *Data Mining and Knowledge Discovery*, 17(1):39–56.

[Miettinen, 2008c] Miettinen, P. (2008c). On the positive–negative partial set cover problem. *Information Processing Letters*, 108(4):219–221.

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

[Milligan and Cooper, 1985] Milligan, G. W. and Cooper, M. C. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179.

[Mirisaee et al., 2015a] Mirisaee, H., Douzal, A., and Termier, A. (Accepted to be published in the Proc. to DSAA '15, 2015a). Finding representative instances from dataset.

[Mirisaee et al., 2014] Mirisaee, H., Gaussier, E., and Termier, A. (2014). Itemset approximation using constrained binary matrix factorization. In *Conference on Data Science and Advanced Analytics (DSAA)*, pages 39–45.

[Mirisaee et al., 2015c] Mirisaee, H., Gaussier, E., and Termier, A. (2015c). Local heuristic search for binary matrix factorization. In *Proceedings of AAAI*.

[Mirisaee et al., 2015b] Mirisaee, H., Gaussier, E., and Termier, A. (Accepted to be published in the IDA journal, 2015b). Efficient local search for L1 and L2 binary matrix factorization.

[Monga and Mıhçak, 2007] Monga, V. and Mıhçak, M. K. (2007). Robust and secure image hashing via non-negative matrix factorizations. *Information Forensics and Security, IEEE Transactions on*, 2(3):376–390.

[Paatero, 1999] Paatero, P. (1999). The multilinear engine—a table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *Journal of Computational and Graphical Statistics*, 8(4):854–888.

[Paatero and Tapper, 1994a] Paatero, P. and Tapper, U. (1994a). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126.

[Paatero and Tapper, 1994b] Paatero, P. and Tapper, U. (1994b). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5:111–126.

[Pak and Paroubek, 2010] Pak, A. and Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 1320–1326.

[Pang and Lee, 2008] Pang, B. and Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135.

[Pasquier et al., 1999] Pasquier, N., Bastide, Y., Taouil, R., L., and Lakhal (1999). Discovering frequent closed itemsets for association rules. In *Proc. of 7th international conference of database theory (ICDT 99)*, pages 398–416, Jerusalem, Israe.

[Pauca et al., 2006] Pauca, V. P., Piper, J., and Plemmons, R. J. (2006). Nonnegative matrix factorization for spectral data analysis. *Linear algebra and its applications*, 416(1):29–47.

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Pei et al., 2000] Pei, J., Han, J., and Mao, R. (2000). CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. of ACM SIGMOD 00, Workshop on Research Issues in Data Mining and Knowledge Discovery*.

[Piper et al., 2004] Piper, J., Pauca, V. P., Plemmons, R. J., and Giffin, M. (2004). Object characterization from spectral data using nonnegative factorization and information theory. In *Proceedings of AMOS Technical Conference*.

[Praks et al., 2003] Praks, P., Snasel, V., Dvorsky, J., and Cernohorsky, J. (2003). On svd-free latent semantic indexing for image retrieval for application in a hard industrial environment. In *Industrial Technology, 2003 IEEE International Conference on*, volume 1, pages 466–471. IEEE.

[Rymon, 1992] Rymon, R. (1992). Search through systematic set enumeration. *Technical Reports (CIS)*, page 297.

[Sadasivan and Dutt, 1996] Sadasivan, P. and Dutt, D. N. (1996). Svd based technique for noise reduction in electroencephalographic signals. *Signal Processing*, 55(2):179–189.

[Sandler and Lindenbaum, 2011] Sandler, R. and Lindenbaum, M. (2011). Nonnegative matrix factorization with earth mover's distance metric for image analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(8):1590–1602.

[Shen et al., 2009] Shen, B., Ji, S., and Ye, J. (2009). Mining discrete patterns via binary matrix factorization. *ACM SIGKDD*, pages 757–766.

[Soukup and Bajla, 2008] Soukup, D. and Bajla, I. (2008). Robust object recognition under partial occlusions using nmf. *Computational Intelligence and Neuroscience*, 2008.

[Tu et al., 2011] Tu, S., Chen, R., and Xu, L. (2011). A binary matrix factorization algorithm for protein complex prediction. *Proteome science*, 9(Suppl 1):S18.

[Uno et al., 2004] Uno, T., Kiyomi, M., and Arimura, H. (2004). Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, volume 126.

[Uno et al., 2005] Uno, T., Kiyomi, M., and Arimura, H. (2005). Lcm ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 77–86. ACM.

[Varmuza, 1980] Varmuza, K. (1980). *Lecture Notes in Chemistry*. Springer.

[Virtanen, 2007] Virtanen, T. (2007). Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(3):1066–1074.

[Vreeken et al., 2011] Vreeken, J., van Leeuwen, M., and Siebes, A. (2011). Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214.

[Wang et al., 2005] Wang, J. T.-L., Zaki, M. J., Toivonen, H., and Shasha, D., editors (2005). *Data Mining in Bioinformatics*. Springer.

[Witten et al., 2011] Witten, H., Frank, E., and A.Hall (2011). *Data mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3rd edition.

[Witten and Frank, 2005] Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[Wold et al., 1987] Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1):37–52.

[Xin et al., 2005] Xin, D., Han, J., Yan, X., and Cheng, H. (2005). Mining compressed frequent-pattern sets. In *Proceedings of the 31st international conference on Very large data bases*, pages 709–720. VLDB Endowment.

[Xu et al., 2003] Xu, W., Liu, X., and Gong, Y. (2003). Document clustering based on non-negative matrix factorization. In *Proc. of SIGIR*, pages 267–273.

[Zhang et al., 2005] Zhang, D., Chen, S., and Zhou, Z.-H. (2005). A new face recognition method based on svd perturbation for single example image per person. *Applied Mathematics and computation*, 163(2):895–907.

[Zhang and Li, 2010] Zhang, Q. and Li, B. (2010). Discriminative k-svd for dictionary learning in face recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2691–2698. IEEE.

[Zhang et al., 2011] Zhang, W., Yoshida, T., and Tang, X. (2011). A comparative study of tf* idf, lsi and multi-words for text classification. *Expert Systems with Applications*, 38(3):2758–2765.

[Zhang et al., 2007] Zhang, Z., Li, T., Ding, C., Ren, X., and Zhang, X. (2007). Binary matrix factorization with applications. *ICDM*, pages 391–400.

[Zhang et al., 2010] Zhang, Z., Li, T., Ding, C., Ren, X. W., and Zhang, X. (2010). Binary matrix factorization for analyzing gene expression data. *Data Mining and Knowledge Discovery*, 20(1):28–52.

[Zhou et al., 2009] Zhou, T., Su, R.-Q., Liu, R.-R., Jiang, L.-L., Wang, B.-H., and Zhang, Y.-C. (2009). Accurate and diverse recommendations via eliminating redundant correlations. *New Journal of Physics*, 11(12):123008.

[Zhu et al., 2014] Zhu, L., Galstyan, A., Cheng, J., and Lerman, K. (2014). Tripartite graph clustering for dynamic sentiment analysis on social media. In *Proceedings*

of the 2014 ACM SIGMOD international conference on Management of data, pages 1531–1542. ACM.

[Zou et al., 2010] Zou, J., Xiao, J., Hou, R., and Wang, Y. (2010). Frequent instruction sequential pattern mining in hardware sample data. In *ICDM*, pages 1205–1210.