

Weighted Rank-One Binary Matrix Factorization

Haibing Lu, Jaideep Vaidya, Vijayalakshmi Atluri, Heechang Shin, [§]Lili Jiang
MSIS and CIMIC, Rutgers University

[§]School of Information Science and Engineering, Lanzhou University, China
{haibing, jsvaidya, atluri, hshin}@cimic.rutgers.edu, [§]jianglili06@lzu.cn

Abstract

Mining discrete patterns in binary data is important for many data analysis tasks, such as data sampling, compression, and clustering. An example is that replacing individual records with their patterns would greatly reduce data size and simplify subsequent data analysis tasks. As a straightforward approach, rank-one binary matrix approximation has been actively studied recently for mining discrete patterns from binary data. It factorizes a binary matrix into the multiplication of one binary pattern vector and one binary presence vector, while minimizing mismatching entries. However, this approach suffers from two serious problems. First, if all records are replaced with their respective patterns, the noise could make as much as 50% in the resulting approximate data. This is because the approach simply assumes that a pattern is present in a record as long as their matching entries are more than their mismatching entries. Second, two error types, 1-becoming-0 and 0-becoming-1, are treated evenly, while in many application domains they are discriminated. To address the two issues, we propose weighted rank-one binary matrix approximation. It enables the trade-off between the accuracy and succinctness in approximate data and allows users to impose their personal preferences on the importance of different error types. The decision problem, however, as proved in the paper is NP-complete. To solve it, several different mathematical programming formulations are provided, from which 2-approximation algorithms are derived for some special cases. An adaptive tabu search heuristic is presented for solving the general problem, and our experimental study shows the effectiveness of the heuristic.

1 Introduction

With the fast development of computer and internet technologies and the decreased cost of data storage devices, a large volume of data are generated and gathered every day. Many datasets have discrete attributes, such as those generated from information retrieval, bio-informatics, and market transactions. Much attention has been focused on efficient techniques for analyzing large and high dimensional datasets. Common tasks for analyzing high dimensional data

include extracting correlations between data items, classification, and clustering data items and finding condensed representations. The analysis of large-scale datasets commonly has to deal with the curse of dimensionality. A useful approach is to compress datasets while preserving important underlying patterns. Conventional matrix factorization techniques can effectively and efficiently compress datasets with continuous attributes. For example, singular value decomposition (SVD) can efficiently reduce a given matrix into a low-rank matrix while minimizing the Frobenius norm of the difference. However, result interpretation is difficult for datasets with discrete attributes.

Existing techniques for mining discrete patterns from binary data include PROXIMUS [13], which can serve the purpose of data compression as well. The idea is to decompose a given binary matrix into a pattern vector and a presence vector, which are restricted to be binary, such that the multiplication of two decomposed vectors has the least mismatching entries with the original binary matrix. It is also called *rank-one binary matrix approximation*. Given the presence vector, the matrix is partitioned into two parts. The part with the presence of the pattern is grouped together. By recursively applying the same process, all row vectors are clustered and their respective patterns are identified. The task of analyzing the original binary matrix can be switched on those mined patterns, which have much smaller size. This technique has received increased attention recently [14, 20]. For example, Shen et al. [20] even provide an efficient 2-approximation algorithm for rank-one binary matrix approximation recursively conducted in a process of PROXIMUS.

PROXIMUS does greatly reduce data size and convey certain original underlying patterns in the approximate matrix. However it suffers from two problems. First, in the worst case the degree of noise in such a generated approximate matrix could be as much as fifty percent. This problem originates from rank-one binary matrix approximation, which is to solve an optimization problem as $\max_{X,Y} \|A - XY^T\|_F$, where A is a given binary matrix, and X and Y are variables restricted to be binary vectors. X and Y are also called presence vector and pattern vector respectively. From the mathematical point, pattern Y is considered to be

present in a row vector of A as long as their matching entries are more than mismatching entries. In other words, in the worst case mismatching values could be as many as matching values. As a result, the final collected patterns could be far different from the original matrix. Analyzing those patterns would hardly give any convincing data analysis result. The second issue is that rank-one binary matrix approximation treats two types of mismatching errors, 1-becoming-0 and 0-becoming-1, evenly. In fact those two types of errors are discriminated by many real applications. For instance, in the application of outlier detection, entries with the value of 1 associated with outliers are relatively sparse. It would be desired to keep entries originally with the value of 1 in the approximate matrix. Otherwise much important information would be lost. On the contrary, when it comes to the role mining problem [16], 0-becoming-1 errors are more disliked. The role mining problem arising from implementing a Role-Based Access Control system, is to mine roles from permissions existing in an organization, which can be represented as a binary matrix. It would be computationally easy to identify promising roles from an approximate permissions matrix with a much simpler structure. However, if the approximate matrix contains many 0-becoming-1 errors, it would induce roles with 0-becoming-1 errors. In semantic it means assigning extra permissions to users, which would cause serious security problems and should be prevented. Based on the above discussion, a sound compression technique on binary datasets should provide much flexibility to users and enable the trade-off between the accuracy and the succinctness of the approximate matrix. In addition, it should allow users to reflect their preferences on error types, which sometimes might be requested by applications themselves.

