Discovering Structure in Genotype Matrices using Boolean Matrix Factorization

Internship Report

Conducted at CBI

From May 1 to July 12 2024

Under the supervision of Prof. Dr. Sven Rahmann.

Alexine

June 19, 2024

Université de Rennes

Abstract

The BMF problem aims to approximate an $m \times n$ Boolean matrix X as the Boolean product of $m \times k$ and $k \times n$ matrices $A \circ B$, defined as $(A \odot B)_{ij} = \bigvee_{r=1}^{k}(A_{ir} \wedge B_{rj})$. Here, k is a critical parameter because it represents the number of factors or patterns used to approximate the original matrix. The goal of BMF is to find the smallest possible k that still provides a good approximation of X.

The importance of having a small k lies in the interpretability and efficiency of the factorization. A smaller k means the data is compressed into fewer patterns, making it easier to analyze and interpret the underlying structure of the genotype matrix, often it can also leads to faster computations and less storage requirement, which can become important when handling large genomic datasets typical in genome-wide association studies (GWAS).

In this report, various BMF algorithms, including GreConD, ASSO, FastUndercover, TopFiberM, and OptiBlock*, are evaluated for their effectiveness and efficiency. Our results demonstrate that GreConD consistently outperforms the other algorithms. Its reconstruction is more accurate, and it has a relatively low computational time, making it a promising candidate for further application in genetic data analysis. Results demonstrate that GreConD consistently outperforms the other algorithms. Its reconstruction is better accurate and he has relatively low computational time, making it a promising candidate for further optimization and application.

Key words : *Combinatorial optimization, Matrix factorizations, Boolean algebra, heuristic algorithms*

Contents

List of Figures

1 Introduction

1.1 Genotype analysis

Based on the comprehensive overview provided by Uffelmann et al. (2021) [9], genome-wide association studies (GWAS) are essential for discovering the genetic basis of many traits and diseases. Genome-wide association studies are research methods used to identify genetic variants associated with specific conditions by examining the genomes of large groups of individuals. This approach involves examining an individual's DNA sequence to identify genetic variations, such as single nucleotide polymorphisms (SNPs). Genotypes represent an individual's unique set of genetic markers, and analyzing these markers helps detect patterns and correlations essential for understanding genetic influences on biological traits. Genotype analysis is key to understanding the complex genomic data underlying biological traits and disease susceptibilities.

To facilitate these analyses, genotype matrices are employed. These matrices provide a structured framework to represent the SNPs of multiple individuals, with m rows corresponding to samples and n columns to SNPs. This setup helps in identifying patterns and correlations essential for understanding genetic variations. As the complexity of analyzing genotype matrices increases, there is a growing need for more tools to manage large datasets. These techniques are important for simplifying the data into manageable components, enabling deeper insights into genomic data.

1.2 Boolean Matrix Factorization techniques

One of the techniques that can be utilized is the Boolean Matrix Factorization (BMF). In fact, genotype matrices can be converted into Boolean matrices, which then allows for their factorization. BMF is a fundamental method used to simplify binary datasets by decomposing a Boolean matrix. It has been applied to various fields like data mining (Miettinen, 2010)[8] and functional interactions in brain networks (Haddad et al., 2018)[7].

Consider a Boolean matrix X of dimensions $m \times n$, where each element x_{ij} is either 0 or 1. BMF aims to find two matrices, A (size $m \times k$) and B (size $k \times n$), such that their Boolean product $A \odot B$ closely approximates X.

This product is defined mathematically as $(A \odot B)_{ij} = \bigvee_{r=1}^{k} (A_{ir} \wedge B_{rj})$, where $\bigvee$ denotes the logical OR and $\wedge$ represents the logical AND. Each row X_i in the matrix X can be viewed as the logical OR of the corresponding rows in B that are selected by the entries in

row A_i of matrix A. More precisely, for each i, the i-th row of X is given by :

$$X_i = \bigvee_{r=1}^{k} (A_{ir} \wedge B_r),$$

where B_r denotes the r-th row of matrix B. The row X_i is constructed by performing a logical OR operation on the rows B wherever A_{ir} is 1. Each column X_j in the matrix X can be viewed as the logical OR of the corresponding colums in A that are selected by the entries in column B_j of matrix B. More precisely, for each j, the j-th row of X is given by :

$$X_i = \bigvee_{r=1}^{k} (A_r \wedge B_{rj}),$$

where A_r denotes the r-th row of matrix A. The column X_j is constructed by performing a logical OR operation on the rows A wherever B_{rj} is 1. This technique simplifies the complexity of X by breaking it down into more manageable components.

