Efficient local search for L_1 and L_2 binary matrix factorization

Hamid Mirisaee, Eric Gaussier, Alexandre Termier

Efficient Local Search for L_1 and L_2 Binary Matrix Factorization

Hamid Mirisaee*[1], Eric Gaussier[1] and Alexandre Termier†[2]

[1]University of Grenoble Alps/CNRS, France
[2]University of Rennes I, France

Abstract

Rank K Binary Matrix Factorization (BMF) approximates a binary matrix by the product of two binary matrices of lower rank, K. 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. The rank 1 problem (for which the L_1 and L_2-norms are equivalent) has been shown to be related to the Integer Linear Programming (ILP) problem. We first show here that the alternating strategy with the L_2-norm, at the core of several methods used to solve BMF, can be reformulated as an Unconstrained Binary Quadratic Programming (UBQP) problem. This reformulation allows us to use local search procedures designed for UBQP in order to improve the solutions of BMF. We then introduce a new local search dedicated to the BMF problem. We show in particular that this solution is in average faster than the previously proposed ones. We then assess its behavior on several collections and methods and show that it significantly improves methods targeting the L_2-norms on all the datasets considered; for the L_1-norm, the improvement is also significant for real, structured datasets and for the BMF problem without the binary reconstruction constraint.

Keywords– Binary Matrix Factorization, Local Search, Heuristic

*Corresponding author: Hamid.Mirisaee@imag.fr, 110 Ave. de la Chimie, BP 53, 38041 Grenoble, France. Tel: +33 4 56 52 03 20, Fax: +33 4 76 51 42 52

†Work done at University of Grenoble Alps/CNRS, France

1 Introduction

Many datasets of interest for scientific or industrial applications are high dimensional binary matrices. The high dimensionality negatively impacts the performance of traditional data analysis algorithms such as clustering or association rule mining. Matrix factorization is a way to compress the data while preserving most of the characteristic patterns found inside. When the input matrix and the factors are both binary matrices, the operation is called a Binary Matrix Factorization (BMF). BMF has been successfully applied in the context of gene expression analysis [23, 24], digit reconstruction [16], document clustering [13, 24] and frequent itemset mining [20].

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 takes the following form:

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

where $||.||_p$ denotes either the L_1-norm ($p = 1$) or the L_2-norm ($p = 2$). In this paper, we denote the L_1 and L_2 cases as **$\mathbf{L_1}$-BMF** and **$\mathbf{L_2}$-BMF** respectively. 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-BMF, efficient approaches usually solve a relaxed version of the problem, e.g. through non-negative matrix factorization (NMF) or singular value decomposition (SVD), and then project the solution into the admissible domain. 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 [9, 10]. 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-BMF, the most efficient approaches, to our knowledge, consider the problem as a clustering problem and make use of a variant of the K-means algorithm to solve it [6]. These related studies are presented in more detail in Section 2 of this paper.

Previous studies have shown that computing a rank 1 ($K = 1$) approximation could be reformulated as a $0-1$ integer linear programming problem [21]. They exploit this reformulation to guarantee the computation of an optimal solution on small matrices, and through relaxations provide approximations with a bounded error rate on larger matrices.

Our first contribution, presented in Section 3, is to show that in the

general case (rank $K \geq 1$), L_2-BMF can be reformulated as an *Unconstrained Binary Quadratic Programming* (UBQP) problem [2]. The relation to UBQP suggests the use of *p-opt* local search procedures developed in that context for L_2-BMF. We explore this direction and, as our second contribution, we propose (in Section 4) a new *p-opt* local search, which can be applied to both L_1- and L_2-BMF, that we prove to be more efficient comparing to other approaches.

With thorough experiments on both synthetic and real data, presented in Section 5, we demonstrate that the heuristic we propose significantly improves the quality of existing methods in a reasonable amount of time. We also show that this local search procedure fares well when compared to other heuristic strategies [7].

2 Related work

The optimization problem (1) can be seen as an NMF [11, 12] problem with additional binary constraints. Two main approaches have been followed along this line. The first one (used in [19]) amounts to first solve a standard NMF problem and then to project the solution into 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 pre-defined value, as 0.5. Learning the thresholds, even though more costly, usually leads to better results [19]. The second approach (used in [23] and [24] for example) amounts to integrating, through regularization terms, the binary constraints into the objective function being minimized in NMF, leading to:

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

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. 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 [18] for example, for a similar problem on boolean, and not binary, matrix decomposition).

A problem similar to (1) is formulated in [13]. In this latter work, however, a third, non-binary matrix is introduced, making the problem addressed differently from the one we are interested in. This said, the approach developed in [13] bears strong similarities with the ones used for solving Problem (1).

A simple and efficient heuristics to solve Problem (1) is the one at the basis of the Proximus system [9, 10]. In this approach, one iteratively solves rank 1 decompositions and combines them to form the global solution. The rank 1 problems are solved through an alternate rule that fixes either $\mathbf{w}$ or $\mathbf{h}$ (we use here a vector notation as $\mathbf{W}$ and $\mathbf{H}$ reduce to vectors for $K = 1$) and finds the best $\mathbf{w}$ or $\mathbf{h}$; this can be done efficiently by noticing that, given $\mathbf{w}$, the element h_l should be 1 if at least half of the non-zero elements of $\mathbf{w}$ are covered in $\mathbf{X}_{.l}$ (the l^{th} column of $\mathbf{X}$). In between each iteration, the original matrix $\mathbf{X}$ is decomposed and a new rank 1 solution is searched on a subpart of it. As the presence vectors (*i.e.* the different $\mathbf{h}$ vectors) are orthogonal to each other, the global solution, obtained by concatenating the different rank 1 solutions, satisfies all the constraints in (1).

In a more recent work, Shen et. al. [21] show that the rank 1 binary matrix factorization problem can be formulated as a $0-1$ integer linear program (ILP) through the maximum weight problem. They then study a continuous relaxation of this problem, as well as a minimum $s-t$ cut approximation that can be solved with maximum flow algorithms. Finally, they use this approach as an initialization step for Proximus.

