Undercover Boolean Matrix Factorization with MaxSAT

Florent Avellaneda,[1,2] **Roger Villemaire**[1]

[1] Université du Québec à Montréal (UQAM), Montréal, Canada
[2] Centre de Recherche de l'Institut Universitaire de Gériatrie de Montréal, Montréal, Canada
avellaneda.florent@uqam.ca, villemaire.roger@uqam.ca

Abstract

The k-undercover Boolean matrix factorization problem aims to approximate a $m \times n$ Boolean matrix $\boldsymbol{X}$ as the Boolean product of an $m \times k$ and a $k \times n$ matrices $\boldsymbol{A} \circ \boldsymbol{B}$ such that $\boldsymbol{X}$ is a cover of $\boldsymbol{A} \circ \boldsymbol{B}$, i.e., no representation error is allowed on the 0's entries of the matrix $\boldsymbol{X}$. To infer an optimal and "block-optimal" k-undercover, we propose two exact methods based on MaxSAT encodings. From a theoretical standpoint, we prove that our method of inferring "block-optimal" k-undercover is a $(1 - \frac{1}{e}) \approx 0.632$ approximation for the optimal k-undercover problem. From a practical standpoint, experimental results indicate that our "block-optimal" k-undercover algorithm outperforms the state-of-the-art even when compared with algorithms for the more general k-undercover Boolean Matrix Factorization problem for which only minimizing reconstruction error is required.

1 Introduction

The Boolean Matrix Factorization (BMF) problem consists, for an $m \times n$ Boolean matrix $\boldsymbol{X}$ and an integer $k \leq min(n, m)$, of finding $m \times k$ and $k \times n$ Boolean matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ such that $\boldsymbol{A} \circ \boldsymbol{B}$ is equal to $\boldsymbol{X}$ for as many entries as possible. The Boolean semiring operator $\circ$ is matrix multiplication where the product is logical "AND" and addition logical "OR".

When the rows of the matrix $\boldsymbol{X}$ represent instances and the columns attributes, such a decomposition yields a matrix $\boldsymbol{A}$ that represents the same set of instances using only k derived attributes and a matrix $\boldsymbol{B}$ that defines these new attributes in terms of the original ones. Thus, since BMF summarizes data in terms of these new k attributes, this factorization has been applied to many areas such as role-based access control (Vaidya, Atluri, and Guo 2007), multi-label classification (Wicker, Pfahringer, and Kramer 2012), network pattern mining (Kocayusufoglu, Hoang, and Singh 2018) and functional interactions in brain networks (Haddad et al. 2018).

Although many methods have been developed for the non-Boolean case, these methods fail to work in a Boolean context. For example, it is possible that the well-known Singular Value Decomposition (SVD) obtains a greater reconstruction error than a BMF for the same decomposition size (Miettinen et al. 2008). For this reason, specific algorithms dealing with the Boolean context must be developed.

In recent years, many different approaches have been applied to BMF (Miettinen and Neumann 2020). Representing BMF as a maximum a posteriori inference problem, (Ravanbakhsh, Póczos, and Greiner 2016) presented a message passing algorithm "Message Passing", while (Kovacs, Günlük, and Hauser 2021) introduced the "CG" algorithm that used mixed integer linear programming. This last paper also presented a greedy algorithm called "k-greedy". Still widely used is also the greedy algorithm "ASSO" from (Miettinen et al. 2008). As for the "MEBF" algorithm of (Wan et al. 2020), it used matrix permutations to approximate an upper triangular-like matrix.

The quality of a BMF solution is usually evaluated in terms of the *reconstruction error*, which is the proportion of entries where $\boldsymbol{A} \circ \boldsymbol{B}$ differs from $\boldsymbol{X}$. This considers erroneous 0's and 1's to be of equal importance. However, in applications such as access control where an erroneous 1 in $\boldsymbol{A} \circ \boldsymbol{B}$ would represent a spurious permission, incorrect 1's in $\boldsymbol{A} \circ \boldsymbol{B}$ must be avoided. In *undercover BMF* (also known as *from-below approximation BMF*) one hence searches for a solution $\boldsymbol{A} \circ \boldsymbol{B}$ whose 1-entries have this same value in $\boldsymbol{X}$.

Unexpectedly, undercover BMF algorithms have been shown to give good approximations compared to general BMF algorithms (Belohlavek and Trnecka 2015) even though BMF allows more candidate factorizations. As for BMF, different approaches have been applied to undercover BMF. For instance, Formal Concept Analysis (FCA) has been used by (Belohlavek and Trnecka 2015) to introduce the "GreEss" algorithm. This approach was later improved upon with the "IterEss" algorithm (Belohlavek, Outrata, and Trnecka 2019). Also, the "Tiling" algorithm (Geerts, Goethals, and Mielikäinen 2004) greedily searches for rectangular patterns (tiles). Building on the approach used with "GreConD", the "GRECOND+" algorithm (Belohlavek and Trnecka 2018) adds a greedy step to minimize reconstruction error. However, this last step will usually not preserve the undercovering property. Similarly, "PANDA+" (Lucchese, Orlando, and Perego 2013) is not strictly an undercover BMF algorithm, but it first finds dense initial components that overcover only a strictly limited amount of entries, before greedily extending without increasing recovery error.

The contribution of this paper is a MaxSAT approach to

solve the undercover BMF problem. MaxSAT is the optimization version of the well-known Boolean Satisfiability Testing (SAT) problem. We leverage recent progress in MaxSAT solving that occurred in the wake of the impressive developments in SAT solving (Bacchus, Järvisalo, and Martins 2021). We introduce the notion of *block-optimal factorization* with efficient algorithms for computing this factorization and show that even if a block-optimal factorization is a restricted form of undercover BMF, our algorithms outperform state-of-the-art undercover and even BMF algorithms.

This paper is structured as follows. First, to formalize the approach, we provide definitions related to Boolean matrix factorization (Section 2). Then, we give a MaxSAT encoding for the k-undercover Boolean matrix factorization, propose two optimizations and evaluate the performance on synthetic data (Section 3). Finally, we introduce block-optimal factorizations, propose exact methods for finding such a factorization, and compare the results obtained with the classical approach in the literature (Section 4).

2 Definitions

We denote by $\boldsymbol{A}_{m\times n}$ a *Boolean matrix* $\boldsymbol{A} \in \{0,1,null\}^{n\times m}$ with m rows, and n columns. We use $null$ to represent missing data and if $\boldsymbol{A} \in \{0,1\}^{n\times m}$ we say that the matrix $\boldsymbol{A}$ is complete.

We also use the notation $\boldsymbol{A}_{i,j}$ to represent the entry in the i-th row and the j-th column and the notation $\boldsymbol{A}_{i,:}$ and $\boldsymbol{A}_{:,j}$, to represent the i-th row and the j-th column of $\boldsymbol{A}$, respectively. A *submatrix* $\boldsymbol{A}_{I,J}$ of $\boldsymbol{A}$ is the matrix obtained by selecting a subset $I \subseteq [1,m]$ of $\boldsymbol{A}$'s rows and a subset $J \subseteq [1,n]$ of $\boldsymbol{A}$'s columns.

We now formally define the Boolean product of two matrices.

Definition 1. *The* Boolean product *of two complete matrices $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ is a matrix $(\boldsymbol{A}\circ\boldsymbol{B})_{m\times n}$ define by:*

$$(\boldsymbol{A}\circ\boldsymbol{B})_{i,j} = \bigvee_{\ell=1}^{k} (\boldsymbol{A}_{i,\ell} \wedge \boldsymbol{B}_{\ell,j})$$

This definition is similar to the classical matrix product, where the product is replaced by the logical "AND" and addition by the logical "OR".