Determining the optimal value of k, aiming for a smaller k optimizes the factorization for minimal rank, reducing the dimensionality of the data. Thus, the value of k, which is the number of factors, is important because it significantly impacts the quality of the factorization and the interpretability of the resulting matrices A and B. A smaller k might not capture all the necessary patterns in the data, leading to a poor approximation of the original matrix X. Although, a larger k can lead to overfitting, where the factorization captures noise in the data rather than meaningful patterns. Selecting an appropriate k involves balancing the trade-off between simplicity and accuracy.

This lead to consider two approaches to matrix factorization: exact factorization and error minimization factorization. Exact factorization aims to find matrices A and B such that $A \odot B$ equals X exactly, ensuring that there is no deviation in the approximation. This method is stringent, demanding that the factorization perfectly reconstructs the original matrix X without any discrepancies. On the other hand, error minimization factorization allows for some level of error, aiming to find matrices A and B that minimize the difference between their Boolean product and X. The discrepancy, or error, is quantified as the number of differing entries, formally defined as the count of pairs (i, j) where $(A \odot B)_{ij} \neq X_{ij}$. This approach is particularly advantageous when an exact factorization is unattainable or impractical, as it prioritizes the identification of underlying patterns within the data while mitigating the influence of noise. Error minimization factorization balances the trade-off between achieving a close approximation of the original matrix and simplifying the data structure.

1.3 Context

To effectively manage the vast amount of data generated by GWAS, researchers employ genotype matrices. Processing these matrices can present several challenges due to their size and complexity. Thus, advanced analytical techniques are required to harness these data. BMF techniques provide an efficient method for simplifying and analyzing these complex matrices. By breaking down genotype matrices into manageable components, BMF helps reveal hidden patterns and relationships within genetic data.This technique can facilitate the discovery of new insights in genetic studies. Also, it can contribute to a better understanding of the genetic bases of complex traits.

1.4 Objectives of the internship

The objectives of this internship were: 1) to familiarize myself with the existing literature on BMF. This part required a significant amount of time as it involved understanding the underlying mathematics in scientific papers, and thus learning certain concepts in linear algebra and how matrix factorization works. Next, 2) to apply algorithms or reimplement existing algorithms to apply them to our genotype matrices. Finally, 3) to analyze the obtained results to identify a reasonable value for k and ensure a good reconstruction

2 Materials and methods

SAT and MaxSAT

To understand how the algorithms used in this project work, we need to focus on certain concepts. For FastUndercover and Optiblock*, F. Avellaneda [3] applied SAT and MaxSAT problems to the factorization. SAT (Satisfiability) and MaxSAT (Maximum Satisfiability) approaches are decision problems that determine if there exists an assignment of truth values to variables that satisfies all given logical constraints (clauses). In recent years, this Boolean satisfiability problems has been applied to Boolean Matrix Factorization. [2] The SAT approach is used for the exact factorization of the matrix X, encoding the factorization into a SAT problem.

To ensure accurate factorization, auxiliary variables and constraints are introduced to guarantee that the factorization respects both '0's (preventing false positives) and '1's (ensuring coverage of true positives). The product of matrices A and B should reproduce matrix X as precisely as possible. To reduce the number of equivalent solutions, symmetry break-

ing is proposed, which includes enforcing a lexicographical order on the rows and columns of matrices A and B. By adding supplementary constraints, this technique eliminates symmetrically redundant solutions in the SAT search space.

MaxSAT is an modification of SAT that involves finding an assignment that satisfies the maximum number of clauses in a given Boolean formula. It is used in BMF to allow some level of error, aiming to find matrices A and B that minimize the difference between their Boolean product and X. Two types of constraints are required: hard constraints ensure the factorization does not introduce errors (covering 0s), and soft constraints aim to maximize the correct coverage of 1s, focusing on capturing as many true patterns as possible while allowing for some errors that can be considered as noise.

2.1 Genotype Matrices

The genotype matrix consists of binary data representing individual single nucleotide polymorphisms (SNPs) arranged sequentially within a gene. Each row corresponds to the genetic variation at a specific SNP location for all individuals in the matrix. The binary values in each row denote the genotype of each individual at that SNP location: 0 signifies homozygosity (either reference or variant), 1 denotes heterozygosity, and NA indicates missing or uncertain data.