These two latter studies, with a focus on rank 1 decompositions, are suitable for both L_1- and L_2-BMF. The study recently presented in [6] provides an original approach, called Constrained Binary Matrix Factorization (CBMF), for L_1-BMF. In this work, the complete problem (with the binary reconstruction constraint) is addressed via $K-$means clustering, by selecting initial cluster representatives from column vectors of $\mathbf{X}$ and reassigning each column vector to the class of the closest representative. New representatives are computed at each iteration, and the process is repeated until the clusters do not evolve anymore. 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. Using results from [5], the authors furthermore show that their approach yields a $4(2 + \log K)$-approximation of the optimal solution. A variant of this problem, called Unconstrained Binary Matrix factorization (UBMF), has been also presented in [6] for the unconstrained case.

The NMF problem as well as the $K-$means clustering one are known to be NP-hard [1, 3]. A related problem to the one considered here, namely the weighted rank 1 binary matrix factorization, is also shown to be NP-complete in [15]. This problem, defined for $K = 1$, however differs from ours in that the errors made in the reconstruction (having a 1 instead of a 0 or a 0 instead of a 1) are weighted differently (they have the same weight, 1, in

our formulation). Even though this difference may seem marginal, it suffices to obtain a reduction from the maximum edge weight biclique problem, and the reduction proposed in [15] can not be directly applied to our problem. In fact, even though Problem (1) has been claimed in several places to be NP-hard, we know of no formal proof for this fact. Several relations, as the one we display here in Section 3, to NP-hard problems have been established; however, no clear reduction from an NP-hard problem has been shown to our knowledge. This said, no proof that the problem can be solved in polynomial time has been proposed either, and all the methods proposed so far to solve it are heuristics. We conjecture here, as done in several prior studies (*e.g.* [21]), that the rank K binary matrix factorization problem is NP-hard.

3 General considerations

We are interested here in studying a local search heuristic that can improve the solutions provided by standard methods for the rank K binary factorization problem, for both L_1- and L_2-BMF. A general way to solve such problems is to relate them to known problems and use the solutions of the latter to solve or improve the solutions the former. For instance, as mention in Section 2, BMF has been reformulated as an ILP problem in [21] or as a clustering problem in [6]. In the following, we establish another relation between L_2-BMF and the Unconstrained Binary Quadratic Programming (UBQP) problem in order to open a new direction for this problem. This relation allows us to use the local search procedures designed for UBQP [2] to improve the solutions of L_2-BMF.

3.1 L_2-BMF and UBQP

As mentioned above, 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 $\mathbf{W}$):

$$\text{argmin}_{\mathbf{w}_{i\cdot}} \sum_{j=1}^{N}(x_{ij}^2 + (\sum_{k=1}^{K} w_{ik}h_{kj})^2 - 2x_{ij}\sum_{k=1}^{K} w_{ik}h_{kj}) \tag{3}$$

where $\mathbf{w}_{i.}$ is the i^{th} row vector of $\mathbf{W}$. The above quantity is equal to

$$\text{argmin}_{\mathbf{w}_{i.}} \sum_{j=1}^{N}(\sum_{k=1}^{K} w_{ik}h_{kj})^2 + \sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N}(-2x_{ij}h_{kj}) \tag{4}$$

since x_{ij} is not subject to any change and does not play any role in the minimization problem. The first term in (4), $\sum_{j=1}^{N}(\sum_{k=1}^{K} w_{ik}h_{kj})^2$, can be rewritten as:

$$\sum_{j=1}^{N}(\sum_{k=1}^{K} w_{ik}h_{kj})^2 = \sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N} h_{kj} + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik}w_{ik'}2 \sum_{j=1}^{N} h_{kj}h_{k'j} \tag{5}$$

Now if we use (5) to extend (4), we need to minimize the following quantity with respect to $w_{i.}$:

$$\underbrace{\sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N} h_{kj}}_{\alpha} + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik}w_{ik'}2 \sum_{j=1}^{N} h_{kj}h_{k'j} + \underbrace{\sum_{k=1}^{K} w_{ik} \sum_{j=1}^{N}(-2x_{ij}h_{kj})}_{\beta} \tag{6}$$

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

$$\sum_{k=1}^{K} w_{ik}(\sum_{j=1}^{N} h_{kj} - \sum_{j=1}^{N}(2x_{ij}h_{kj})) + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik}w_{ik'}2 \sum_{j=1}^{N} h_{kj}h_{k'j} \tag{7}$$

which is equal to

$$\sum_{k=1}^{K} w_{ik}(\sum_{j=1}^{N} h_{kj}(1 - 2x_{ij})) + \sum_{k=1}^{K} \sum_{k'=k+1}^{K} w_{ik}w_{ik'}2 \sum_{j=1}^{N} h_{kj}h_{k'j} \tag{8}$$

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

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

The quantity minimized in problem (4) can be rewritten with $\mathbf{Q}$ as follows:

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

and thus problem (4) can be reformulated as:

$$\text{argmin}_{\mathbf{w}_{i.}}\ \mathbf{w}_{i.}\mathbf{Q}\mathbf{w}_{i.}^T \tag{9}$$

which corresponds to a UBQP problem [2, 17]. One should note that the placement of the transposed vector is not important in this problem and it could be either before the weighting matrix (Q) or after that. Applying the same development to $\mathbf{H}$ (when $\mathbf{W}$ is fixed) leads to the following property.

Property 1 *Iteratively solving the L_2-BMF problem by fixing either* $\mathbf{W}$ *or* $\mathbf{H}$ *and solving for the other is equivalent to iteratively solving UBQP problems of the form:*

$$\text{argmin}_{\mathbf{v}}\ \mathbf{v}^T\mathbf{Q}\mathbf{v} \tag{10}$$

with:

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

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

Note that when $k = 1$, the optimal $\mathbf{W}$ or $\mathbf{H}$, when the other one is fixed, is directly obtained by setting the i^{th} element of $\mathbf{W}$ (or $\mathbf{H}$) to 1 if more than half of the elements with a 1 in $\mathbf{H}$ (or $\mathbf{W}$) have also a 1 in the i^{th} row (or column) of $\mathbf{X}$, 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. Using 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 Section 2).

3.2 p-opt local search for UBQP and relation to L_2-BMF

Despite the fact mentioned above, it might still be interesting to improve the NMF, SVD or Proximus solutions through local search algorithms designed for UBQP. Local search algorithms are heuristics aimed at improving a current solution by searching in its neighborhood for a better solution (hence the name "local"). 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*). [2] and [17] present such local search heuristics, called *p-opt local search*, for UBQP. The *1-opt* local search algorithm proposed in [17] looks for the solution in the neighborhood of size 1 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 [4] discussed in [2] and based on [8] and [14], 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.