Note that $(\boldsymbol{A}\circ\boldsymbol{B})$ can also be written as the union of products of rank 1 matrices:

$$(\boldsymbol{A}\circ\boldsymbol{B}) = \bigcup_{\ell=1}^{k} (\boldsymbol{A}_{:,\ell}\circ\boldsymbol{B}_{\ell,:})$$

where $(\boldsymbol{A}\cup\boldsymbol{B})_{i,j} = \boldsymbol{A}_{i,j} \vee \boldsymbol{B}_{i,j}$. The matrix $\boldsymbol{A}_{:,\ell}\circ\boldsymbol{B}_{\ell,:}$ is also called the (ℓ-th) *block* of the product. The Boolean product $\boldsymbol{A}\circ\boldsymbol{B}$ is then the union of its blocks.

Definition 2. *A matrix $\boldsymbol{D}_{m\times n}$* undercovers *a matrix $\boldsymbol{X}_{m\times n}$ (denoted $\boldsymbol{D}\leq\boldsymbol{X}$) if $\boldsymbol{D}$ is a complete matrix such that there are no i,j such that $\boldsymbol{X}_{i,j}=0$ and $\boldsymbol{D}_{i,j}=1$.*

An entry $\boldsymbol{X}_{i,j}$ such that $\boldsymbol{D}_{i,j}=1$ is said to be "covered" by $\boldsymbol{D}$. We define $(\boldsymbol{A}-\boldsymbol{B})$, the subtraction operator between two matrices by:

$$(\boldsymbol{A}-\boldsymbol{B})_{i,j} = \begin{cases} null & \text{if } \boldsymbol{B}_{i,j}=1 \\ \boldsymbol{A}_{i,j} & \text{otherwise} \end{cases}$$

We denote by $|\boldsymbol{A}|_1$ the number of 1's in $\boldsymbol{A}$.

Note that when $\boldsymbol{B}\leq\boldsymbol{A}$ the i,j-entry of $(\boldsymbol{A}-\boldsymbol{B})$ is 1 exactly when $\boldsymbol{A}_{i,j}=1$ and $\boldsymbol{B}_{i,j}=0$ and $|\boldsymbol{A}-\boldsymbol{B}|_1$ is equal to the number of erroneous entries in the reconstruction $\boldsymbol{B}$ of $\boldsymbol{A}$.

Definition 3. *Two matrices $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ are an optimal k-undercover of the matrix $\boldsymbol{X}_{m\times n}$ if:*

- $(\boldsymbol{A}\circ\boldsymbol{B}) \leq \boldsymbol{X}$
- $\forall \boldsymbol{A'}_{m\times k}\, \forall \boldsymbol{B'}_{k\times n}$: $(\boldsymbol{A'}\circ\boldsymbol{B'}) \leq \boldsymbol{X} \Rightarrow |\boldsymbol{X}-(\boldsymbol{A'}\circ\boldsymbol{B'})|_1 \geq |\boldsymbol{X}-(\boldsymbol{A}\circ\boldsymbol{B})|_1$

3 Optimal Undercover Factorization

In this section, we propose a MaxSAT encoding for optimal k-undercover. We start by giving a simple encoding of the problem in MaxSAT, then we propose two optimizations. The first one is a classical optimization which consists in adding constraints to break the symmetry of the solutions. The second optimization, which we call cardinality generation, is to the best of our knowledge, a type of optimization not found in the literature. Here, we will show that cardinality generation allows us to gain several orders of magnitude on the computation time needed to solve the MaxSAT formula.

3.1 MaxSAT Encoding

As with SAT instances a MaxSAT instance is specified by a set of constraints (clauses) that are logical "OR" of Boolean variables and their negations. MaxSAT also considers some clauses to be *soft* and one searches for an assignment of Boolean values to variables that satisfy all non-soft clauses while maximizing the number of satisfied soft clauses.

Given a matrix $\boldsymbol{X}_{m\times n}$, the k-undercover problem can be encoded in MaxSAT as the set of following clauses.

For every i,j such that $\boldsymbol{X}_{i,j}=0$:

$$\bigwedge_{\ell=1}^{k} (\neg\boldsymbol{A}_{i,\ell} \vee \neg\boldsymbol{B}_{\ell,j}) \tag{1}$$

These clauses guarantee that $(\boldsymbol{A}\circ\boldsymbol{B})_{i,j}=0$.

For every i,j such that $\boldsymbol{X}_{i,j}=1$:

$$\bigwedge_{\ell=1}^{k} (\boldsymbol{T}^{\ell}_{i,j} \Rightarrow \boldsymbol{A}_{i,\ell}) \wedge \bigwedge_{\ell=1}^{k} (\boldsymbol{T}^{\ell}_{i,j} \Rightarrow \boldsymbol{B}_{\ell,j}) \tag{2}$$

These clauses guarantee that if $\boldsymbol{T}^{\ell}_{i,j}=1$ then $(\boldsymbol{A}\circ\boldsymbol{B})_{i,j}=1$.

Finally, for every i,j such that $\boldsymbol{X}_{i,j}=1$:

$$\neg\boldsymbol{S}_{i,j} \vee \bigvee_{\ell=1}^{k} \boldsymbol{T}^{\ell}_{i,j} \tag{3}$$

where $\boldsymbol{S}_{i,j}$ are unary soft clauses. These clauses guarantee that if $\boldsymbol{S}_{i,j}=1$ then $(\boldsymbol{A}\circ\boldsymbol{B})_{i,j}=1$.

Note that since clauses (1) guarantee to infer an undercover, and since (2) and (3) guarantee to cover $\boldsymbol{X}_{i,j}$ if $\boldsymbol{S}_{i,j}=1$, then an assignment that maximizes the number of $\boldsymbol{S}_{i,j}=1$ corresponds to a maximal undercover.

3.2 Symmetry Breaking

We propose in this section an optimization based on symmetry breaking. Symmetry breaking, a well-known practice of the SAT community (Aloul et al. 2002; Aloul, Sakallah, and Markov 2006; Brown, Finkelstein, and Purdom Jr 1988), consists in adding constraints to quickly remove some assignments as acceptable solutions. The idea is to reduce the number of solutions that are equivalent to each other.

In our matrix factorization context, for any solution $\boldsymbol{A} \circ \boldsymbol{B}$ and for all $i, j \leq k$, we can switch the i-th column and the j-th column on the matrix $\boldsymbol{A}$ and switch the i-th row and the j-th row on the matrix $\boldsymbol{B}$ to obtain a new solution whose Boolean product remains unchanged.

To avoid this kind of permutation, we impose a lexicographical order on the rows of the matrix $\boldsymbol{B}$. This can be performed with the following set clauses.

For every $i \in [1, k-1]$ and $j \in [1, n-1]$:

$$(\boldsymbol{Z}_{i,j} \wedge \boldsymbol{B}_{i,j} \wedge \boldsymbol{B}_{i+1,j}) \Rightarrow \boldsymbol{Z}_{i,j+1} \tag{4}$$

For every $i \in [1, k-1]$ and $j \in [1, n-1]$:

$$(\boldsymbol{Z}_{i,j} \wedge \neg\boldsymbol{B}_{i,j} \wedge \neg\boldsymbol{B}_{i+1,j}) \Rightarrow \boldsymbol{Z}_{i,j+1} \tag{5}$$

For every $i \in [1, k-1]$ and $j \in [1, n]$:

$$(\boldsymbol{Z}_{i,j} \wedge \neg\boldsymbol{B}_{i,j}) \Rightarrow \neg\boldsymbol{B}_{i+1,j} \tag{6}$$

And finally, for each $i \in [1, k]$:

$$\boldsymbol{Z}_{i,1} \tag{7}$$

3.3 Generate Cardinalities

A recent and efficient algorithm for solving the MaxSAT problem is the OLL algorithm (Andres et al. 2012). This algorithm originally created for ASP solvers has been adapted to MaxSAT solvers (Morgado, Dodaro, and Marques-Silva 2014) and is used in tools such as MSCG (Morgado, Ignatiev, and Marques-Silva 2014), RC2 (Ignatiev, Morgado, and Marques-Silva 2019) and EvalMaxSAT (Avellaneda 2020).