To address the two identified issues, we propose weighted rank-one binary matrix decomposition. To determine whether a pattern exists in a data vector, three pieces of information need to be considered: (i) components of 1 in both the pattern and the data vector; (ii) components which are 1 in the pattern and 0 in the data vector; and (iii) components which are 0 in the pattern and 1 in the data vector. By applying different weights to the three parts, instead of the same value as in conventional rank-one binary matrix approximation, users can effectively control the level of accuracy in the final approximate matrix and realize their preferences on error types. We call the problem weighted rank-one binary matrix approximation. It is combinatorial in nature. To study it, firstly we relate it to known problems and examine their similarities. Secondly, we formulate the problem in several different mathematical programming forms, from which approximation algorithms are derived for several of its special cases. Thirdly, efficient adaptive tabu search heuristic is presented for the general problems.

2 Background and Related Work

Probabilistic subsampling and data compression are conventional approaches for analysis of large scale data. The general idea of data compression is to find a compact representation for data by discovering important underlying patterns. Matrix factorization techniques are very useful in large scale data analysis, as most datasets can be organized in the form of matrix. Many matrix factorization techniques have been proposed and studied in literature, because in different application domains different expectations on matrix factorization results exist. We tend to classify them, based on data type, into real-valued, non-negative, semi-discrete, and binary.

The most well-known matrix factorization technique on real-valued datasets could be singular value decomposition (SVD) [7], which forms the basis for latent semantic indexing (LSI) commonly used in information retrieval [2]. SVD is closely related to eigenvalue decomposition, such as principal component analysis (PCA) [11]. Notice that PCA and SVD allow real-valued entries in decomposed matrices. A practical shortage is that negative entries cause a very distributed representation and also are hard to interpret.

Non-negative matrix factorization (NMF) [15] was proposed to try to address this interpretability issue. It decomposes matrix $A_{m \times n}$ as $A_{m \times n} \approx UV$, where U and V are non-negative. Non-negative entries enable learning a parts representation of the data and also provide much interpretability to decomposed matrices. Its advantages have attracted much attention from data mining and machine learning fields. But NMF still has the difficulty in interpreting fractional elements.

Semi-discrete decomposition (SDD) restricts the elements in decomposed matrices to be in $\{-1, 0, 1\}$. Its primary advantage is its lower storage requirement. It enables a system to store a high-rank approximation matrix of high accuracy with less storage.

Binary matrix decomposition (BMD) decomposes a binary matrix A as $A = B \otimes C$, where \otimes is the boolean matrix multiplication operator. B and C can respectively be interpreted as a set of concepts and a combination matrix representing each row vector of A as the union of a subset of concepts. It has found many applications such as the role mining problem [16]. BMD has even been extended to include the set difference operation, enabling to capture more real data semantics [17].

3 Problem Statement

Our goal is to find an effective and efficient way to mine discrete patterns from binary matrix data and apply those patterns to compress matrix data and cluster row vectors. In the following, we denote a binary matrix by $A_{m \times n}$ with columns corresponding to n attributes and rows corresponding to m relations (records).

3.1 Motivating Examples Rank-one binary matrix approximation has been actively studied recently to mine discrete patterns and compress binary data. It is to solve $\min_{X,Y} \|A - XY^T\|_F$, where X and Y are binary vectors and called presence vector and pattern vector respectively. Based on entry values of X , rows of A can be divided into two parts. The part associated with entries with the value of 1 in X are considered having pattern Y . By recursively applying rank-one binary matrix approximation, a collection of discrete patterns are mined, which can be utilized to approximate the original data and cluster row vectors.

EXAMPLE 1. Given a matrix A , a rank-one approximation is computed as follows:

$$(3.1) \quad A = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} (1 \ 1 \ 1 \ 1 \ 0) = XY^T$$

The discrete pattern mining approach based on rank-one approximation is straightforward and not difficult to implement. However, it suffers from two serious issues.