Complexity Considerations

p-opt local search algorithms are of course interesting if the gains can be computed efficiently; the complexity of the *p-opt* local search algorithm pro-

posed in [17], when applied to the i^{th} row of $\mathbf{W}$, *i.e.* Problem (9), is $O(K^2)$ once the matrix $\mathbf{Q}$, which depends on i, has been constructed.

In each round of updating $\mathbf{W}$, the 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 with 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 of vectors of size N. Note that $\mathbf{Q}$ is symmetric and one only needs to compute the upper (or the lower) triangle part. Once $\mathbf{Q}$ is constructed for the first row of $\mathbf{W}$, it can then be reused for other rows with a simple update of the diagonal which could be done in $O(KN)$, as explained before. Accordingly, computing $\mathbf{Q}$ for all rows of $\mathbf{W}$ has a time complexity of $O(KMN)$ as it is one time K^2N and $(M-1)$ times KN. As updating each row is done in $O(K^2)$, for M rows we have $O(MK^2)$ for the update 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 $O(MKN)$. By dividing this quantity by the number of rows, M, one will have $O(KN)$ as the complexity of updating one row of $\mathbf{W}$ with the UBQP-based technique, denoted as *1-opt-UBQP* hereafter. One can simply see that updating a column of $\mathbf{H}$ has exactly the same complexity.

4 p-opt local search for L_1- and L_2-BMF

As discussed in Section 3, L_2-BMF can be reformulated as UBQP which enables us to apply the UBQP's *p-opt* local heuristic on L_2-BMF solutions. As mentioned in that section, the time complexity of updating one row of $\mathbf{W}$ is $O(KN)$. In this section, a more efficient *p-opt* local search procedure is introduced and its time complexity is studied.

As mentioned before, rows of $\mathbf{W}$ (or columns of $\mathbf{H}$) can be optimized independently. We show here how to optimize, in a neighborhood of size 1, a row of $\mathbf{W}$ (the reasoning is the same for a column of $\mathbf{H}$). For each row $\mathbf{w}_{i.}$ of $\mathbf{W}$, we first compute a partition of columns of $\mathbf{H}$. The main rationale 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. 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 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 1

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

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

Then:

$$W_i^0 = \{\mathbf{h}_{.3}, \mathbf{h}_{.4}\} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad W_i^{\perp} = \{\mathbf{h}_{.2}, \mathbf{h}_{.5}\} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix}$$

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

Given a row of $\mathbf{W}$ (resp. a column of $\mathbf{H}$), the idea is to exploit the characteristics of the corresponding row (resp. column) of $\mathbf{X}$ and $\mathbf{H}$ (resp. $\mathbf{W}$) in order to efficiently compute the gain that 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 1 provides simple expressions for $\Delta E(i,j)$ for improving the solutions of both L_1- and L_2-BMF; expressions that can be computed efficiently with respect to other methods, as it will be shown in Theorem 2.

Theorem 1 *Let* $\Delta E(i,j;0 \to 1, L_2)$ *(resp.* $\Delta E(i,j;1 \to 0, L_2)$*) 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 (similarly for* L_1*-norm). Then:*

$$\begin{aligned} \Delta E(i,j;0 \to 1, L_2) = & \sum_{\mathbf{h} \in W_i^0} (1 + 2 < \mathbf{w}_{i.}, \mathbf{h} >) h_j \\ & + \sum_{\mathbf{h} \in W_i^{\not\perp}} (2 < \mathbf{w}_{i.}, \mathbf{h} > -1) h_j - \sum_{\mathbf{h} \in W_i^{\perp}} h_j \end{aligned} \tag{11}$$

Algorithm 1 *1-opt-BMF*

Input: Matrices $\mathbf{X}$, $\mathbf{W}$ and $\mathbf{H}$; $p = 1$ or 2
Output: Improved $\mathbf{W}$ and $\mathbf{H}$

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

$$\Delta E(i, j; 1 \to 0, L_2) = \sum_{\mathbf{h} \in W_i^0} (1 - 2 < \mathbf{w}_{i.}, \mathbf{h} >) h_j + \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j = 1}} 1 + \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} 3 - 2 < \mathbf{w}_{i.}, \mathbf{h} > \tag{12}$$

Similarly, for the L_1-norm we have:

$$\Delta E(i, j; 0 \to 1, L_1) = \sum_{\mathbf{h} \in W_i^0} h_j + \sum_{\mathbf{h} \in W_i^{\not\perp}} h_j - \sum_{\mathbf{h} \in W_i^{\perp}} h_j \tag{13}$$

$$\Delta E(i, j; 1 \to 0, L_1) = \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j = 1}} 1 - \sum_{\mathbf{h} \in W_i^0} h_j - \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} 1 \tag{14}$$

The proof of this theorem is discussed in the appendix.

Theorem 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 1. We denote the proposed technique as **1-opt-BMF**. One should note that in Algorithm 1, we use $\Delta E(i, j; L_p)$ which is equal to $\Delta E(i, j; 0 \to 1, L_p)$ if $w_{ij} = 0$ and to $\Delta E(i, j; 1 \to 0, L_p)$ otherwise. The following example illustrates how one can utilize Theorem 1 in practice.

Example 2 *Consider the following configuration where one aims 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 matrix* $\mathbf{H}$*:*

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

Using Def. 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 0 cells and 1 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 the corresponding cells and in $w_{i.}$ and the superscripts show whether one should use the zero-to-one formulations in Theorem 1 or the one-to-zero formulations. Once the vector $\mathbb{G}$ is 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: one obtains $g_1^{0\to1} = +8$ using Eq. (11). Similarly, using Eq. (12) for $g_2^{1\to0}$ and $g_3^{1\to0}$, one obtains -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. (13) for the first cell and Eq. (14) for the second and the third one which gives us the following gain vector $\mathbb{G} = (+2,\ -1,\ -2)$. In this case, one selects the third bit to flip as it leads to the best improvement comparing to the other bits.*

The procedure expressed in Algorithm 1 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 [17], 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 $O(K^p)$, 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 1 (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 (this condition can be efficiently computed when checking the gains defined in Theorem 1).