The principle of this algorithm is to search for unsatisfiable cores (sets of soft clauses that cannot all be satisfied at the same time) in order to replace many soft clauses by few cardinality constraints. The following example illustrates this algorithm.

Example 1. *Let $\varphi_{soft} = \{x_1, x_2, x_3\}$ be a set of soft (unary) clauses and $\varphi_{hard} = \{(\neg x_1 \vee \neg x_2), (\neg x_2 \vee \neg x_3), (\neg x_1 \vee \neg x_3)\}$ be a set of hard clauses.*

A call to a SAT solver on the set of clauses $\varphi_{soft} \cup \varphi_{hard}$ will find that the formula is unsatisfiable and an unsatisfiable core can be $\{x_1, x_2\}$. This means that at least one of the two clauses x_1 or x_2 has to be false and the cost of the MaxSAT formula is at least 1. Thus, we remove x_1 and x_2 from φ_{soft}, we add the cardinality constraint $(x_1 + x_2 \geq 1)$ to φ_{soft} and we increment the cost of the formula.

On the second iteration, a SAT solver will find that $\varphi_{soft} \cup \varphi_{hard}$ is still unsatisfiable and the core will be $\{x_3, (x_1 + x_2 \geq 1)\}$. This means that at least x_3 has to be false or $x_1 + x_2 \geq 2$. Thus, we remove x_3 from φ_{soft}, replace $x_1 + x_2 \geq 1$ by $x_1 + x_2 \geq 0$, add a new cardinality constraint $(x_3 + (x_1 + x_2 \geq 1) \geq 1)$ and increment the cost of the formula.

Now, the formula $\varphi_{soft} \cup \varphi_{hard}$ is satisfiable and we can conclude that the cost of the initial formula is two.

We observe that a call to a SAT solver is necessary to increment the cost of the formula. Thus, if the cost of the formula to be solved is high, it implies many calls to a SAT solver, which, in practice, can take a long time.

In this section, we propose a method to significantly reduce the number of SAT calls performed. The idea is to use the knowledge of the problem modeled by the formula to be solved in order to quickly find sets of soft clauses in which only k of the clauses can be satisfied at a time. If at most k clauses can be satisfied in the set of clauses $\{c_1, c_2, \ldots, c_f\}$, this means that we can increase the cost of the formula by $f - k$, remove the f clauses from the set φ_{soft} and add a cardinality constraint $c_1 + c_2 + \cdots + c_n \geq k$ to φ_{soft}.

A similar idea has been used by the RC2 solver (Ignatiev, Morgado, and Marques-Silva 2019), but only for $k = 1$. The approach used consists in calling a SAT solver to find incompatibilities between soft clauses. When a set of clauses that are incompatible with each other is found, it generates a cardinality constraint "AtMost1" and removes these clauses from φ_{soft}. Although this approach is efficient, it has a major drawback: calling a SAT solver to find incompatibilities between clauses can be very costly. In their approach, the authors limit the search time of the SAT solver by accepting not to detect some incompatibilities to overcome this drawback.

Knowing the problem modeled by the formulas will allow us to propose a more efficient and general method than the one used in RC2. In order to find sets of soft clauses for which only k clauses can be satisfied at the same time, we simply identify sets of 1's entries in the matrix to undercover that are two by two *incompatible*.

Definition 4. *Two entries $\boldsymbol{X}_{i_1,j_1}$ and $\boldsymbol{X}_{i_2,j_2}$ of a matrix $\boldsymbol{X}_{m\times n}$ are* incompatible *if $\boldsymbol{X}_{i_1,j_1} = 1$, $\boldsymbol{X}_{i_2,j_2} = 1$ and $\boldsymbol{X}$ contains either $\boldsymbol{X}_{i_1,j_2} = 0$ or $\boldsymbol{X}_{i_2,j_1} = 0$.*

Now, the key property is that a rank-1 factorization can cover at most one of two incompatible 1-entries.

Proposition 1. *If $\boldsymbol{X}_{i_1,j_1}$ and $\boldsymbol{X}_{i_2,j_2}$ are two incompatible entries of a matrix $\boldsymbol{X}_{m\times n}$ then for every product of rank-1 matrices $(\boldsymbol{A}_{m\times 1} \circ \boldsymbol{B}_{1\times n}) \leq \boldsymbol{X}$, we have $(\boldsymbol{A} \circ \boldsymbol{B})_{i_1,j_1} = 0$ or $(\boldsymbol{A} \circ \boldsymbol{B})_{i_2,j_2} = 0$.*

Proof. If $(\boldsymbol{A} \circ \boldsymbol{B})_{i_1,j_1} = 1$ and $(\boldsymbol{A} \circ \boldsymbol{B})_{i_2,j_2} = 1$, then $\boldsymbol{A}_{i_1,1} = 1$, $\boldsymbol{A}_{i_2,1} = 1$, $\boldsymbol{B}_{1,j_1} = 1$ and $\boldsymbol{B}_{1,j_2} = 1$. This implies that $(\boldsymbol{A} \circ \boldsymbol{B})_{i_1,j_2} = 1$ and $(\boldsymbol{A} \circ \boldsymbol{B})_{i_2,j_1} = 1$ and therefore that $(i_1, j_1, 1)$ are not incompatible with $(i_2, j_2, 1)$. □

Furthermore, this implies a bound on how many entries in a set of two by two incompatible 1 entries can be covered in a rank-k factorization.

Theorem 1. *Let $\boldsymbol{X}_{m\times n}$ be a matrix. If $I \subseteq \boldsymbol{X}$ is a set of 1's entries in $\boldsymbol{X}$ two by two incompatible, then at most k of these entries can be covered by a k-undercover.*

Proof. By definition, a k-undercover correspond to two matrices $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ such that $\boldsymbol{A}\circ\boldsymbol{B}\leq\boldsymbol{X}$. Also by definition, $\boldsymbol{A}\circ\boldsymbol{B}=\bigcup_{\ell=1}^{k}\boldsymbol{D}_\ell$ where $\boldsymbol{D}_\ell=\boldsymbol{A}_{:,\ell}\circ\boldsymbol{B}_{\ell,:}$. Since $(\boldsymbol{A}\circ\boldsymbol{B})\leq\boldsymbol{X}$ and $\boldsymbol{D}_\ell\leq(\boldsymbol{A}\circ\boldsymbol{B})$ we know that $\boldsymbol{D}_\ell\leq\boldsymbol{X}$ for each $\ell\in[1,k]$. By Proposition 1, since all elements from I are two by two incompatible, each $\boldsymbol{D}_\ell$ can cover at most one element of I. Thus, at most k elements of I can be covered by a k-undercover. □

Theorem 1 implies that for each set I of elements two by two incompatible in $\boldsymbol{X}_{m\times n}$, at most k elements can be covered by an undercover of rank k. Thus, without loss of generality, we can remove the soft clauses $\{\boldsymbol{S}_{i,j}\mid(i,j)\in I\}$ associated with elements from I and replace them by a cardinality constraint $\sum_{(i,j)\in I}\boldsymbol{S}_{i,j}\geq k$. In Algorithm 1, we propose a method that performs these substitutions.