The first issue is that according to the objective function $\min_{X,Y} \|A - XY^T\|_F$, a row vector A_i , denoting the i th row of A , is considered having pattern Y as long as their matching components are greater than mismatching components. In other words, the mismatching ratio could be nearly as much as fifty percent. As a result, the final collection of mined discrete patterns cannot serve as a good approximation to the original data matrix and all subsequent data analysis results would be questionable. To illustrate it, we give Example 2. Clearly rank-one binary matrix approximation is not able to divide row vectors of the binary matrix on the left side of Equation (3). As a result, rank-one binary matrix approximation is only able to identify one discrete pattern, while there are two obvious discrete patterns in the original data. Furthermore, the mined pattern $\{1, 1, 1, 1, 0\}$ can hardly represent row vector $\{0, 1, 1, 1, 1\}$, as their matching entries are only one more than mismatching entries.

EXAMPLE 2. The rank-one binary matrix approximation for a binary matrix with two obvious patterns is computed as below:

$$(3.2) \quad \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} (1 \ 1 \ 1 \ 1 \ 0)$$

The second issue with rank-one binary matrix approximation is that it does not support discrimination on 1-becoming-0 errors and 0-becoming-1 errors, which however is requested by many applications. To illustrate it, we give Example 3. With $(1, 1, 1, 1, 0)$ as the pattern to approximate all row vectors, it introduces one 0-becoming-1 error to

the first row vector and one 1-becoming-0 error to the third row vector. For the role mining problem arising from implementing role-based access control [16], the value of 1 in a binary matrix represents a permission assignment. Analyzing an approximate user-to-permission matrix with many 0-becoming-1 errors would generate roles with surplus permissions, which seriously affect system security and safety. In this application, 0-becoming-1 errors should be forbidden.

EXAMPLE 3. Consider the following rank-one approximation:

$$(3.3) \quad \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} (1 \ 1 \ 1 \ 1 \ 0)$$

3.2 Weighted Rank-One Binary Matrix Approximation

Desired features of a discrete pattern mining technique should include allowing the trade-off between the approximation accuracy and the simplicity of mined patterns, and enabling users to impose their preferences on the error type distribution. We achieve them by proposing a new measure on the significance of a pattern in a row vector and a new definition of pattern presence. They are defined as below.

DEFINITION 1. (**Pattern Significance**) The significance of a pattern $Y \in \{0, 1\}^{1 \times n}$ in an object $A \in \{0, 1\}^{1 \times n}$ denoted by $S(Y, A)$, is measured by

$$S(Y, A) = \max\{0, f_{11}(Y, A) - w_1 f_{10}(Y, A) - w_2 f_{01}(Y, A)\}$$

where $f_{ij}(Y, A)$ is the number of attributes which are i in Y and j in A , and w_1 and w_2 are positive weight parameters.

DEFINITION 2. (**Pattern Presence**) If $S(Y, A) > 0$, the pattern $Y \in \{0, 1\}^{1 \times n}$ is considered present in the object $A \in \{0, 1\}^{1 \times n}$.

Weight parameters w_1 and w_2 have two purposes. First, the sum value of w_1 and w_2 can be utilized to control the level of error tolerance, in other words approximation accuracy. The greater sum value leads to less error tolerance, hence higher accuracy in approximation. The ratio between w_1 and w_2 reflects the preferences on two error types. The greater weight means being more disliked.

The essential goal of rank-one binary matrix approximation is to discover a pattern with the most significance in the data matrix. We call such a pattern *dominant discrete pattern*. Weighted rank-one binary matrix approximation is to take penalty weights into account at the basis of conventional rank-one matrix approximation. When $w_1 = 1$ and $w_2 = 1$, the weighted problem is the same as the conventional problem. Given a pattern Y , it is easy to determine presence vector X even with the presence of penalty weights. So instead of looking for the pair of X and Y at the same time,

we consider weighted rank-one binary matrix approximation as the dominant discrete pattern mining problem defined as following.

DEFINITION 3. (Dominant Discrete Pattern Mining) Given m objects consisting of n binary attributes represented by a matrix $A \in \{0, 1\}^{m \times n}$, find a dominant pattern $Y \in \{0, 1\}^{1 \times n}$, such that its total values of pattern significance $\sum_i S(Y, A_i)$ are maximized, where A_i denotes the i th row vector of A .

Two following examples are illustrated to show how weighted rank-one binary matrix approximation can effectively address the two issues with rank-one binary matrix approximation.