Complexity considerations

As mentioned previously, the complexity of updating one row (say row i) of $\mathbf{W}$ through the *1-opt-UBQP* is $O(KN)$. A standard procedure, denoted as *1-opt-Standard* hereafter, which computes 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 this multiplication needs to be done one time for each cell of $w_{i.}$. Using the same reasoning, *1-opt-Standard* has a complexity of $O(K^2M)$ for updating one column of $\mathbf{H}$. We show, via Theorem 2, that the *1-opt* procedure defined on the basis of Theorem 1, *i.e. 1-opt-BMF*, 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, since we aim at finding the minimum gain that we are able to obtain using the proposed method, we consider updating a row of $\mathbf{W}$ in our calculations. The reason is that for the *1-opt-UBQP* technique the complexity is the same for both cases (updating one row of $\mathbf{W}$ and updating one column of $\mathbf{H}$); however, since the complexity in the standard method is not the same for updating one row of $\mathbf{W}$ and updating one column of $\mathbf{H}$ ($O(K^2N)$ vs. $O(K^2M)$), we consider the faster one (K^2N) in order to obtain the *minimum gain* that we can achieve with the proposed method. Consequently, Theorem 2, the proof of which is given in the appendix, considers the case of updating one row of $\mathbf{W}$.

Theorem 2 *We assume a row vector $\mathbf{w}_{i.}$ of $\mathbf{W}$ and a matrix $\mathbf{H}$. Furthermore, let d be the density of $x_{i.}$ (the i^{th} row of $\mathbf{X}$) and let τ denote the proportion of columns l of $\mathbf{H}$ 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 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.*

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 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. 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, one can note that the gain obtained by the proposed method is independent of K for the solutions of L_2-BMF while there is a dependency between K and the gain obtained by the proposed method for the solutions of L_1-BMF.

5 Experiments

5.1 General settings

We have evaluated the proposed local search heuristic, *1-opt-BMF*, on several methods used to solve the BMF problem and on several datasets. As mentioned in Section 2, in the literature, different techniques have been proposed to address the BMF problem. For instance, one can use NMF or SVD solvers and then project the results into the binary space [19]. The Proximus system, introduced in [9, 10], not only provides binary factors, but also can guarantee a binary reconstructed matrix. All these methods address the L_2-BMF problem.

To tackle the L_1-BMF, a clustering-based method, called CBMF, has been proposed in [6]. This method also provides binary factors as well as binary reconstruction/approximation of the input data. A variant of CBMF, called UBMF, is used to solve the BMF problem without binary reconstruction constraint. Based on the points mentioned above, in our experiments, we use all these methods in order to observe the performance of *1-opt-BMF* local search with other *1-opt* techniques on each of these methods.

As mentioned before, *1-opt-BMF* and *1-opt-Standard* can be applied on both L_1- and L_2-BMF methods while *1-opt-UBQP* can only be applied on L_2-BMF methods. Table 1 shows a summary of the methods used in the experiments as well as the applicable *1-opt* strategies for each of them. One should note that only CBMF and Proximus provide binary reconstruction as they impose orthogonality on one of the factors($\mathbf{W}$ or $\mathbf{H}$).

For projected NMF and SVD, we use a grid search, as done in [19] and discussed in the related work section, with a step size of 0.05 in order to project the matrices into the binary space. To perform NMF and SVD, we used Matlab built-in functions. We used our own Matlab implementations for the other methods.

Table 1: Summary of the methods used in the experiments and the applicable *1-opt* techniques for each method.

Problem	**Applicable methods**	**Applicable 1-opt**
L_1-BMF	CBMF, UBMF	1-opt-BMF 1-opt-Standard
L_2-BMF	Proj. SVD/NMF, Proximus	1-opt-BMF 1-opt-Standard 1-opt-UBQP

Table 2: 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

As mentioned before, we refer to the method proposed in this study 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 our experiments, we let the rank K vary in the set {1, 20, 40, 60, 80, 100}. All the experiments are 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.

5.2 Datasets

We examined both real world datasets as well as synthetic ones from the frequent itemset mining repository, all available at `http://fimi.ua.ac.be/data/` (last visited 21-Jul-2015). The main reason that we use frequent itemset mining datasets is that they are necessarily binary which suits our experimental setting. Table 2 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.

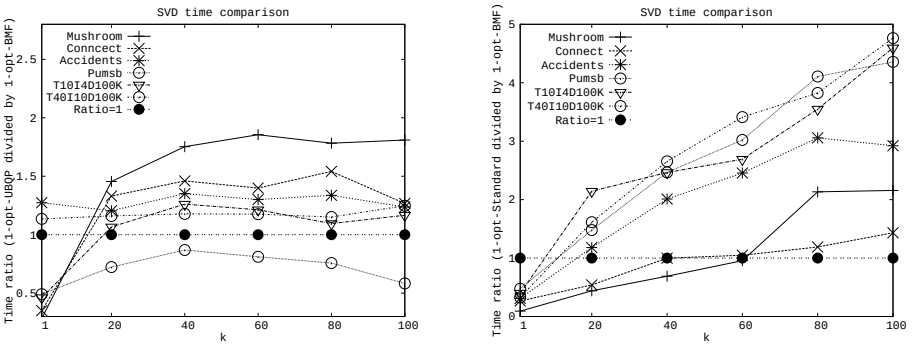

Figure 1: Efficiency of *1-opt-BMF* compared to *1-opt-UBQP* (left) and *1-opt-Standard* (right) for SVD

5.3 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 exactly 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. *1-opt-BMF* and *1-opt-Standard* can be applied to both L_1- and L_2-BMF methods, whereas *1-opt-UBQP* can only be applied on the L_2-BMF methods (NMF, SVD and Proximus). As a result, for the L_2-BMF methods, we compare *1-opt-BMF* with both *1-opt-Standard* and *1-opt-UBQP* while for the L_1-BMF 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 1-4 the **ratios** of the execution time of *1-opt-Standard* divided by execution time of *1-opt-BMF* and of the execution time of *1-opt-UBQP* divided by execution time of *1-opt-BMF*. An additional line, labeled as "Ratio=1", is added to make it simpler to compare different methods: *1-opt-BMF* is faster when the curve is above this line, and slower otherwise.