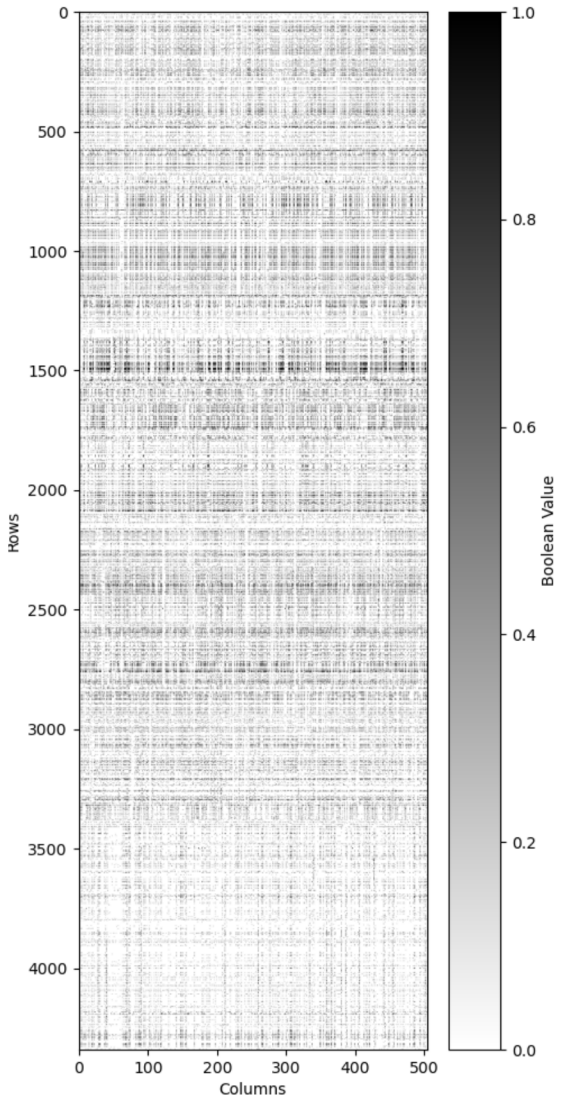

Figure 1: Heatmap of a Boolean Genotype Matrix (Sample 1 - GU)
Visual representation of a genotype matrix with 3101 rows and 504 columns, where black represents 1s and white represents 0s

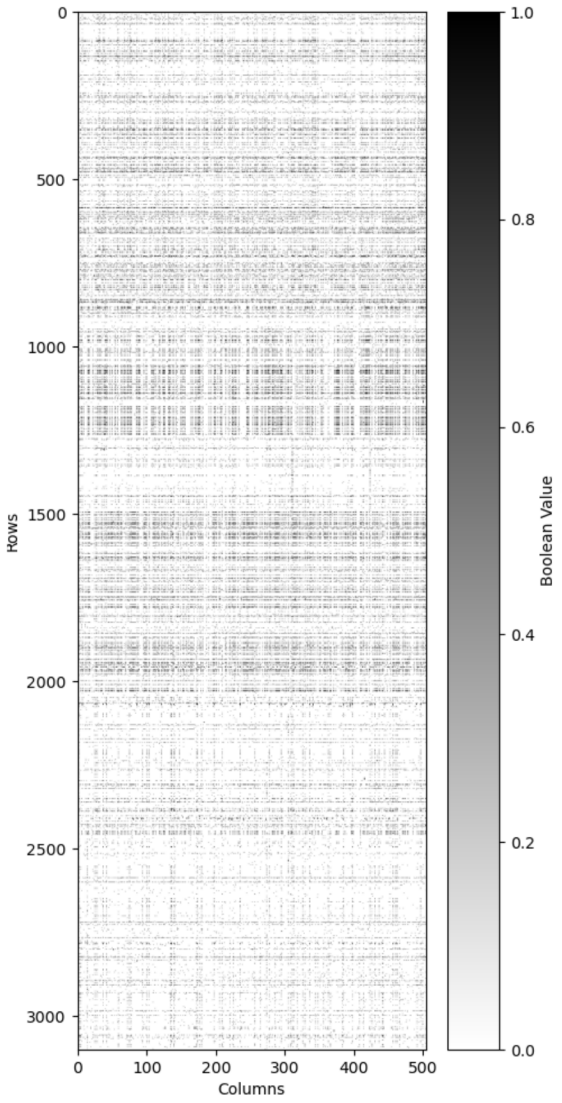

Figure 2: Heatmap of a Boolean Genotype Matrix (Sample 2 - PL)
Visual representation of a genotype matrix with 3101 rows and 504 columns, where black represents 1s and white represents 0s

As illustrated in Figure 1, the heatmap visualization of the genotype matrix reveals recognizable patterns of data points. These patterns are particularly interesting as they suggest the presence of underlying genetic structures and correlations that can be leveraged

for data factorization.

2.2 Algorithms

ASSO

The ASSO algorithm [8] is an early and influential method for approximating Boolean matrix factorization, by constructing binary basis vectors through a greedy approach. It selects and combines attributes based on association confidences, which measures the strength of the relationship between attributes. More precisely, association confidence is defined as :

$$c(i \Rightarrow j; C) = \frac{\langle c_{\cdot i}, c_{\cdot j} \rangle}{\langle c_{\cdot i}, c_{\cdot i} \rangle}$$

where $\langle \cdot, \cdot \rangle$ denotes the vector inner product. This measure indicates the strength of the relationship between attributes, and an association is considered strong if the confidence exceeds a predefined threshold θ.