Algorithm 1: GenerateCardinalities

Input: A matrix $\boldsymbol{X}_{m\times n}$, an integer k, a Boolean formula Φ, and a matrix of soft unary clauses $\boldsymbol{S}$.
Output: A simplified and equivalent Boolean formula.

$AllOne\leftarrow\{(i,j)\mid\boldsymbol{X}_{i,j}=1\}$
while $true$ **do**
  $noComp\leftarrow\{\}$
  for all $(i,j)\in AllOne$ **do**
    if $\forall(i',j')\in noComp:\boldsymbol{X}_{i,j'}=0\vee\boldsymbol{X}_{i',j}=0$ **then**
      $noComp\leftarrow noComp\cup\{(i,j)\}$
    end if
  end for
  if $noComp=\emptyset$ **then**
    return $\Phi,\boldsymbol{S}$
  end if
  $AllOne\leftarrow AllOne\setminus noComp$
  if $|noComp|>k$ **then**
    $\Phi\leftarrow(\sum_{(i,j)\in noComp}\boldsymbol{S}_{i,j}\geq k))$
    $\Phi\leftarrow$ remove $\boldsymbol{S}_{i,j}$ as a soft clause in Φ
  end if
end while
return Φ

3.4 Experimentation

All experiments have been performed on Ubuntu 20.04 with Intel® Core™i7-2600K CPU @ 3.40GHz and 12 GB of RAM.

In this section, we compare the time required to solve randomly generated problems with all MaxSAT solvers competing in the MaxSAT Evaluation 2021 (Bacchus et al. 2021), i.e., MaxHS (Bacchus 2021), CASHWMaxSAT (Lei et al. 2021), EvalMaxSAT (Avellaneda 2020), UWrMaxSAT (Piotrów 2021; Piotrow 2020), Open-WBO-RES (Martins et al. 2021; Martins, Manquinho, and Lynce 2014), Pacose (Paxian and Becker 2021) and Exact (Devriendt 2021). The method followed to perform this experimentation is as follows.

Algorithm 2: OptimalUndercover

Input: A matrix $\boldsymbol{X}$ and an integer k.
Output: Two matrices $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ such that $(\boldsymbol{A}\circ\boldsymbol{B})$ is an optimal k-undercover for $\boldsymbol{X}$.

Initialize the formula Φ with constraints (4), (5), (6), (7).
Let $\boldsymbol{S}_{m\times n}$ be a matrix of soft unary clause.
for all $(i,j)\mid\boldsymbol{X}_{i,j}=0$ **do**
  $\Phi\leftarrow\Phi\wedge\bigwedge_{\ell=1}^{k}(\neg\boldsymbol{A}_{i,\ell}\vee\neg\boldsymbol{B}_{\ell,j})$ {Formula (1)}
end for
for all $(i,j)\mid\boldsymbol{X}_{i,j}=1$ **do**
  $\Phi\leftarrow\Phi\wedge\bigwedge_{\ell=1}^{k}(\boldsymbol{T}^{\ell}_{i,j}\Rightarrow\boldsymbol{A}_{i,\ell})$ {Formula (2)}
  $\Phi\leftarrow\Phi\wedge\bigwedge_{\ell=1}^{k}(\boldsymbol{T}^{\ell}_{i,j}\Rightarrow\boldsymbol{B}_{\ell,j})$ {Formula (2)}
  $\Phi\leftarrow\Phi\wedge(\neg\boldsymbol{S}_{i,j}\vee\bigvee_{\ell=1}^{k}\boldsymbol{T}^{\ell}_{i,j})$ {Formula (3)}
end for
$\Phi\leftarrow GenerateCardinalities(\boldsymbol{X},k,\Phi,\boldsymbol{S})$
return $MaxSAT(\Phi,\sum_{i\in[1,m],j\in[1,n]}\boldsymbol{S}_{i,j})$

We randomly generate two matrices $\boldsymbol{A}_{100\times10}$ and $\boldsymbol{B}_{10\times100}$ and assigned the value 1 to each entry with the probability of $\sqrt{1-\sqrt[10]{1-0.1}}$. Thus, ensuring that the probability of having a 1 in a entry of $\boldsymbol{A}\circ\boldsymbol{B}$ is 10%. We then calculate $\boldsymbol{X}=(\boldsymbol{A}\circ\boldsymbol{B})$ and remove 10% of entries randomly.

For k ranging from 1 to 10 we call $OptimalUndercover(\boldsymbol{X},k)$ (Algorithm 2) with or without the GenerateCardinality (Algorithm 1) optimization and compare the time required by many MaxSAT solvers. For the sake of clarity, we plot in Figure 1 the times required to solve the problem for only four MaxSAT solvers.

This experimentation showed that the GenerateCardinality optimization reduced computation time by several orders of magnitude. We do not compare our tool to others in this section because we are not aware of any other tool that can find optimal k-undercover. Note however that the CG method (Kovacs, Günlük, and Hauser 2021), the only tool allowing to infer exact BMF, requires many hours, even with small k (3h30 with $k=2$).

4 Block-Optimal Undercover Factorization

To handle large datasets and to be able to deal with larger ranks, we propose a weaker definition of the optimality of an undercover that we call *block-optimal* undercover.

Definition 5. *Two matrices $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ form a* block-optimal *k-undercover for a matrix $\boldsymbol{X}_{m\times n}$ if for each $p\in[1,k]$, the block $(\boldsymbol{A}_{:,p}\circ\boldsymbol{B}_{p,:})$ is an optimal 1-undercovers for $\boldsymbol{X}-\bigcup_{\ell\neq p}(\boldsymbol{A}_{:,\ell}\circ\boldsymbol{B}_{\ell,:})$.*

The key insight is that two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ form a block-optimal undercover of $\boldsymbol{X}$ if we cannot find a better undercover by modifying a single block. The advantage of this

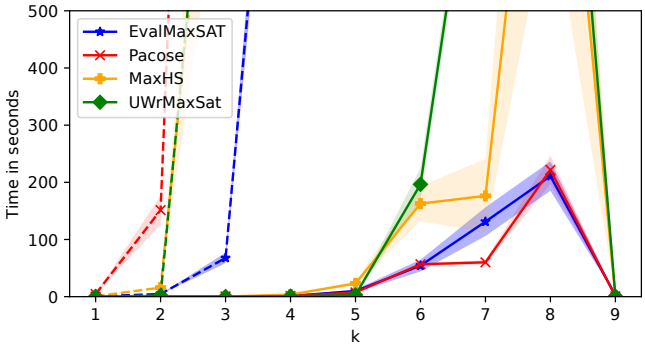

Figure 1: Average execution time over ten runs of OptimalUndercover (Algorithm 2) on a randomly generated dataset using various MaxSAT solvers. The dashed lines correspond to our encoding without the GenerateCardinality optimization and the solid lines correspond to our encoding with the optimization.

definition is that one can find a block-optimal undercover by incrementally improving an initial solution one block at a time. Moreover, finding an appropriate block consists in searching for an optimal 1-undercover, a problem that we are able to solve quickly as we have seen in the previous section. Naturally, it remains that a better solutions can potentially be found by modifying several blocks as allowed by the BMF problem. However, we show in practice that the solutions found with our algorithm are of good quality compared to general BMF algorithms. Furthermore, although under the $P \neq NP$ assumption, it is not possible to approximate the k-undercover Boolean matrix factorization problem with a finite ratio in polynomial time (Miettinen et al. 2008), we show that a block-optimal k-undercover is a $\frac{1}{2}$-approximation.