5.3.1 Projected SVD and NMF

Figure 1 illustrates the time comparison between *1-opt-UBQP* and *1-opt-BMF* on the left hand side as well as the comparison between *1-opt-Standard* and *1-opt-BMF* on the right hand side. The first issue to note is that, generally speaking, *1-opt-UBQP* is faster than *1-opt-Standard* as the ratio of the latter reaches up to 4.7 while for the former it reaches up to 1.8 in the best case. Another point on this figure is that by increasing the value of K the ratio increases significantly with respect to the standard method. In case of UBQP, the curves become almost stable for larger values of K. As discussed in Theorem 2, the gain of *1-opt-Standard* is directly influenced by

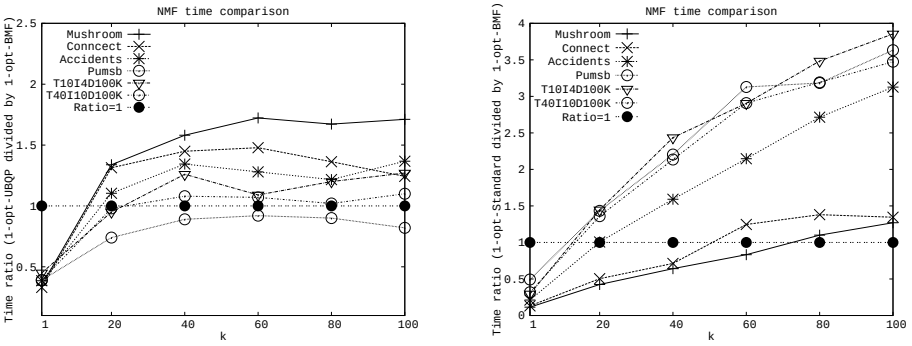

Figure 2: Efficiency of *1-opt-BMF* compared to *1-opt-UBQP* (left) and *1-opt-Standard* (right) for NMF

K, unlike that of *1-opt-UBQP*.

As one can see on the right figure, the curves are under the fixed line (Ratio=1) for `Mushroom` and `Connect` for $K < 60$. These two sets are the smallest sets in our experiments; as *1-opt-BMF* has an overhead of finding the sets defined in Def. 1, for small datasets this overhead is more *visible* with respect to the update part. However, 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 figure, 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, a small value of $(1 - \tau)$ results in a small gain, and since we have an overhead for computing the sets defined in Def. 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, 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 *1-opt-UBQP* is used. Additionally, the *semi-constant* line for ratios can also be seen in this case after a certain value of K (the figure on left) while in the *1-opt-standard* case (the figure on right) the ratio grows as the value of K increases. This is in line with the theoretical analysis provided in Theorem 2.

5.3.2 Proximus

First of all, one should note that in Proximus, one may not necessarily obtain the number of latent factors s/he had initially in mind. The reason is that,

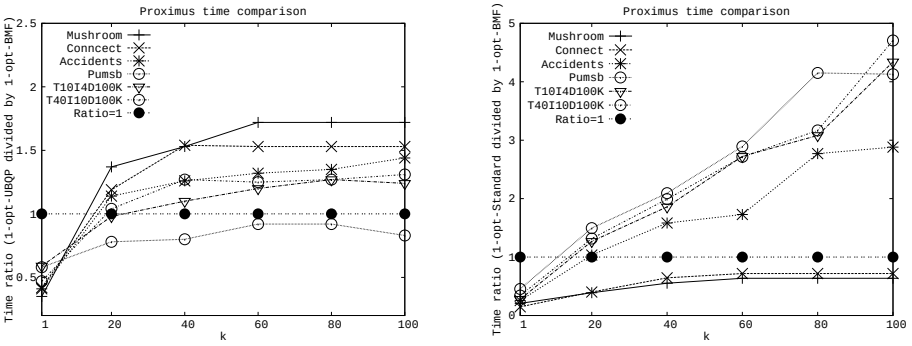

Figure 3: Efficiency of *1-opt-BMF* compared to *1-opt-UBQP* (left) and *1-opt-Standard* (right) for Proximus

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 it runs 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 3, for $K \geq 60$, we report the same results corresponding to the maximum number of factors obtained.

As one can observe in Figure 3, *1-opt-Standard* has a similar behavior to the other L_2-BMF methods (SVD and NMF). The situation with respect to *1-opt-UBQP* is also similar to that of SVD or NMF. When applied on Proximus, *1-opt-BMF* is faster on 5 collections out of 6, *i.e.* all except `Pumsb` for the same reason mentioned for the SVD and the NMF case. Here again one can see that *1-opt-BMF* gets significantly faster than the *1-opt-Standard* as the value of K increases.

5.3.3 CBMF and UBMF

For the L_1-BMF methods, one can only apply *1-opt-Standard* (Figure 4). Here, as we have seen before, except the smallest sets (`Mushroom` and `Connect`), *1-opt-BMF* is up to 8 times faster than the standard method where this efficiency in more considerable with larger values of K. Again, this is in line with the theoretical analysis provided before.

Summary

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,

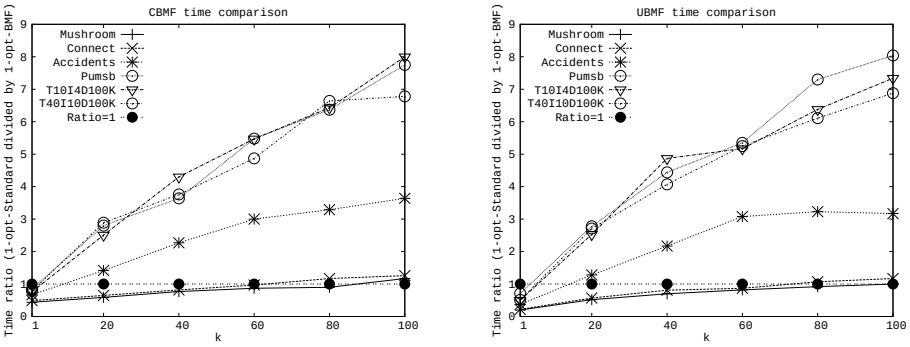

Figure 4: Efficiency of *1-opt-BMF* compared to *1-opt-Standard* for CBMF (left) and for UBMF (right)

Table 3: Summary of improvement made by *1-opt-BMF* over other methods. The numbers show the average and the best ratio obtained over all datasets and over all values of K.