Then, the algorithm constructs an association matrix A, where each row a_i corresponds to a potential basis vector. Each element a_{ij} of A is set to 1 if the confidence $c(i \Rightarrow j; C)$ is greater than or equal to θ; otherwise, it is set to 0. This matrix A contains candidate basis vectors that the algorithm will consider.

The core of the algorithm is a greedy selection process where it iteratively selects the best basis vectors from A. For each iteration, it adds the basis vector a_i to the matrix B that maximizes the coverage of 1s in C while minimizing the coverage of 0s. This is controlled by a weighted reconstruction error, where correctly covering a 1 is rewarded (bonus) and incorrectly covering a 0 is penalized (penalty).

Mathematically, the coverage function is defined as:

$$\text{cover}(B, S, C, w^+, w^-) = w^+ \sum_{C_{ij}=1} (S \circ B)_{ij} - w^- \sum_{C_{ij}=0} (S \circ B)_{ij}$$

where w^+ is the weight (bonus) for correctly covering a 1, and w^- is the weight (penalty) for incorrectly covering a 0. The algorithm iteratively refines the matrices S and B to maximize this coverage function, effectively fine-tuning the balance between covering 1s and avoiding incorrect coverage of 0s.

This process involves iterating over potential basis vectors, evaluating their contribution to the coverage function, and updating the basis matrix B and the coefficient matrix S accordingly.

GreConD

The GreCon and GreConD algorithms[4], are foundational methods in BMF. While GreCon is no longer primarily used due to its slow running time, GreConD has gained significant popularity and widespread use due to its simplicity and efficiency. It is recognized as one of the fastest BMF algorithms. [1] The *GreConD* algorithm constructs formal concepts on demand rather than generating all of them beforehand. Initially, the algorithm starts with a set of attribute concepts and extends these concepts iteratively by adding attributes that minimize the error in the factorization.

In this context, an "attribute concept" refers to a specific formal concept focusing on the attributes. A formal concept is a pair $\langle A, B\rangle$ consisting of a set of objects A (called the extent) and a set of attributes B (called the intent) such that A is the set of all objects sharing the attributes in B, and B is the set of all attributes shared by the objects in A. The process begins by computing the set of all formal concepts $B(I)$ for the input matrix I. For each formal concept $\langle A, B\rangle$ in the set, the algorithm calculates an initial coverage as the product of the number of objects and attributes, storing it in an array called "covers." Each entry in the matrix I keeps track of which concepts cover it. As long as the product of the factor matrices $AF \circ BF$ does not equal the input matrix I, the algorithm adds the concept $\langle A_l, B_l\rangle$ that covers the most uncovered entries to the set F. For each pair (i, j) covered by this concept, the coverage counts for all concepts covering (i, j) are reduced, and the list for (i, j) is deleted once it is fully covered. This loop continues until $AF \circ BF$ matches I. This greedy approach reduces computational complexity, making it one of the most well-known and effective techniques in the field.

TopFiberM

TopFiberM [6] operates by repeatedly selecting and extending fibers (rows or columns) that optimally represent the input matrix $I \in \{0, 1\}^{n\times m}$. The goal is to approximate I using two Boolean matrices $A \in \{0, 1\}^{n\times k}$ and $B \in \{0, 1\}^{k\times m}$ such that $I \approx A \circ B$, where $(A \circ B)_{ij} = \max_{l=1}^{k} \min(A_{il}, B_{lj})$. Initially, the algorithm sets A_s and B_s to zero matrices of appropriate dimensions and copies the input matrix I into X. In each iteration, the row sum vector r and column sum vector c are computed as:

$$r_i = \sum_{j=1}^{m} X_{ij}, \quad c_j = \sum_{i=1}^{n} X_{ij}$$

[1] In the original work [**?**], these methods were initially referred to as Algorithm 1 and Algorithm 2. They're now widely recognized as GreCon and GreConD

The fiber (row or column) with the maximum sum of ones is identified. If the selected fiber is a row r_i, the algorithm attempts to extend this row by finding similar rows in X according to a precision threshold tP. This is done by comparing each element in the selected row with the corresponding elements in other rows and calculating the proportion of matching elements. Rows that meet or exceed the threshold are considered similar and are added to the fiber. The gain of each fiber is calculated as the difference between the number of true positives (ones correctly covered in X) and false positives (incorrect ones in I). This gain is essential to evaluate the quality of the fiber. The list of top fibers tf stores the index and gain of each fiber. If a newly found fiber has a higher gain than the current minimum gain in the list, it replaces the fiber with the minimum gain. After identifying a fiber and extending it, the algorithm marks the entries covered by this fiber as excluded from further processing. This involves setting the corresponding elements in X to zero. This iterative process continues until the matrix X is completely covered or an acceptable approximation is achieved. The entries covered by the current fiber are then marked as excluded from further processing. This iterative process continues until the matrix X is completely covered or the factorization reaches an acceptable approximation. The final matrices A and B are constructed from the top fibers identified during the iterations. The final matrices A and B are constructed from the top fibers identified during the iterations. Specifically, the indices of the selected top fibers are used to form the corresponding rows and columns in A and B.