EXAMPLE 4. Reconsider the binary matrix in Example 2. By letting $w_1 = w_2 = 2$, its weighted rank-one binary matrix approximation, where the vector on the right side is the dominant discrete pattern, is as below:

$$(3.4) \quad \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} (1 \ 1 \ 1 \ 1 \ 0)$$

In the above example, by doubly penalizing errors, row vectors are successfully divided and $\{1, 1, 1, 1, 0\}$ is indeed a true pattern for rows associated with the presence vector. By applying weighted rank-one binary matrix approximation to the other part, pattern $\{0, 1, 1, 1, 1\}$ is also successfully revealed.

EXAMPLE 5. Look at the binary matrix in Example 3. By letting $w_1 = 1$ and $w_2 = 3$, its weighted rank-one binary matrix approximation is as below:

$$(3.5) \quad \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} (1 \ 1 \ 1 \ 0 \ 0)$$

which is free of 1-becoming-0 errors.

Due to the binary data type, each row can be represented as a subset of n attributes. Such a representation provides a convenient way to describe some relationships between two binary row vectors which we will utilize later. They are defined as following.

DEFINITION 4. (Superset, Strict Superset, Subset, and Strict Subset) For two binary n -dimensional row vectors X and Y , if $Y_i = 1, \forall X_i = 1$, Y is a superset of X and X is a subset of Y . If Y is superset of X and there exists i such that $Y_i = 1$ and $X_i = 0$, Y is a strict superset of X , while X is a strict subset of Y .

4 Relation with Other Problems

The dominant discrete pattern mining problem can be related to many research problems that have been studied in the literature. Those problems can be viewed as either its special case or its variant.

DEFINITION 5. (Maximum Edge Biclique Problem [19]) Given a bipartite graph $G = (V_1 \cup V_2, E)$ and a positive integer K , does G contain a biclique with at least k edges?

A biclique is a kind of bipartite graph where every vertex of the first set is connected to every vertex of the second set. The maximum edge biclique problem defined as above is a special case of the dominant discrete pattern mining problem when w_1 is 0 and w_2 is greater than the maximal number of 1's entries in a row of A , denoted by $\max_i \|A_i\|_1$. With w_1 being 0, $f_{10}(Y, A_i)$ is not counted in the pattern significance formula of $S(Y, A_i)$. With w_2 being greater than $\max_i \|A_i\|_1$, a pattern Y can never be present in a row vector A_i , which is an exact subset of A_i . Therefore the dominant discrete pattern mining problem with $w_1 = 0$ and $w_2 \geq \max_i \|A_i\|_1$ is to find a pattern Y which covers the most 1's entries in row vectors of A , which are supersets of Y . Any binary matrix $A_{m \times n}$ can be expressed as an equivalent bipartite graph $G = (V_1 \cup V_2, E)$, where V_1 has m vertices corresponding to all rows, V_2 has n vertices corresponding to all attributes, and there is an edge connecting vertices $V_1(i)$ and $V_2(j)$ if $A_{i,j} = 1$. As a 1's entry in A corresponds to an edge in its equivalent bipartite graph, a dominant discrete pattern with $w_1 = 0$ and $w_2 \geq \max_i \|A_i\|_1$ induces a maximum edge biclique.

For illustration, consider the binary matrix in Equation (3.1) as an example. It can be expressed as the bipartite graph in Figure 1a. Its dominant discrete pattern with $w_1 = 0$ and w_2 greater than n is given as in Equation (4.6) and its induced biclique is as shown in Figure 1b.

$$(4.6) \quad \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \times (1 \ 1 \ 1 \ 1 \ 0)$$

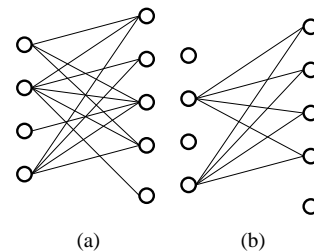


Figure 1: A Bipartite Graph and Its maximum Edge Biclique

DEFINITION 6. (Maximum Tile [4]) Given a database D as a binary matrix $A_{m \times n}$, find the tile with the largest area

in D , where a tile corresponds to a row index subset R and a column index set C , such that $A(i, j) = 1, \forall i \in R, j \in C$.

The maximum tile problem defined as above is essentially to find the largest tile full of 1's entries in a binary matrix, allowing the manipulation on the order of rows and columns. It is equivalent to the maximum edge clique problem, and hence equivalent to the dominant discrete pattern problem with $w_1 = 0$ and $w_2 \geq \max_i \|A_{i:}\|_1$.