Method	**1-opt-UBQP**		**1-opt-Standard**	
	Average	Best	Average	Best
Projected NMF	1.07	1.72	1.73	3.85
Projected SVD	1.15	1.85	2.06	4.76
Proximus	1.13	1.72	1.65	4.70
CBMF	NA	NA	2.90	8.00
UBMF	NA	NA	2.86	8.04

unlike *1-opt-UBQP*, *1-opt-BMF* can be applied to all the BMF methods we know of. Table 3 summarizes the average and the best improvement made by *1-opt-BMF* wrt both *1-opt-UBQP* and *1-opt-Standard*.

5.4 Impact of 1-opt local search on L_2-BMF

As discussed before, all the previously mentioned methods yield the same solution; in other words, the only difference between different *1-opt* strategies is the time complexity. Here, we would like to examine the significance of the improvement brought by any *1-opt* method. Figure 5 shows the effectiveness of (any) *1-opt* strategy when it is applied on (projected) SVD, (projected) NMF and Proximus. As one can note, projected NMF performs in 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

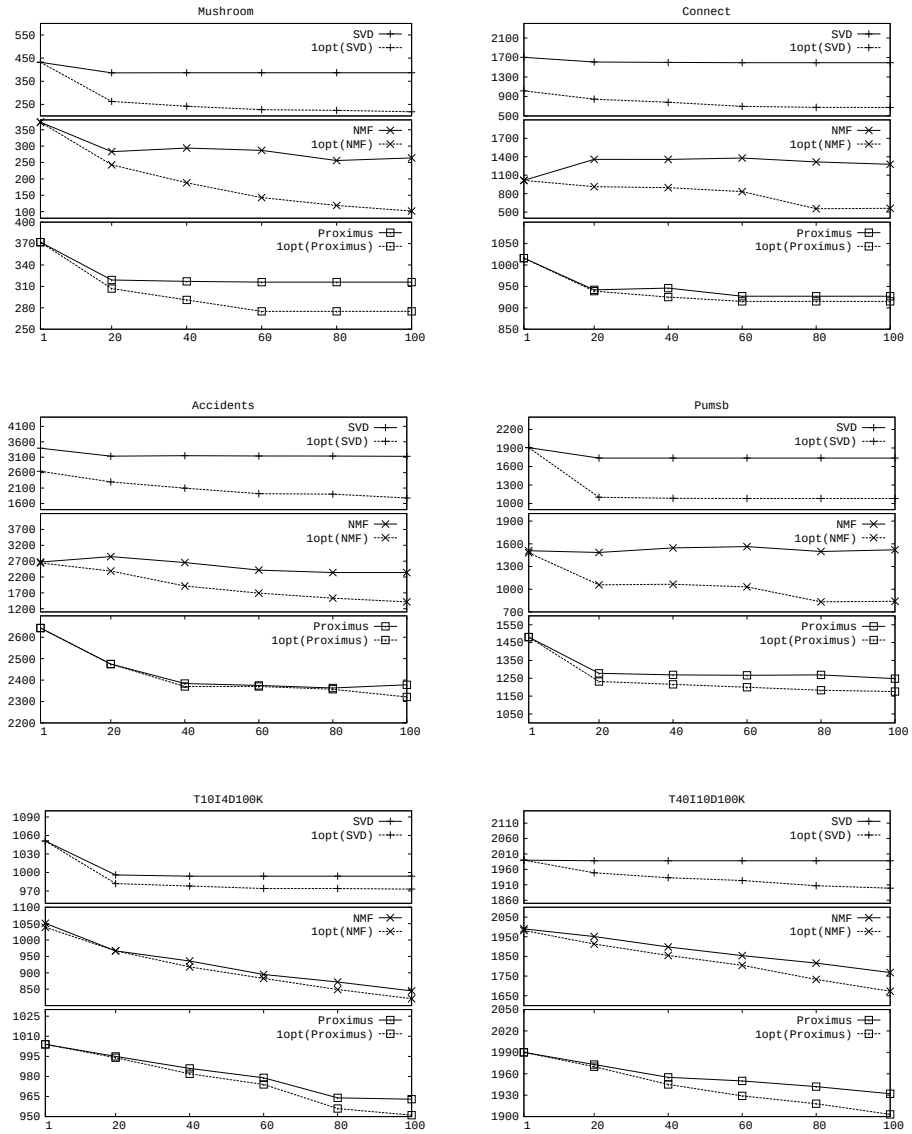

Figure 5: Improvement of L_2-norm with the *1-opt* strategy applied on projected SVD, projected NMF and Proximus. The horizontal axis represents the value of K and the vertical axis shows the L_2-norm.

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* local heuristic 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.

5.5 Impact of 1-opt local search on L_1-BMF

As mentioned before, two methods are discussed in the literature to solve the L_1-BMF problem, namely CBMF and UBMF [6], where the latter is a variant of the former. As CBMF is a K-means-based technique, one can apply swap heuristics, like the one explained in [7], in order to improve the results. We denote this technique as SCBMF ('s' stands for swapped). We also apply the *1-opt* heuristic over SCBMF and denote it as *1-opt-SCBMF*. The main difference between SCBMF and (any) *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. The swap procedures, mainly applied to K-means-like methods, replaces a center with one data point and redoes the clustering to see if the new solution is better than the previous one. Since CBMF is based the on K-means approach, one can apply the swap procedure by randomly replacing a center (a column of $\mathbf{W}$) with a random column of $\mathbf{X}$ and redo the assignments. As mentioned before, one can still apply a *1-opt* heuristic on top SCBMF (*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, *i.e. 1-opt-BMF*, (which is in general faster than the other approaches as discussed in Section 5.3) on CBMF and measure its running time; then, we let the SCBMF procedure run the same amount of time to improve the CBMF results. Like that, one can observe which method performs better under a given time budget.

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. As before, we use the Wilcoxon sum-rank test at the 1% significance level to assess whether differences in the results are significant or not.

Table 4 illustrates the improvement that *1-opt* can make over the L_1-BMF 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 the Wilcoxon rank-sum test

Table 4: Effectiveness of the *1-opt* on L_1-BMF 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

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 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 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 number of patterns spanning few lines [22], 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 [22]. 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 a *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.