FastUndercover

The *FastUndercover* [3] method addresses the problem of BMF by iteratively refining two matrices A and B such that their Boolean product $A \circ B$ approximates a given Boolean matrix X. Both A and B are set to zero matrices. An *undercover* in this context means that the product $A \circ B$ should match the 1s in X as closely as possible while ensuring that no 1s are introduced where X has 0s. This avoids false positives in the factorization. The problem is formulated into a MaxSAT instance with hard and soft constraints. Hard constraints ensure that the Boolean product $A \circ B$ does not introduce false positives. Specifically, for every (i, j) such that $X_{ij} = 0$, the constraint $(A \circ B)_{ij} = 0$ must be met and is represented as :

$$\forall (i, j) \text{ such that } X_{ij} = 0, \quad \neg (A \circ B)_{ij},$$

ensuring that the factorization does not cover any 0s in X. Soft constraints aim to maximize the number of 1s in $A \circ B$ that correspond to 1s in X. The objective is to maximize the sum

$$\sum_{(i,j) \text{ such that } X_{ij}=1} (A \circ B)_{ij},$$

driving the algorithm to cover as many 1s in X as possible. A key feature of the algorithm is the use of cardinality constraints to optimize the solving process. Cardinality constraints are a type of constraint in optimization problems that limit the number of variables that can be true (or selected) within a set. They reduce the number of SAT solver calls and enhance computational efficiency. In the context of FastUndercover, these constraints manage the number of 1s in each row and column of A and B During each iteration, the MaxSAT solver adjusts the values of A and B based on the current solution, with the aim to better satisfy the hard and soft constraints. Thus, the solver attempts to find the best assignments for the entries in A and B that maximize the number of 1s in $A \circ B$ that correspond to 1s in X while ensuring that no 1s are placed where X has 0s.

Optiblock*

The *OptiBlock* method [**?**], is an extension of FastUndercover, introduced in the same study, aims to find block-optimal factorizations. Given two matrices A and B, the method iteratively improves the factorization by optimizing individual blocks. Initially, A and B are set to zero matrices or are initialized using the *FastUndercover* method for a better starting solution. The algorithm then proceeds by selecting each block $A_{:,p}$ and $B_{p,:}$ in turn, and finding an optimal 1-undercover for the submatrix of X not already covered by the other blocks. This process is repeated for each block until no further improvements can be made. The iterative refinement ensures that the solution converges to a block-optimal k-undercover, providing an approximation of the given Boolean matrix X while respecting the undercover constraints.

2.3 Workflow Overview

The different algorithms are integrated into the *Snakemake* workflow manager, which automates the entire process and enhances reproducibility. The majority of the scripts are written in Python, except for FastUndercover and Optiblock*, which are imported from the author's repository. The evaluation was conducted on two matrices: GU.txt and PL.txt. The first step involves reading genotype matrices. The input consists of raw genotype data files in text format. The script readmatrix.py reads the raw matrices and converts them into a compressed format suitable for algorithms. The second step is factorization. Each algorithm takes the matrices as input and outputs a text file with the factorization results, including coverage, the number of false negatives, false positives, and true positives. Different parameters are considered for each algorithm, and a loop is performed for each to test different values of k: 5, 10, 20, 30, 40, 50, and 60.

GreConD

For this algorithm, I reimplemented it in Python to adapt it to our data. The original algorithm can be found in R in the BMF repository from Abdelmoneim Amer Desouki[5].

Algorithm 1 GreConD Algorithm

Input: Binary matrix I, Max factors $max\_factors$
Convert I to binary matrix M
Initialize $U \leftarrow M$, $k \leftarrow 0$
Initialize $A \leftarrow$ empty matrix, $B \leftarrow$ empty matrix
while $\sum U > 0$ **do**
    Initialize $v \leftarrow 0$, $\mathbf{d} \leftarrow \mathbf{0}$, $\mathbf{d}_{old} \leftarrow \mathbf{0}$, $\mathbf{d}_{mid} \leftarrow \mathbf{0}$
    Initialize $\mathbf{e} \leftarrow \mathbf{1}$
    $atr \leftarrow \{j \mid \sum U[:, j] > 0\}$
    while True **do**
        for each $j \in atr$ **do**
            if $d[j] = 0$ **then**
                $\mathbf{a} \leftarrow \mathbf{e} \wedge M[:, j]$
                $\mathbf{b} \leftarrow \bigwedge_{i \in \{i \mid a_i = 1\}} M[i, :]$
                $cost \leftarrow \sum U[a, b]$
                if $cost > v$ **then**
                    $v \leftarrow cost$
                    $\mathbf{d}_{mid} \leftarrow \mathbf{b}$
                    $\mathbf{c}_c \leftarrow \mathbf{a}$
                end if
            end if
        end for
        $\mathbf{d} \leftarrow \mathbf{d}_{mid}$
        $\mathbf{e} \leftarrow \mathbf{c}_c$
        if $\mathbf{d} = \mathbf{d}_{old}$ **then**
            break
        else
            $\mathbf{d}_{old} \leftarrow \mathbf{d}$
        end if
    end while
    $k \leftarrow k + 1$
    Update A and B with $\mathbf{c}_c$ and $\mathbf{d}$
    $U[\mathbf{c}_c, :] \leftarrow U[\mathbf{c}_c, :] \wedge \neg\mathbf{d}$
    if $k \geq max\_factors$ **then**
        break
    end if
end while
Return A, B, k