Theorem 2. *A block-optimal k-undercover is a $\frac{1}{2}$-approximation for the k-undercover Boolean matrix factorization problem.*

Proof. Let $\boldsymbol{X}$ be the matrix to factorize, $\widehat{\boldsymbol{A}} \circ \widehat{\boldsymbol{B}}$ be an optimal k-undercover factorization of $\boldsymbol{X}$ and $\boldsymbol{A} \circ \boldsymbol{B}$ be a block-optimal k-undercover factorization for $\boldsymbol{X}$. Let $\#APX = |\boldsymbol{A} \circ \boldsymbol{B}|_1$ be the number of entries covered by the block-optimal k-undercover.

Note that since $\boldsymbol{A} \circ \boldsymbol{B}$ are block-optimal, a block corresponds to an optimal 1-undercover. Let b_{min} be a block of $\boldsymbol{A} \circ \boldsymbol{B}$ that covers the minimum number of entries not covered by another block. The block b_{min} covers at most $\frac{\#APX}{k}$ entries that are not covered by another block. Let b_{max} be the block of $\widehat{\boldsymbol{A}} \circ \widehat{\boldsymbol{B}}$ that covers the maximum number no of entries not covered by $\boldsymbol{A} \circ \boldsymbol{B}$. Now, no cannot be strictly greater than $\frac{\#APX}{k}$ since otherwise b_{max} would be a better 1-undercover than b_{min}, which contradicts the fact that b_{min} is an optimal 1-undercover.

It then follows that $\widehat{\boldsymbol{A}} \circ \widehat{\boldsymbol{B}}$ covers at most $k \times (\frac{\#APX}{k}) = \#APX$ additional entries compared to $\boldsymbol{A} \circ \boldsymbol{B}$. Therefore $\widehat{\boldsymbol{A}} \circ \widehat{\boldsymbol{B}}$ covers at most twice as many entries than $\boldsymbol{A} \circ \boldsymbol{B}$, establishing $\frac{1}{2}$-approximation. □

Algorithm 3: OptiBlock

Input: A matrix $\boldsymbol{X}_{m,n}$ and two matrices $\boldsymbol{A}_{m\times k}$, $\boldsymbol{B}_{k\times n}$ such that $(\boldsymbol{A} \circ \boldsymbol{B}) \leq \boldsymbol{X}$.
Output: Two matrices that are block-optimal k-undercover for $\boldsymbol{X}$.

repeat
  $modif \leftarrow false$
  for all $p = 1$ to k **do**
    $\boldsymbol{X}' \leftarrow \boldsymbol{X} - \bigcup_{\ell \neq p} (\boldsymbol{A}_{:,\ell} \circ \boldsymbol{B}_{\ell,:})$
    $\boldsymbol{Y}, \boldsymbol{Z} \leftarrow OptimalUndercover(\boldsymbol{X}', 1)$
    if $|\boldsymbol{X}' - (\boldsymbol{Y} \circ \boldsymbol{Z})|_1 < |\boldsymbol{X}' - (\boldsymbol{A}_{:,p} \circ \boldsymbol{B}_{p,:})|_1$ **then**
      $\boldsymbol{A}_{:,p} \leftarrow \boldsymbol{Y}$
      $\boldsymbol{B}_{p,:} \leftarrow \boldsymbol{Z}$
      $modif \leftarrow true$
    end if
  end for
until $modif = false$
return $\boldsymbol{A}, \boldsymbol{B}$

4.1 Algorithm to Find Block-Optimal Undercover

To find a block-optimal k-undercover of a matrix $\boldsymbol{X}_{m,n}$, our method consists in incrementally improving an initial solution until reaching a valid solution.

Let $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ be two matrices such that $(\boldsymbol{A} \circ \boldsymbol{B}) \leq \boldsymbol{X}$. A trivial value for $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ can be $\boldsymbol{A} = \{0\}^{m\times k}$ and $\boldsymbol{B} = \{0\}^{k\times n}$. Then, if $\boldsymbol{A}$ and $\boldsymbol{B}$ are not a block-optimal k-undercover for $\boldsymbol{X}$, then there exists a block $\boldsymbol{A}_{:,p} \circ \boldsymbol{B}_{p,:}$ that is not an optimal 1-undercover for $\boldsymbol{X} - \bigcup_{\ell \neq p} (\boldsymbol{A}_{:,\ell} \circ \boldsymbol{B}_{\ell,:})$. In that case we can replace $\boldsymbol{A}_{:,p}$ and $\boldsymbol{B}_{p,:}$ to have an optimal 1-undercover and iterate until we reach a block-optimal k-undercover for $\boldsymbol{X}$. The pseudo-code of our method is illustrated in Algorithm 3.

Theorem 3. *Algorithm 3 is correct.*

Proof. When the algorithm stops, it means that the variable "modif" is false, i.e., for each $p \in [1, k]$, $(\boldsymbol{A}_{:,p} \circ \boldsymbol{B}_{p,:})$ is an optimal 1-undercovers for $\boldsymbol{X} - \bigcup_{\ell \neq p} (\boldsymbol{A}_{:,\ell} \circ \boldsymbol{B}_{\ell,:})$. Since this property corresponds to definition 5, we know that when we leave the "until" loop, the two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are a block-optimal k-undercover for a matrix $\boldsymbol{X}$.

Let us now show that the algorithm terminates after a finite number of loops. Note that the number of 1's in $\boldsymbol{X}$ not covered by $\boldsymbol{A} \circ \boldsymbol{B}$ can be calculated by the equation $|\boldsymbol{X} - \bigcup_{\ell=1}^{k} (\boldsymbol{A}_{:,\ell} \circ \boldsymbol{B}_{\ell,:})|_1$ which can also be rewritten as:

$$|(\boldsymbol{X} - \bigcup_{\ell \neq p} (\boldsymbol{A}_{:,\ell} \circ \boldsymbol{B}_{\ell,:})) - (\boldsymbol{A}_{:,p} \circ \boldsymbol{B}_{p,:})|_1$$

By replacing $\boldsymbol{X} - \bigcup_{\ell \neq p} (\boldsymbol{A}_{:,\ell} \circ \boldsymbol{B}_{\ell,:}))$ by $\boldsymbol{X}'$ as in Algorithm 3, the equation becomes $|\boldsymbol{X}' - (\boldsymbol{A}_{:,p} \circ \boldsymbol{B}_{p,:})|_1$. Thus, when $|\boldsymbol{X}' - (\boldsymbol{Y} \circ \boldsymbol{Z})|_1 < |\boldsymbol{X}' - (\boldsymbol{A}_{:,p} \circ \boldsymbol{B}_{p,:})|_1$, by replacing $\boldsymbol{A}_{:,p}$ by $\boldsymbol{Y}$ and $\boldsymbol{B}_{p,:}$ by $\boldsymbol{Z}$, the number of 1's in $\boldsymbol{X}$ not covered by $\boldsymbol{A} \circ \boldsymbol{B}$ strictly decreases. Since at least one replacement is necessary to not leave the "until" loop, the number of 1's covered by $\boldsymbol{A} \circ \boldsymbol{B}$ decreases strictly between each new "until" loop, guaranteeing the termination of the algorithm. □

Although any block-optimal k-undercover is a $\frac{1}{2}$-approximation for the k-undercover problem, we show that a $OptiBlock(\boldsymbol{X}_{m\times n}, \{0\}^{m\times k}, \{0\}^{k\times n})$ is a $(1-\frac{1}{e})$-approximation.