6 Conclusion

We have addressed in this study 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 (L_1- and L_2-BMF) for measuring the quality of the approximation, through variants of the alternating strategy typically used for non-negative matrix factorization [12].

We have shown here that this alternating strategy for L_2-BMF can be reformulated as an Unconstrained Binary Quadratic Programming (UBQP) problem. This reformulation allows us to use the heuristics of UBQP in order to improve the solutions of L_2-BMF. Then we introduced a new local search dedicated to both L_1- and L_2-BMF, and studied its complexity with respect to other local search approaches.

Our experiments, conducted with several state-of-the-art methods, on

several datasets, have confirmed that the *1-opt* local search procedure proposed in this study is in general faster than the previously proposed ones. They have also shown that this procedure significantly improves the solutions found by state-of-the-art L_2-BMF methods: projected NMF, projected SVD and Proximus.

For the L_1-BMF methods, the experimental results show that the *1-opt* local search procedure improves the results of the state-of-the-art methods (CBMF and its "swapped" version) solving the complete L_1-BMF problem, with the binary reconstruction constraint. The situation is more contrasted with UBMF, which solves the L_1-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.

References

[1] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2), May 2009.

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

[3] N. Gillis and F. Glineur. Nonnegative factorization and the maximum edge biclique problem. In *CORE Discusssion paper*, 2008.

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

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

[6] Peng Jiang, Jiming Peng, Michael Heath, and Rui Yang. A clustering approach to constrained binary matrix factorization. In *Data Mining and Knowledge Discovery for Big Data*, pages 281–303. Springer, 2014.

[7] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18. ACM, 2002.

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

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

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

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

[12] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *In NIPS*, pages 556–562. MIT Press, 2000.

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

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

[15] Haibing Lu, Jaideep Vaidya, Vijayalakshmi Atluri, Heechang Shin, and Lili Jiang. Weighted rank-one binary matrix factorization. In *SDM*, pages 283–294, 2011.

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

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

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

[19] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Trans. Knowl. Data Eng.*, 20(10):1348–1362, 2008.

[20] Hamid Mirisaee, Eric Gaussier, and Alexandre Termier. Itemset approximation using constrained binary matrix factorization. In *Conference on Data Science and Advanced Analytics (DSAA)*, pages 39–45, 2014.

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

[22] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. 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, 2005.

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

[24] Z. Zhang, T. Li, C. Ding, X.W. Ren, and X.S. Zhang. Binary matrix factorization with applications. *ICDM*, pages 391–400, 2007.

Appendix

Theorem 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 (similarly for L_1-norm). Then:*

$$\begin{aligned}\Delta E(i,j;0\rightarrow 1,L_2) = &\sum_{\mathbf{h}\in W_i^0}(1+2<\mathbf{w}_{i.},\mathbf{h}>)h_j \\ &+\sum_{\mathbf{h}\in W_i^{\not\perp}}(2<\mathbf{w}_{i.},\mathbf{h}>-1)h_j-\sum_{\mathbf{h}\in W_i^{\perp}}h_j\end{aligned} \tag{11}$$

$$\begin{aligned}\Delta E(i,j;1\rightarrow 0,L_2) = &\sum_{\mathbf{h}\in W_i^0}(1-2<\mathbf{w}_{i.},\mathbf{h}>)h_j \\ &+\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j=1}}1+\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}}3-2<\mathbf{w}_{i.},\mathbf{h}>\end{aligned} \tag{12}$$

Similarly, for the L_1-norm we have:

$$\Delta E(i,j;0\rightarrow 1,L_1)=\sum_{\mathbf{h}\in W_i^0}h_j+\sum_{\mathbf{h}\in W_i^{\not\perp}}h_j-\sum_{\mathbf{h}\in W_i^{\perp}}h_j \tag{13}$$

$$\Delta E(i,j;1\rightarrow 0,L_1)=\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j=1}}1-\sum_{\mathbf{h}\in W_i^0}h_j-\sum_{\substack{\mathbf{h}\in W_i^{\not\perp}\\ <\mathbf{w}_{i.},\mathbf{h}>h_j>1}}1 \tag{14}$$

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

$$E_0^2 - E_0^1 = \sum_{\mathbf{h} \in W_i^0} (< \mathbf{w}_{i.}, \mathbf{h} > + h_j)^2 - \sum_{\mathbf{h} \in W_i^0} < \mathbf{w}_{i.}, \mathbf{h} >^2 \tag{15}$$

$$= \sum_{\mathbf{h} \in W_i^0} \big((< \mathbf{w}_{i.}, \mathbf{h} > + h_j)^2 - < \mathbf{w}_{i.}, \mathbf{h} >^2 \big) \tag{16}$$

$$= \sum_{\mathbf{h} \in W_i^0} < \mathbf{w}_{i.}, \mathbf{h} >^2 + 2 < \mathbf{w}_{i.}, \mathbf{h} > h_j + h_j^2 - < \mathbf{w}_{i.}, \mathbf{h} >^2 \tag{17}$$

$$= \sum_{\mathbf{h} \in W_i^0} 2 < \mathbf{w}_{i.}, \mathbf{h} > h_j + h_j^2 \tag{18}$$

$$= \sum_{\mathbf{h} \in W_i^0} (1 + 2 < \mathbf{w}_{i.}, \mathbf{h} >) h_j \tag{19}$$

Note that in (18) 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. 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:

$$E_{\not\perp}^2 - E_{\not\perp}^1 = \sum_{\mathbf{h} \in W_i^{\not\perp}} (< \mathbf{w}_{i.}, \mathbf{h} > - 1 + h_j)^2 - \sum_{\mathbf{h} \in W_i^{\not\perp}} (< \mathbf{w}_{i.}, \mathbf{h} > - 1)^2 \tag{20}$$

$$= \sum_{\mathbf{h} \in W_i^{\not\perp}} \big((< \mathbf{w}_{i.}, \mathbf{h} > - 1 + h_j)^2 - (< \mathbf{w}_{i.}, \mathbf{h} > - 1)^2 \big) \tag{21}$$

$$= \sum_{\mathbf{h} \in W_i^{\not\perp}} (2 < \mathbf{w}_{i.}, \mathbf{h} > - 1) h_j \tag{22}$$

- 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

$$E_{\perp}^2 - E_{\perp}^1 = -\sum_{\mathbf{h} \in W_i^{\not\perp}} h_j \tag{23}$$

Summing (19), (22) and (23) yields Eq. (11).