ASSO

The approximation of ASSO is implemented in RBmf package available on CRAN. To integrate it into my workflow, I wrote a Python script using the rpy2 package to interface with the R implementation. This script takes parameters such as verbose, threshold, penalty coverage, and bonus coverage, mentionned previously.

TopFiberM

TopFiberM is also implemented in RBmf package. TopFiberM is also implemented in the RBmf package. The same process as Asso was followed. It should be noted that it takes different parameters: r values corresponding to the rank k, tP corresponding to the threshold, and search limits corresponding to the maximum number of fibers considered during the search process.

FastUndercover and Optiblock*

These algorithms are imported from Florent Avellaneda's repository[1]. It involved cloning the repository and compiling it to access the different scripts. A simple call to the two algorithms with the k factor is required. Unfortunately, due to the time required by Optiblock* and the short time frame for writing this report, the results are not yet available. However, they will be updated as soon as possible.

The last step of the workflow is to plot the results for visualization. For this, I created three rules in Snakemake: the first rule generates a plot representation for the normalized errors of False Positives (FP) and False Negatives (FN) across various values of k, allowing for an assessment of the algorithms' accuracy. The second rule produces a plot comparing the running times of different algorithms as the k factor varies, highlighting their computational efficiency. The third rule plots the Matthews Correlation Coefficient (MCC) to evaluate the overall performance and reliability of each algorithm. These plots provide a comprehensive overview of the algorithms' effectiveness and efficiency, facilitating an in-depth analysis of their performance. The inputs for these rules include the factorized matrices, the original matrix, and log files from the algorithm executions.

3 Results

3.1 Reconstruction Evalution

After performing matrix factorization using these algorithms, we obtain matrices A and B. The reconstruction $A \odot B$ allows us to evaluate our algorithm's performance by comparing the original matrix M with the reconstructed matrix AB.
Here's a proposed table to represent these values:

Table 1: **Confusion Matrix.**

		M	
		1	**0**
A x B	**1**	TP	FP
	0	FN	TN

TP: Both original and reconstructed matrices have a 1. FP: Original matrix has a 0, reconstructed matrix has a 1. FN: Original matrix has a 1, reconstructed matrix has a 0. TN: Both original and reconstructed matrices have a 0.

The number of false positives and FN false negative are used to evaluate the quality of the reconstruction because they quantify the errors made by the algorithms, indicating how often it incorrectly identifies the elements in the original matrix. The normalized errors for False Positives and False Negatives across various values of k are presented in Figure 3. These normalized errors are critical for evaluating the performance of the differents algorithms in terms of accuracy. The normalization of these errors is performed as :

$$\text{Normalized FP} = \frac{\text{FP Count}}{\text{Total Count of Zeros in the Original Matrix}}$$

$$\text{Normalized FN} = \frac{\text{FN Count}}{\text{Total Count of Zeros in the Original Matrix}}$$

Analyzing these metrics helps in understanding how the number of factors (k) affects the rate of errors. The plot highlights the trend of errors as the k value increases from 5 to 50. It is evident that all algorithms show a decrease in both FN errors with increasing k values, indicating improved performance with larger k values. All algorithms except ASSO maintain a zero FP rate, as their design does not permit false positives. ASSO performs terribly as the number of false positives increases with increasing k values, effectively disqualifying it as a reliable algorithm. A good reconstruction will minimize the number of FP (False Positives)