Consider the binary matrix on the left side of Equation (4.6) as a tiling database. The largest tile induced by the dominant discrete pattern is as shown in Equation (4.7).

$$(4.7) \quad \left(\begin{array}{cccc|c} 1 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 0 \end{array} \right)$$

DEFINITION 7. (Maximum Edge Weight Biclique Problem [18]) Given a bipartite graph $\{V_1 \cup V_2, E\}$ and weights $\{w_{ij}\}$ associated with edges $\{(V_1(i), V_2(j))\}$ respectively, find a biclique C , where the sum of the edge weights is maximum.

The dominant discrete pattern problem with w_1 and w_2 being equal can be mapped to a special case of the maximum edge weight biclique problem. The maximum edge weight biclique problem is to find a biclique with the maximum weight from a complete bipartite graph. As we have illustrated, any binary matrix $A_{m \times n}$ can be expressed as a bipartite graph $G(V_1 \cup V_2, E)$. We can further expand it as a weighted complete bipartite graph. First complete all missing edges to make it as a complete graph. Then assign a weight of 1 to all edges in the original bipartite graph and a negative weight of $-w_1$ to the newly added edges. A dominant discrete pattern in the original binary matrix would correspond to a maximum weight biclique in the constructed weighted complete bipartite graph.

For illustration, reconsider the binary matrix in Equation (3.1). Its corresponding maximum edge weight biclique problem instance is as shown in Figure 2, where dashed lines are original edges with the weight of 1 and thick lines are added edges with the weight of $-w_1$.

5 Mathematical Programming Formulation

The main result of this section is to present mathematical programming formulations for the dominant discrete pattern mining problem. In particular, we give an unconstrained quadratic binary programming formulation, an integer linear programming formulation, and a relaxed linear programming formulation, which for the case of $w_1 = w_2 = 1$ induces a 2-approximate algorithm.

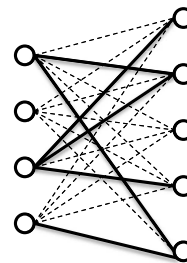


Figure 2: Edge-Weighted Complete Bipartite Graph

5.1 Unconstrained Quadratic Binary Programming

The dominant discrete pattern mining problem is to find the pattern Y maximizing the global pattern significance $\sum_i S(Y, A_{i:})$ for a given binary matrix A . Its corresponding presence vector X can be obtained easily after the dominant discrete pattern Y is found. But in the quadratic binary programming formulation given as below, we treat presence vector X as variables along with pattern vector Y .

$$(5.8) \quad \begin{aligned} & \sum_i S(Y, A_{i:}) \\ &= \sum_i \max\{0, f_{11}(Y, A_{i:}) - w_1 f_{10}(Y, A_{i:}) - w_2 f_{01}(Y, A_{i:})\} \\ &= \sum_i X_i (f_{11}(Y, A_{i:}) - w_1 f_{10}(Y, A_{i:}) - w_2 f_{01}(Y, A_{i:})) \\ &= \sum_i X_i (\sum_j A_{ij} Y_j - w_1 \sum_j (1 - A_{ij}) Y_j \\ & \quad - w_2 \sum_j A_{ij} (1 - Y_j)) \\ &= \sum_{ij} (A_{ij} + w_1 A_{ij} + w_2 A_{ij} - w_1) X_i Y_j - \sum_{ij} w_2 A_{ij} X_i \\ &= X^T U Y - X^T V \vec{1} \end{aligned}$$

where U is defined as $U_{ij} = A_{ij} + w_1 A_{ij} + w_2 A_{ij} - w_1$, V_{ij} is defined as $w_2 A_{ij}$, and $\vec{1}$ denotes the all-ones vector.

The explanation of the above equation deduction process is as follows: (i) The first induction step is obvious; (ii) The second induction step is by the fact that if $f_{11}(Y, A_{i:}) - w_1 f_{10}(Y, A_{i:}) - w_2 f_{01}(Y, A_{i:}) \leq 0$, $X_i = 0$; (iii) In the third induction step, $f_{11}(Y, A_{i:})$ counting the number of attributes where Y and $A_{i:}$ both are 1 is measured by $\sum_j A_{ij} Y_j$, and $f_{10}(Y, A_{i:})$ and $f_{01}(Y, A_{i:})$ are by $(1 - A_{ij}) Y_j$ and $A_{ij} (1 - Y_j)$ respectively; (iv) The last induction step arranges the whole formula as matrix multiplications.

Therefore the dominant discrete pattern mining problem can be simplified as an unconstrained binary quadratic programming problem as below.

$$(5.9) \quad \max\{X^T U Y - X^T V \vec{1} | X \in \{0, 1\}^m, Y \in \{0, 1\}^n\}$$

A binary matrix is closely related to a bipartite graph. It is also a useful approach for problems involved with a bi-

partite graph to be formulated as an unconstrained quadratic binary programming problem [1].