Theorem 4. *$OptiBlock(\boldsymbol{X}_{m\times n}, \{0\}^{m\times k}, \{0\}^{k\times n})$ is a $1-(1-\frac{1}{k})^k \leq (1-\frac{1}{e}) \simeq 0.632$ approximation for the k-undercover Boolean matrix factorization problem.*

Proof. Since each block covers a subset of $\boldsymbol{X}$ entries and the problem is to cover as many entries as possible with k blocks among the admissible blocks, the problem can be seen as the Maximum Coverage problem. It is established that the greedy cover algorithm, which consists of selecting at each iteration a block that covers the maximum number of 1 entities not already covered, is a $1-(1-\frac{1}{k})^k \leq (1-\frac{1}{e}) \simeq 0.632$ approximation (Hochbaum 1996).

Since when $\boldsymbol{A} = \{0\}^{m\times k}$ and $\boldsymbol{B} = \{0\}^{k\times n}$, the algorithm $OptiBlock(\boldsymbol{X}, \boldsymbol{A}, \boldsymbol{B})$ behaves for the first k iterations like the greedy cover algorithm, and after the k-th iteration, the number of covered entries can only increase, so it follows that $OptiBlock(\boldsymbol{X}_{m\times n}, \{0\}^{m\times k}, \{0\}^{k\times n})$ is a $1-(1-\frac{1}{k})^k \leq (1-\frac{1}{e}) \simeq 0.632$ approximation for the k-undercover Boolean matrix factorization problem. □

4.2 Using a Good Initial Undercover

We have seen that Algorithm 3 starts with an initial solution $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ such that $\boldsymbol{A}\circ\boldsymbol{B} \leq \boldsymbol{X}$. Although a trivial solution consists in using $\boldsymbol{A} = \{0\}^{m\times k}$ and $\boldsymbol{B} = \{0\}^{k\times n}$, we propose in this section an efficient algorithm to start with a better solution. Note that although this algorithm is used to initialize a solution for $OptiBlock$, it can also be used independently as we will see in the experimentation section.

Our approach involves finding an optimal 1-undercover $\boldsymbol{A}_{:,p}$ and $\boldsymbol{B}_{p,:}$ with the constraint that a particular entry $\boldsymbol{X}_{i,j} = 1$ has to be covered by $\boldsymbol{A}_{:,p}\circ\boldsymbol{B}_{p,:}$. If $\boldsymbol{X}_{i,j} = 1$ is covered by $\boldsymbol{A}_{:,p}\circ\boldsymbol{B}_{p,:}$, we know that for each i' such that $\boldsymbol{X}_{i',j} = 0$, $\boldsymbol{A}_{i',p} = 0$ and for each j' such that $\boldsymbol{X}_{i,j'} = 0$, we have $\boldsymbol{B}_{p,j'} = 0$. Therefore, it is not necessary to consider the entire $\boldsymbol{X}$ matrix and it is sufficient to only keep rows in $\{i' \mid \boldsymbol{X}_{i',j} = 1\}$ and columns in $\{j' \mid \boldsymbol{X}_{i,j'} = 1\}$. In practice, we choose $\boldsymbol{X}_{i,j} = 1$ as a entry to be covered such that $|\boldsymbol{X}_{i,:}|_1 \times |\boldsymbol{X}_{:,j}|_1$ is maximal. The pseudo-code of our method is illustrated in Algorithm 4.

4.3 Experimentation

Our algorithms have been implemented[1] in C++ and in this section, we evaluate our methods FastUndercover (Algorithm 4), OptiBlock (Algorithm 3) with $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ initialized to $\{0\}^{m\times k}$, $\{0\}^{k\times n}$ and OptiBlock* which corresponds to first running FastUndercover, then using the solution found as the initial values of $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{B}_{k\times n}$ for OptiBlock. We have performed experiments on 25 datasets from UCI (Dua and Graff 2017), namely: Audiology, Autism Screening Adult, Balance Scale, Breast Cancer, Car Evaluation, Chess (King-Rook vs. King), Congressional Voting Records, Contraceptive Method Choice, Dermatology, Hepatitis, Iris, Lung Cancer, Lymphography,

[1] See https://github.com/FlorentAvellaneda/UndercoverBMF

Algorithm 4: FastUndercover

Input: A matrix $\boldsymbol{X}_{m,n}$ and an integer k.
Output: Two matrices $\boldsymbol{A}_{m\times k}$, $\boldsymbol{B}_{k\times n}$ such that $(\boldsymbol{A}\circ\boldsymbol{B}) \leq \boldsymbol{X}$

$\boldsymbol{A} \leftarrow \{0\}^{n\times k}$
$\boldsymbol{B} \leftarrow \{0\}^{k\times m}$
for $p = 1$ **to** k **do**
  Let $(i,j) \in \arg\max_{(i,j)}((\sum_{j'=1}^{n} \boldsymbol{X}_{i,j'}) \times (\sum_{i'=1}^{m} \boldsymbol{X}_{i',j}))$
  $I \leftarrow \{i' \mid \boldsymbol{X}_{i',j} = 1\}$
  $J \leftarrow \{j' \mid \boldsymbol{X}_{i,j'} = 1\}$
  $\boldsymbol{A}_{I,\{p\}}, \boldsymbol{B}_{\{p\},J} \leftarrow OptimalUndercover(\boldsymbol{X}_{I,J}, 1)$
  $\boldsymbol{X} \leftarrow \boldsymbol{X} - (\boldsymbol{A}_{:,p}\circ\boldsymbol{B}_{p,:})$
end for
return $\boldsymbol{A}, \boldsymbol{B}$

Mushroom, Nursery, Primary Tumor, Solar Flare, Soybean (Large), Statlog (Heart), Student Performance, Thoracic Surgery, Tic-Tac-Toe Endgame, Website Phishing, Wine and Zoo. All datasets have been binarized with a one-shot encoding when there were more than two categories in a column, and remain binary when there are only two categories. The binarized datasets used are also available in the appendix.

For each dataset, we compute the factorization over three difference rank value $\lceil\frac{kmax}{4}\rceil\times 1$, $\lceil\frac{kmax}{4}\rceil\times 2$ and $\lceil\frac{kmax}{4}\rceil\times 3$ where $kmax$ is a trivial rank for the dataset to factorize (generally the minimum between the number of lines and the number of columns of the dataset).

Figure 2 shows the time required to execute OptiBlock* in solid line, and OptiBlock in dashed line according to the MaxSAT solver used. The execution time of OptimalUndercover (Algorithm 2) is not shown because the size of the datasets and the value of k are too large to expect to obtain a solution in a reasonable time. Note however that for smaller datasets such as Balance, Iris, Vote or Zoo, the al-

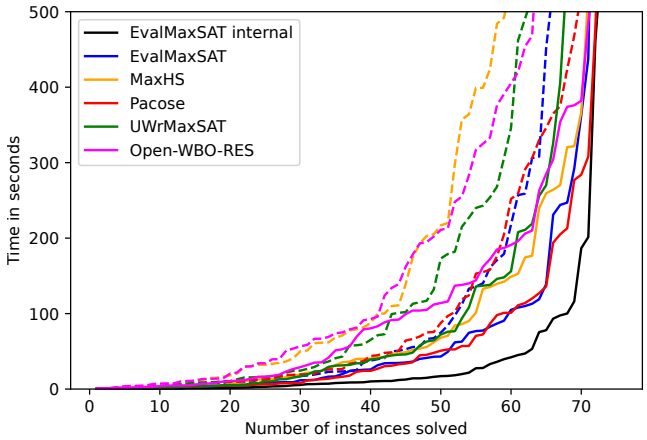

Figure 2: Number x of instances solved in y seconds with different MaxSAT solvers on many real-world datasets. The solid lines correspond to OptiBlock* and the dashed lines correspond to OptiBlock.

gorithm OptimalUndercover can find the optimal solution for small values of k. For example, OptimalUndercover can find an optimal 7-undercover for the dataset Iris in about 100 minutes, while CG is not able to prove the optimality of the solution after 5 hours.