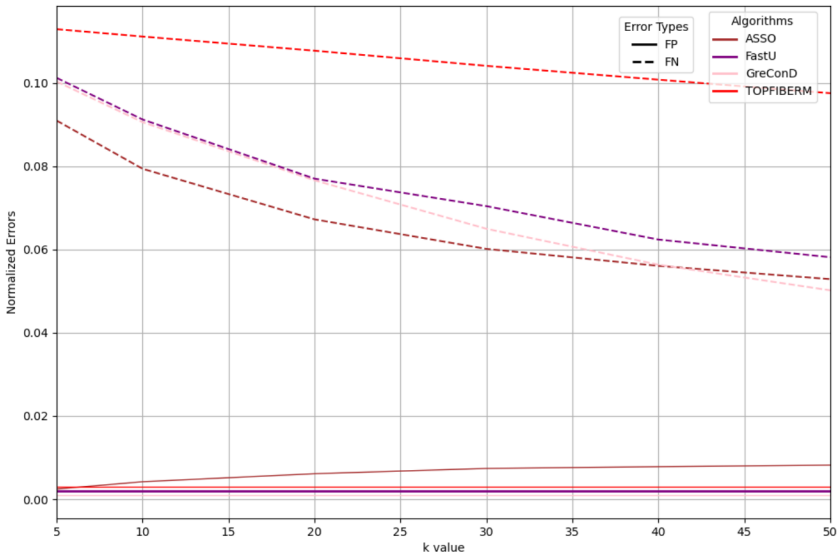

Figure 3:
Chart illustrating the normalized errors for false positives (FP) and false negatives (FN) across various values of k in different algorithms.

and FN (False Negatives). Although the results shown in the plot are closer to 0 than to 1, they still follow the trend as the curves descend, indicating improved performance with increasing k values.

The Matthews Correlation Coefficient (MCC) is a metric that summarizes the confusion matrix of a binary classification, providing a balanced measure even if the classes are of different sizes. It is defined as follow :

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{1}$$

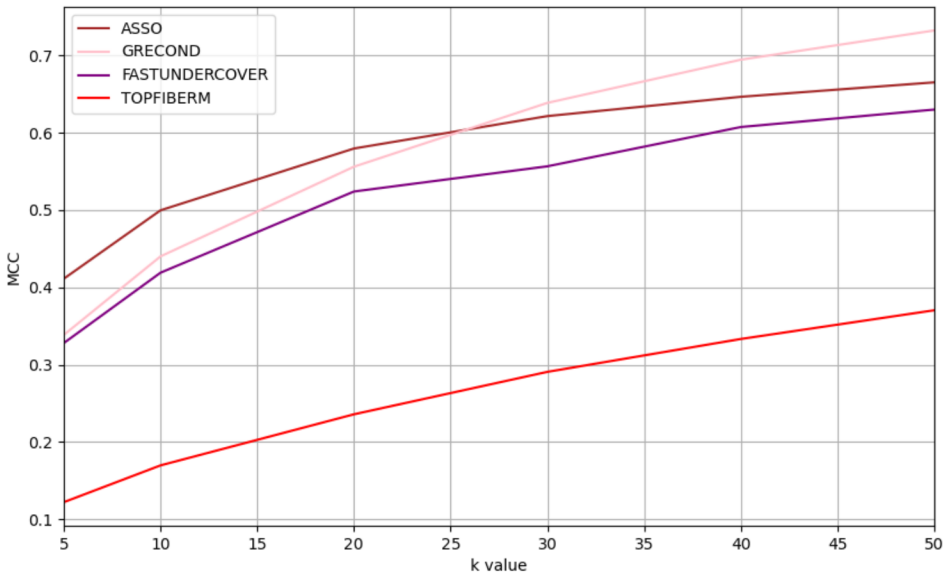

Figure 4: Matthews Correlation Coefficient (MCC) Analysis
Graph comparing the MCC performance of different algorithms across various k values, highlighting their effectiveness in balancing true positives, true negatives, false positives, and false negatives.

The second evaluation assessed the MCC performance of the differents algorithms across various k values, focusing on balancing true positives, true negatives, false positives, and false negatives in reconstruction. GreConD consistently outperformed the other methods, achieving an MCC of around 0.4 at $k = 5$ and improving to about 0.75 at $k = 50$, indicating great perfomance. According to the MCC, scores below 0.7 such as FastUndercover and ASSO are good but not sufficient in the context of our study. TopFiberM has a low score (below 0.3) and thus can't be reliable.

3.2 Running Time Evaluation

We use the running time of the algorithms to evaluate their computational efficiency as it indicates how long each algorithm takes to complete the factorization process. The log files enabled accurate assessment of the computational efficiency of each algorithm. Figure 5, shows that FastUndercover has the highest running time, starting around 200 seconds for $k = 5$ and increasing to over 500 seconds by $k = 50$. Fast Undercover also has a significant running time, beginning at approximately 300 seconds and rising steadily.