5.2 Integer Linear Programming The unconstrained quadratic binary programming formulation can be linearized as an integer linear programming problem by introducing auxiliary binary variables $\{Z_{ij}\}$ as below.

$$(5.10) \quad \begin{aligned} & \max \sum_{ij} U_{ij} Z_{ij} - \sum_{ij} V_{ij} X_i \\ & \text{s.t.} \begin{cases} -X_i - Y_j + 2Z_{ij} \leq 0 & \forall i, j \\ X_i + Y_j - Z_{ij} \leq 1 & \forall i, j \\ X_i, Y_j, Z_{ij} \in \{0, 1\} \end{cases} \end{aligned}$$

The linearization is done by replacing $X_i Y_j$ with Z_{ij} . As the value of $X_i Y_j$ can only be either 0 or 1, so Z_{ij} is binary. Constraints $-X_i - Y_j + 2Z_{ij} \leq 0$ and $X_i + Y_j - Z_{ij} \leq 1$ guarantee: (i) $Z_{ij} = 1$ when both X_i and Y_j are 1; (ii) $Z_{ij} = 0$ when one of X_i and Y_j is 0.

The linearization skill of replacing $X_i Y_j$ with a binary variable Z_{ij} is also employed by Shen et al. in [20] for formulating the rank-one approximation problem. However their enforcing constraints are as below.

$$(5.11) \quad \begin{cases} -X_i - Y_j + 2Z_{ij} \leq 0 & \text{for } A_{ij} = 1 \\ X_i + Y_j - Z_{ij} \leq 1 & \text{for } A_{ij} = 0 \\ X_i, Y_j, Z_{ij} \in \{0, 1\} \end{cases}$$

Notice that constraints $-X_i - Y_j + 2Z_{ij} \leq 0$ and $X_i + Y_j - Z_{ij} \leq 1$ are not being enforced for every (ij) . Therefore, technically speaking their integer linear programming formulation is not strict, but a relaxed version.

5.3 Linear Programming Relaxation In general the integer linear programming problem is NP-hard, while the linear programming problem is not. It has polynomial algorithms such as interior point method [12] and also has simplex method algorithms which have very good practical performance despite exponential worst-case running time [3]. Thus a typical approach to solve an integer linear programming problem is to derive an approximate solution, also an upper bound to its optimum (if it is a maximization problem) by solving its linear programming relation.

The linear programming relaxation for the dominant discrete pattern mining problem can be easily obtained by simply replacing the constraint set $\{X_i, Y_i, Z_{ij} \in \{0, 1\}\}$ in Equation (5.10) by $\{X_i, Y_i, Z_{ij} \in [0, 1]\}$. The relaxation essentially expands the feasible solution region to a polytope and makes the problem easier to solve. But the optimal solution of the relaxation problem is not necessarily a feasible solution of the original problem, because its components could be fractional. So people usually round either down or up those fractional components to make a feasible solution and use

it as an approximate solution. However, the approximation ratio is not guaranteed.

As we mentioned earlier, the integer programming formulation for the rank-one approximation problem provided by Shen et al. in [20] is not a strict formulation, but a relaxed version. However, such a relaxed integer programming formulation surprisingly leads to a 2-approximation algorithm via simplex methods. We may directly apply their approach to our problem. Accordingly a new linear programming relaxation for Equation (5.12) is given as below.

$$(5.12) \quad \begin{aligned} & \max \sum_{ij} U_{ij} Z_{ij} - \sum_{ij} V_{ij} X_i \\ & \text{s.t.} \begin{cases} -X_i - Y_j + 2Z_{ij} \leq 0 & \text{for } A_{ij} = 1 \\ X_i + Y_j - Z_{ij} \leq 1 & \text{for } A_{ij} = 0 \\ X_i, Y_j, Z_{ij} \in [0, 1] \end{cases} \end{aligned}$$

THEOREM 5.1. *The optimal solution of Equation (5.12) via simplex algorithms is integral, in other words a feasible solution of Equation (5.10).*