We can see that running FastUndercover first to get an initial solution and then using this initial solution as a starting point for OptiBlock allows us to gain a few orders of magnitude on the execution time. Since the solvers EvalMaxSAT and Pacose seem slightly more efficient on this problem, we have chosen one of them (EvalMaxSAT), and have integrated it in our tool in order to insert the constraints generated by GenerateCardinalities (Algorithm 1) directly in the form of cardinality constraints instead of as a set of clauses. This allows us to save even more computation time and we will only report the results of this solver for the rest of this section. In Figure 2 this solver integration version appears as (EvalMaxSAT internal).

We compare our method to two types of tools: tools that aim to solve the same problem as us, namely to determine a k-undercover; and tools for the general Boolean matrix factorization problem.

We considered the following k-undercover tools:

- "IterEss" (Belohlavek, Outrata, and Trnecka 2019), which is an extension of "GreEss" (Belohlavek and Trnecka 2015).
- "Tile" (Geerts, Goethals, and Mielikäinen 2004), one of the first effective methods to solve this problem.
- "GreConD+" (Belohlavek and Trnecka 2018), which is an extension of the popular "GreConD" method (Belohlavek and Vychodil 2010) and can be configured to calculate k-undercover (rBMF package).

And the following BMF tools:

- "MP" (Ravanbakhsh, Póczos, and Greiner 2016), a message passing technique for approximate a BMF.
- "k-greedy" (Kovacs, Günlük, and Hauser 2021), an heuristic used before calling the "CG" method.
- "CG" (Kovacs, Günlük, and Hauser 2021), an integer programming method using CPLEX to find solutions. CG is an exact method that improves a solution until finding an optimal one. A time budget must be given to obtain a solution in a reasonable time. In our evaluation, we set the time budget to twice the time used by OptiBlock*.
- "Asso" (Miettinen et al. 2008), that is probably one of the most popular tool for BMF (rBMF package).
- "MEBF" (Wan et al. 2020), which uses matrix permutations in order to approximate an upper triangular-like matrix.

We have run each of these tools on the 25 datasets with the three k values discussed above. A detailed table of the results obtained can be found in the appendix. Table 1 summarizes these results through two values: the number of times each tool found a better solution than the other tools (# TOP) and the total time used by each tool to execute all instances.

First of all, we observe that the time used varies a lot from one method to another. We distinguish three groups of methods:

Method	# TOP	Time (min)
FastUndercover	8	3.5
OptiBlock	29	544
OptiBlock*	**45**	98
IterEss	6	0.4
Tile	7	220
GreConD+	9	107
MP	14	372
k-greedy	5	59
CG	19	469
Asso	0	96
MEBF	0	4.0

Table 1: Benchmark on 25 real-world datasets with three values of k (75 cases).

- Fast methods using less than 5 min: FastUndercover, IterEss and MEBF.
- Medium fast methods using between 5 and 100 min: OptiBlock*, k-greedy and Asso.
- Slow methods using more than 100 min: OptiBlock, Tile, GreConD+, MP and CG.

In the fast group, our algorithm FastUndercover finds better quality factorizations, however, IterEss is significantly faster. In the medium fast group, our OptiBlock* method obtains notably higher quality factorizations, even when compared to the slower method. Moreover, although OptiBlock* is constrained to find an undercover, this method still finds better quality factorizations than methods without this constraint such as MP or CG.

5 Conclusion

We have presented two exact methods to infer optimal and block-optimal k-undercover matrix factorizations. Although these two problems are known to be NP-hard, we proposed efficient methods to solve them.

Our first contribution is an efficient MaxSAT formulation for finding optimal k-undercover matrix factorizations. The main idea is to use the knowledge of the problem to find unsatisfiable cores in order to replace many soft clauses by few cardinality constraints. We have shown that this method allows us to gain several orders of magnitude on the computation time.

Our second contribution addressed the scalability issue. We propose a weaker definition of optimality, called block-optimal, and show that every block-optimal solution is a $\frac{1}{2}$-approximation of an optimal solution. In order to find such factorizations efficiently, we propose the OptiBlock* algorithm. This algorithm consists in initializing a solution with our FastUndercover heuristic and then improving this solution until it is block optimal.

The experimental results show that OptiBlock* produces much better reconstruction errors than other tools in the literature and within reasonable computation times.

Dataset	k	FastUndercover		OptiBlock*		IterEss		Tile		GreConD+	
		Errors (number)	Time (sec.)	Errors (number)	Time (sec.)	Errors (number)	Time (sec.)	Errors (number)	Time (sec.)	Errors (number)	Time (sec.)
Car	6	8006	0.2	**7676**	14.2	8006	0.1	8006	0.0	8006	0.3
	12	4838	0.3	**4608**	6.5	4838	0.1	4838	0.0	4838	0.6
	18	2246	0.4	**2060**	3.0	2246	0.1	2246	0.0	2246	0.9
Chess	10	11335	1.6	10454	14.9	11332	0.1	**10435**	0.2	**10435**	1.9
	20	4220	1.8	**3023**	9.0	3052	0.1	3809	0.3	3809	4.0
	30	1346	1.9	**453**	10.4	**453**	0.1	1308	0.3	1353	6.5
Cmc	18	3194	0.5	**2994**	7.1	3114	0.1	3234	0.1	3234	4.3
	36	1314	0.6	**1104**	8.3	1196	0.1	1200	0.1	1200	8.9
	54	391	0.6	279	12.6	346	0.1	347	0.1	347	14.1
Flare	11	2027	0.3	1953	1.8	**1669**	0.1	1699	0.1	1684	1.3
	22	284	0.4	**241**	2.3	429	0.1	354	0.2	346	2.2
	33	33	0.4	**15**	8.0	95	0.1	81	0.2	82	3.6
Heart	68	760	0.4	**728**	17.4	799	0.6	795	0.0	795	107.6
	136	419	0.5	**401**	47.1	462	0.6	441	0.1	444	206.9
	204	222	0.6	**208**	186.6	248	0.6	231	0.1	234	306.2
Iris	32	288	0.1	**282**	1.2	284	0.0	287	0.0	**282**	3.2
	64	122	0.1	**117**	2.9	119	0.0	121	0.0	**117**	6.8
	96	37	0.1	**32**	10.1	37	0.0	36	0.0	34	10.9
Lymph	14	766	0.1	754	0.5	761	0.0	755	0.1	**753**	0.4
	28	275	0.1	**229**	1.1	283	0.0	292	0.2	288	0.8
	42	73	0.1	**33**	1.2	83	0.0	103	0.2	94	1.2
Mushroom	28	39560	48.5	**31226**	1483	33852	2.4	33208	119	33844	176.5
	56	12264	53.9	**7596**	460	9732	2.5	10176	143	9364	337.4
	84	2194	50.7	**436**	587	708	2.6	1554	141	1280	471.7
Nursery	8	73440	5.6	**68004**	1723	**68004**	0.7	**68004**	0.6	69120	4.2
	16	**35970**	9.0	**35970**	78.0	36216	0.7	36216	0.8	36684	9.3
	24	**10698**	10.0	**10698**	53.2	11028	0.7	11028	0.9	12708	15.4
Phishing	7	6029	0.3	6029	1.6	6029	0.1	**6026**	0.1	6029	0.4
	14	3043	0.4	2999	5.0	3043	0.1	**2937**	0.2	3027	0.8
	21	1253	0.5	**806**	3.7	1199	0.1	1172	0.2	1218	1.1
Student	44	3408	0.5	**3213**	18.3	3413	0.4	3400	6.9	3400	21.6
	88	1230	0.6	**967**	19.8	1223	0.4	1234	7.9	1227	42.9
	132	227	0.7	**125**	33.1	271	0.4	305	7.8	298	69.0
Thoracic Surgery	85	662	0.6	**626**	36.1	647	0.7	647	0.0	652	174.8
	170	233	0.9	**219**	100	229	0.7	227	0.0	230	341.3
	255	54	1.0	**51**	201	52	0.7	64	0.1	58	532.5
Tictactoe	7	6068	0.2	**6062**	7.0	**6062**	0.0	**6062**	0.0	6068	0.2
	14	**3588**	0.3	**3588**	1.4	**3588**	0.1	**3588**	0.0	**3588**	0.5
	21	**1518**	0.3	**1518**	1.4	**1518**	0.1	**1518**	0.0	**1518**	0.8
Wine	45	**1766**	0.7	**1766**	12.0	1862	0.6	**1766**	0.1	**1766**	475.5
	90	**1181**	0.9	**1181**	27.9	1232	0.6	**1181**	0.1	**1181**	895.7
	135	596	1.0	**576**	97.6	602	0.6	596	0.2	596	1404.7
Zoo	7	192	0.0	173	0.1	176	0.0	**171**	0.0	178	0.0
	14	86	0.0	**48**	0.1	52	0.0	66	0.0	63	0.1
	21	17	0.0	**6**	0.1	11	0.0	18	0.0	17	0.2