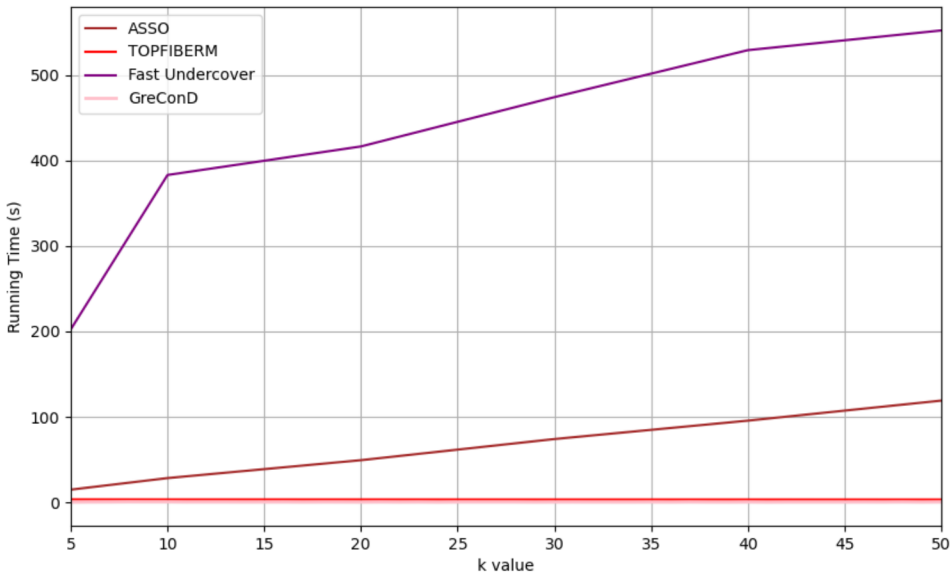

Figure 5: Algorithm Running Time Comparison
Graph showing the running times of various algorithms as the k factor varies, highlighting their computational efficiency.

TopFiberM and GreConD have much lower running times and maintains a nearly constant running time just below 100 seconds across all k values, while ASSO slightly increases but consistently stays under 200 seconds. While all of these running times seems reasonable for a single matrix, they can become prohibitive when applied to a larger set of datasets. These evaluations indicate that while GreConD and FastUndercover may offer better reconstruction quality, FastUndercover do so at the cost of higher running times. On the other hand, ASSO is not suitable for factorization, TOPFIBERM is more computationally efficient.

4 Discussion and Conclusion

The evaluation of matrix factorization algorithms showed varying performance. GreConD was the most effective in balancing true positives and true negatives while minimizing errors, followed by FastUndercover and TopFiberM, with ASSO performing the worst. Despite these results, the reconstructions were mostly not as accurate as desired. GreConD had a "reasonable" running time, and seems to be viable for further analysis. On the other hand, FastUndercover's significantly longer running time makes it impractical despite its moderate performance. ASSO and TopFiberM were more efficient but less accurate.

Most of these algorithms did not meet our expectations for performing efficient factoriza-

tion with a low k value. However, it is worth noting that GreConD shows promising results, and we should consider focusing on this algorithm, potentially exploring optimizations to further improve its score. At this midpoint of the internship, since it is not finished yet, it is difficult to establish a definitive conclusion. It would be interesting to obtain the results of Optiblock* even though the running time does not seem promising. Several approaches can be pursued to enhance the performance of matrix factorization algorithms. Further refining existing algorithms to enhance their accuracy while maintaining reasonable running times is one approach. Conducting extensive parameter optimization is another strategy to improve these algorithms' efficiency and accuracy, which will come later in the internship. Investigating advanced methods such as Graph Neural Networks (GNN) for matrix factorization may provide more accurate reconstructions and better handling of complex data structures.

Bibliography

[1] Florent Avellaneda. Undercoverbmf. https://github.com/FlorentAvellaneda/UndercoverBMF, 2022.

[2] Florent Avellaneda and Roger Villemaire. Boolean Matrix Factorization with SAT and MaxSAT. June 2021. arXiv:2106.10105 [cs].

[3] Florent Avellaneda and Roger Villemaire. Undercover Boolean Matrix Factorization with MaxSAT. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3672–3681, June 2022.

[4] Radim Belohlavek and Vilem Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computer and System Sciences*, 76(1):3–20, February 2010.

[5] Abdelmoneim Amer Desouki. Bmf. `https://github.com/dice-group/BMF`, 2024.

[6] Abdelmoneim Amer Desouki, Michael Röder, and Axel-Cyrille Ngonga Ngomo. topFiberM: Scalable and Efficient Boolean Matrix Factorization, March 2019. arXiv:1903.10326 [cs].

[7] Ali Haddad, Foroogh Shamsi, Li Zhu, and Laleh Najafizadeh. 2018 52nd asilomar conference on signals, systems, and computers. pages 661–665, Oct 2018.

[8] Pauli Miettinen. The Discrete Basis Problem. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 20(10), 2008.

[9] Emil Uffelmann, Qin Qin Huang, Nchangwi Syntia Munung, Jantina De Vries, Yukinori Okada, Alicia R. Martin, Hilary C. Martin, Tuuli Lappalainen, and Danielle Posthuma. Genome-wide association studies. *Nature Reviews Methods Primers*, 1(1):59, August 2021.