Proof. As Equation (5.12) has the same constraints as the linear programming relaxation problem studied in [20], the same result holds that the coefficient matrix of the inequality constraint set is a totally unimodular matrix. A totally unimodular matrix is a matrix for which the determinant of every square non-singular submatrix is 1 or -1. All parameters on the right side of inequalities are also integral. Two properties lead to that the optimal solution of Equation (5.12) via a simplex algorithm is integral. The reason is as following. A simplex method is searching from one basic feasible solution to another basic feasible solution till the optimum is reached. Suppose the constraint set has a standard form as $\{AX = b, X \geq 0\}$, to which any linear constraint set can be transformed. A basic feasible solution is corresponding to a partition $\{X_B, X_N\}$ of variables X and accordingly the constraint set can be reformed as $BX_B + NX_N = b$. By letting $X_N = 0$, $X_B = B^{-1}b$. If $B^{-1}b > 0$, $\{B^{-1}b, 0\}$ is a basic feasible solution. According to the Cramer's rule [6], the i th component of X_B is $\frac{\det(B')}{\det(B)}$, where $\det(B)$ denotes the determinant of B , B' is B except its i th column is replaced by b . Because in Equation (5.12) the coefficient matrix of the constraint set is totally unimodular, the determinant of every square non-singular submatrix is 1 or -1. Hence the denominator of $\frac{\det(B')}{\det(B)}$ is 1 or -1. Since all parameters in Equation (5.12) is integral, the nominator of $\frac{\det(B')}{\det(B)}$ is also integral. Thus every component of any basic feasible solution of Equation (5.12) via a simplex algorithm is integral, and the optimal solution of the relaxation problem is a feasible solution of its original problem.

Our proof is built on the basis of the proof given by Shen et al. [20]. But in addition to pointing out the existence of

two sufficient conditions that a totally unimodular coefficient matrix and all integral parameters, we provide a deep insight on why those two conditions make all basic feasible solutions via a simplex algorithm integral.

Unfortunately, one cannot prove that the optimal solution of Equation (5.12) is 2-approximate to the optimal solution of Equation (5.10). We conjecture that they are at least very close. More importantly the optimal solution of Equation (5.12) is a feasible solution of Equation (5.10). The problem of finding a feasible solution of a system of inequalities in integers in general is NP-complete. Many good heuristics for complex combinatorial optimization problems are starting from a feasible solution and then iteratively improving the current solution by certain rules. A good starting feasible solution such as the optimal solution of Equation (5.12) could save much computing time.

6 Computational Complexity and Approximation Algorithm

The main result of this section is to prove that the decision problem of dominant discrete pattern mining is NP-complete and present 2-approximation algorithms to some special cases of the dominant discrete pattern mining problem.

6.1 Computational Complexity We prove NP-completeness of the decision problem of dominant discrete pattern mining by a reduction from the decision maximum edge biclique problem.

LEMMA 6.1. *The decision maximum edge biclique problem is NP-complete [19].*

THEOREM 6.1. *The decision problem of dominant discrete pattern mining is NP-complete.*

Proof. The decision problem of dominant discrete pattern mining is given a binary matrix $A_{m \times n}$ and a value δ to determine if there is a pattern Y such that $\sum_j S(A_i, Y) \geq \delta$. Given a dominant discrete pattern Y , it is easy to determine whether the instance is true. Thus the decision problem of dominant discrete pattern mining belongs to NP. In the previous section, we showed that the dominant discrete pattern mining problem is a generalization of the maximum edge biclique problem, the decision version of which is NP-complete. Therefore, the decision problem of dominant discrete pattern mining is NP-complete.

6.2 Approximation Algorithm We will study three special cases of the dominant discrete pattern mining problem and present approximation algorithms for them. Before doing that, we first introduce a result.

LEMMA 6.2. *Any integer linear programming problem with n variables subject to m linear constraints with at most*

two variables per inequality, and with all variables bounded between 0 and U , has a 2-approximation algorithm which runs in polynomial time $O(mnU^2 \log(Un^2m))$ [10].

Hochbaum et al. in [10] present a 2-approximation algorithm for integer linear programs with at most two variables per inequality. We briefly introduce this algorithm here. It transforms the integer program into a monotone integer system first ¹, computes an optimal solution for it, and then modifies the result via some simple rule to obtain a feasible solution to the original problem, which is also a 2-approximate solution.

Case 1: $w_1 = 1$ and $w_2 \geq T$ where T is the maximal number of entries with the value of 1 in a row of A .

As we have shown in the preceding section, the dominant discrete pattern mining problem with $w_1 = 1$ and $w_2 \geq T$ is equivalent to the maximum edge biclique problem. Hochbaum in [9] gives a linear programming formulation for the edge weighted biclique problem which is to delete from a bipartite graph $\{V_1, V_2, E\}$, a minimum weight collection of edges so that the remaining edges induce a complete bipartite graph. By slight modifications, we obtain a linear programming formulation for our special case as below.

$$(6.13) \quad \begin{aligned} & \max \sum_{A_{ij}=1} Z_{ij} \\ & \text{s.t.} \begin{cases} 2Z_{ij} - (X_i + Y_j) \leq 0, \text{ for } A_{ij} = 1 \\ X_i + Y_j \leq 1, \text{ for } A_{ij} = 0 \\ X_i, Y_j, Z_{ij} \in \{0, 1\} \end{cases} \end{aligned}$$