Table 2: Subset of the results obtained for the undercover Boolean matrix factorization problem. All the benchmarks results are available on https://github.com/FlorentAvellaneda/UndercoverBMF.

References

Aloul, F. A.; Ramani, A.; Markov, I. L.; and Sakallah, K. A. 2002. Solving difficult SAT instances in the presence of symmetry. In *Proceedings of the 39th annual Design Automation Conference*, 731–736. ACM.

Aloul, F. A.; Sakallah, K. A.; and Markov, I. L. 2006. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(5): 549–558.

Andres, B.; Kaufmann, B.; Matheis, O.; and Schaub, T. 2012. Unsatisfiability-based optimization in clasp. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Avellaneda, F. 2020. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation 2020*, 8.

Bacchus, F. 2021. MaxHS in the 2021 MaxSat Evaluation. *MaxSAT Evaluation 2021*, 14.

Bacchus, F.; Berg, J.; Järvisalo, M.; and Martins, R. 2021. MaxSAT Evaluation 2021: Solver and Benchmark Descriptions. *SAT*, 2021.

Bacchus, F.; Järvisalo, M.; and Martins, R. 2021. Maximum Satisfiabiliy. In Biere, A.; Heule, M. J. H.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 24, 929–991. IOS Press.

Belohlavek, R.; Outrata, J.; and Trnecka, M. 2019. Factorizing Boolean matrices using formal concepts and iterative usage of essential entries. *Information Sciences*, 489: 37–49.

Belohlavek, R.; and Trnecka, M. 2015. From-below approximations in Boolean matrix factorization: Geometry and new algorithm. *Journal of Computer and System Sciences*, 81(8): 1678–1697.

Belohlavek, R.; and Trnecka, M. 2018. A new algorithm for Boolean matrix factorization which admits overcovering. *Discrete Applied Mathematics*, 249: 36–52.

Belohlavek, R.; and Vychodil, V. 2010. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computer and System Sciences*, 76(1): 3–20.

Brown, C. A.; Finkelstein, L.; and Purdom Jr, P. W. 1988. Backtrack searching in the presence of symmetry. In *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, 99–110. Springer.

Devriendt, J. 2021. Exact: evaluating a pseudo-Boolean solver on MaxSAT problems. *MaxSAT Evaluation 2021*, 12–13.

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

Geerts, F.; Goethals, B.; and Mielikäinen, T. 2004. Tiling databases. In *International conference on discovery science*, 278–289. Springer.

Haddad, A.; Shamsi, F.; Zhu, L.; and Najafizadeh, L. 2018. Identifying Dynamics of Brain Function Via Boolean Matrix Factorization. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 661–665. IEEE.

Hochbaum, D. S. 1996. *Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems*, 94–143. PWS Publishing Co.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2019. RC2: An efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1): 53–64.

Kocayusufoglu, F.; Hoang, M. X.; and Singh, A. K. 2018. Summarizing Network Processes with Network-Constrained Boolean Matrix Factorization. In *IEEE International Conference on Data Mining (ICDM)*, 237–246.

Kovacs, R. A.; Günlük, O.; and Hauser, R. A. 2021. Binary Matrix Factorisation via Column Generation. In *AAAI Conference on Artificial Intelligence, AAAI 2021*, 3823–3831. AAAI Press.

Lei, Z.; Cai, S.; Wang, D.; Peng, Y.; Geng, F.; Wan, D.; Deng, Y.; and Lu, P. 2021. CASHWMaxSAT: Solver Description. *MaxSAT Evaluation 2021*, 8–9.

Lucchese, C.; Orlando, S.; and Perego, R. 2013. A Unifying Framework for Mining Approximate Top-k Binary Patterns. *IEEE Transactions on Knowledge and Data Engineering*, 26(12): 2900–2913.

Martins, R.; Manquinho, V.; and Lynce, I. 2014. Open-WBO: A modular MaxSAT solver. In *International Conference on Theory and Applications of Satisfiability Testing*, 438–445. Springer.

Martins, R.; Manthey, N.; Terra-Neves, M.; Manquinho, V.; and Lynce, I. 2021. Open-WBO MaxSAT Evaluation 2021. *MaxSAT Evaluation 2021*, 15–16.

Miettinen, P.; Mielikäinen, T.; Gionis, A.; Das, G.; and Mannila, H. 2008. The discrete basis problem. *IEEE transactions on knowledge and data engineering*, 20(10): 1348–1362.

Miettinen, P.; and Neumann, S. 2020. Recent Developments in Boolean Matrix Factorization. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 4922–4928. ijcai.org.

Morgado, A.; Dodaro, C.; and Marques-Silva, J. 2014. Core-guided MaxSAT with soft cardinality constraints. In *International Conference on Principles and Practice of Constraint Programming*, 564–573. Springer.

Morgado, A.; Ignatiev, A.; and Marques-Silva, J. 2014. MSCG: Robust core-guided MaxSAT solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 9(1): 129–134.

Paxian, T.; and Becker, B. 2021. Pacose: An Iterative SAT-based MaxSAT Solver. *MaxSAT Evaluation 2021*, 28.

Piotrow, M. 2020. UWrMaxSat: Efficient Solver for MaxSAT and Pseudo-Boolean Problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 132–136. Los Alamitos, CA, USA: IEEE Computer Society.

Piotrów, M. 2021. UWrMaxSat in MaxSAT Evaluation 2021. *MaxSAT Evaluation 2021*, 17–18.

Ravanbakhsh, S.; Póczos, B.; and Greiner, R. 2016. Boolean matrix factorization and noisy completion via message passing. In *International Conference on Machine Learning*, 945–954. PMLR.

Vaidya, J.; Atluri, V.; and Guo, Q. 2007. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, 175–184.

Wan, C.; Chang, W.; Zhao, T.; Li, M.; Cao, S.; and Zhang, C. 2020. Fast and efficient boolean matrix factorization by geometric segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 6086–6093. AAAI Press.

Wicker, J.; Pfahringer, B.; and Kramer, S. 2012. Multi-label classification using boolean matrix decomposition. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 179–186.