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

- 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:

$$E_0^2 - E_0^1 = \sum_{\mathbf{h} \in W_i^0} (< \mathbf{w}_{i.}, \mathbf{h} > - h_j)^2 - \sum_{\mathbf{h} \in W_i^0} < \mathbf{w}_{i.}, \mathbf{h} >^2 \tag{24}$$

$$= \sum_{\mathbf{h} \in W_i^0} \left((< \mathbf{w}_{i.}, \mathbf{h} > - h_j)^2 - < \mathbf{w}_{i.}, \mathbf{h} >^2 \right) \tag{25}$$

$$= \sum_{\mathbf{h} \in W_i^0} < \mathbf{w}_{i.}, \mathbf{h} >^2 - 2 < \mathbf{w}_{i.}, \mathbf{h} > h_j + h_j^2 - < \mathbf{w}_{i.}, \mathbf{h} >^2 \tag{26}$$

$$= \sum_{\mathbf{h} \in W_i^0} -2 < \mathbf{w}_{i.}, \mathbf{h} > h_j + h_j^2 \tag{27}$$

$$= \sum_{\mathbf{h} \in W_i^0} (1 - 2 < \mathbf{w}_{i.}, \mathbf{h} >) h_j \tag{28}$$

Here, we can again replace h_j^2 with h_j in (27) 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):

$$\sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j = 1}} 1 \tag{29}$$

  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

$$E_{\not\perp}^2 - E_{\not\perp}^1 = \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} (< \mathbf{w}_{i.}, \mathbf{h} > -2)^2 - \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} (< \mathbf{w}_{i.}, \mathbf{h} > -1)^2 \tag{30}$$

$$= \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} (< \mathbf{w}_{i.}, \mathbf{h} > -2)^2 - (< \mathbf{w}_{i.}, \mathbf{h} > -1)^2 \tag{31}$$

$$= \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} 3 - 2 < \mathbf{w}_{i.}, \mathbf{h} > \tag{32}$$

  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 (28), (29) and (32) yields Eq. (12).

$\boldsymbol{\Delta E(i,j;0 \rightarrow 1, L_1)}$

- 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:

$$\sum_{\mathbf{h} \in W_i^0} h_j \tag{33}$$

- 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:

$$E_{\not\perp}^2 - E_{\not\perp}^1 = \sum_{\mathbf{h} \in W_i^{\not\perp}} (< \mathbf{w}_{i.}, \mathbf{h} > -1 + h_j) - \sum_{\mathbf{h} \in W_i^{\not\perp}} (< \mathbf{w}_{i.}, \mathbf{h} > -1) \tag{34}$$

$$= \sum_{\mathbf{h} \in W_i^{\not\perp}} < \mathbf{w}_{i.}, \mathbf{h} > -1 + h_j - < \mathbf{w}_{i.}, \mathbf{h} > +1 \tag{35}$$

$$= \sum_{\mathbf{h} \in W_i^{\not\perp}} h_j \tag{36}$$

- 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:

$$- \sum_{\mathbf{h} \in W_i^{\perp}} h_j \tag{37}$$

Summing (33), (36) and (37) yields Eq. (13).

$\boldsymbol{\Delta E(i,j;1 \rightarrow 0, L_1)}$

- 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:

$$- \sum_{\mathbf{h} \in W_i^0} h_j \tag{38}$$

- 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

$$\sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j = 1}} 1 \tag{39}$$

  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

$$- \sum_{\substack{\mathbf{h} \in W_i^{\not\perp} \\ <\mathbf{w}_{i.}, \mathbf{h}> h_j > 1}} 1 \tag{40}$$

- 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 (38), (39) and (40) yields Eq. (14).□

Theorem 2 *We assume a row vector* $\mathbf{w}_{i.}$ *of* $\mathbf{W}$ *and a matrix* $\mathbf{H}$*. Furthermore, let* d *be the density of* $x_{i.}$ *(the* i^{th} *row of* $\mathbf{X}$*) and let* τ *denote the proportion of columns* l *of* $\mathbf{H}$ *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 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:

Table 5: Complexity analysis of Eq. (11)

Sub-equation	**Operations needed**	**Complexity**
$\sum_{\mathbf{h}\in W_i^0}(1+2<\mathbf{w}_{i.},\mathbf{h}>)h_j$	dot prod. 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 prod. 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 $W_i^{\perp}$	$O(\tau N)$

Table 6: Complexity analysis of Eq. (12)

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

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 (11), we have three parts in our complexity analysis which have been shown in Table 5. According to this table, if we sum all the complexities, we will have the following total complexity for Equation (11):

$$O\big(KN(1-d)+KN(d-\tau)+\tau N\big)=O\big(KN(1-\tau)\big) \tag{41}$$

For Equation (12), 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 6 illustrates these entities: according to Table 6, the total complexity of Equation (12) amount to

$$O\big(KN(1-d)+KN(d-\tau)\big)=O\big(KN(1-\tau)\big) \tag{42}$$

And thus, according to (41) and (42) the proposed method in case of L_2 norm has a complexity of $O\big(KN(1-\tau)\big)$. Now if we divide the complexity of the UBQP-based approach, *i.e.* $O(KN)$, by this quantity, we have a gain of $(1-\tau)^{-1}$, and if we divide that by the complexity of the standard method, *i.e.* $O(K^2N)$, we have a gain of $K(1-\tau)^{-1}$.

For the L_1 case, we have two equations: (13) and (14). We can consider Table 7 for Equation (13). Based on that, the total complexity of this

Table 7: Complexity analysis of Eq. (13)

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 $\mathbf{W}_i^{\perp}$	$O(\tau N)$

Table 8: Complexity analysis of Eq. (14)

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 prod. between $w_{i.}$ and $\mathbf{W}_i^{\not\perp}$	$O\big(KN(d-\tau)\big)$

equation is:

$$O\big(N(1-d) + N(d-\tau) + \tau N\big) = O(N) \tag{43}$$

For Equation (14), we have basically two parts to study as shown in Table 8. Comparing (43) and complexities in Table 8, 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 complexity of the standard method, $O(K^2N)$, by this quantity, the minimum gain er obtain will be $min\{K^2, K(d-\tau)^{-1}\}$ which establishes the theorem.□