Because $w_2 \geq T$, XY^T cannot cover 1's entries in A . In other words $X_i Y_j = 0$ if $A_{ij} = 0$, which is guaranteed by the constraint set $\{X_i + Y_j \leq 1, \text{ for } A_{ij} = 0\}$. The objective function $\sum_{A_{ij}=1} Z_{ij}$ counts (i, j) such that both A_{ij} and Z_{ij} are 1. Z_{ij} can be 1 if and only if both X_i and Y_j are 1, which is guaranteed by the first constraint set $\{2Z_{ij} - (X_i + Y_j) \leq 0, \text{ for } A_{ij} = 1\}$.

The constraint of $2Z_{ij} - (X_i + Y_j) \leq 0$ can be split into two equivalent constraints $Z_{ij} - X_i \leq 0$ and $Z_{ij} - Y_j \leq 0$. The similar skill is also used in [9]. Then Equation (6.13) can be put in the form with at most two variables per inequality as below.

$$(6.14) \quad \begin{aligned} & \max \sum_{A_{ij}=1} Z_{ij} \\ & \text{s.t.} \begin{cases} Z_{ij} - X_i \leq 0, \text{ for } A_{ij} = 1 \\ Z_{ij} - Y_j \leq 0, \text{ for } A_{ij} = 1 \\ X_i + Y_j \leq 1, \text{ for } A_{ij} = 0 \\ X_i, Y_j, Z_{ij} \in \{0, 1\} \end{cases} \end{aligned}$$

¹For the definition of a monotone integer system, refer to the paper [8].

It naturally leads to the following theorem.

THEOREM 6.2. *The dominant discrete pattern mining problem with $w_1 = 1$ and $w_2 \geq T$ is 2-approximable.*

Case 2: $w_1 \geq T$ where T is the maximal number of entries with the value of 1 in a row of A .

$w_1 \geq T$ enforces that a pattern Y can only be present in its subset. In the preceding section, we have shown that the dominant discrete pattern mining problem with $w_1 \geq T$ is equivalent to finding a maximum weight biclique from a complete weighted bipartite graph. In which, the weight for the edges corresponding to entries with the value of 1 in A is 1 and the weight for the other edges is $-w_1$. It is a special case of the edge weighted biclique problem studied in [9]. Again by modifying the linear programming formulation for the edge weighted biclique problem provided in [9], we obtain a linear programming formulation for our problem as below.

$$(6.15) \quad \begin{aligned} \max \quad & \sum_{A_{ij}=1} Z_{ij} - \sum_{A_{ij}=0} w_2 Z_{ij} \\ \text{s.t.} \quad & \begin{cases} Z_{ij} - X_i \leq 0 \\ Z_{ij} - Y_j \leq 0 \\ X_i, Y_j, Z_{ij} \in \{0, 1\} \end{cases} \end{aligned}$$

Equation (6.15) is similar to Equation (6.14), except that: (i) the constraint set of $X_i + Y_j \leq 1$, for $A_{ij} = 0$ is deleted because it is not necessary, and the weights of $\{Z_{ij}\}$ are replaced with 1 and $-w_2$ accordingly. Every inequality constraint of Equation (6.15) has only two variables. Obviously the approximation algorithm presented in [10] applies to it. We state it as below.

THEOREM 6.3. *The dominant discrete pattern mining problem with $w_1 \geq T$ is 2-approximable.*

Case 3: $w_1 = 1$ and $w_2 = 1$.

The dominant discrete pattern mining problem with $w_1 = 1$ and $w_2 = 1$ is equivalent to the rank-one approximation problem. A 2-approximation algorithm via linear programming relaxation is proposed by Shen et al. in [20]. For simplicity, we skip its details.

7 Adaptive Tabu Search Heuristic

A tabu search heuristic is presented for the dominant discrete pattern mining problem. For large-scale problems running time for mathematical programming is always an issue. A 2-approximation algorithm sometimes cannot produce satisfactory results in practice. The dominant discrete pattern mining problem essentially is a combinatorial optimization problem. For such type of problem heuristics usually bring good practical performance. An iterative heuristic is employed for the rank-one approximation problem by both [20]

and [13]. Its basic idea is that starting from a feasible solution of pattern Y and presence vector X , alternatively fixing one of them and then searching for the best solution of the counterpart till both are stable. The only difference between [20] and [13] is that in [20] instead of a random starting solution the starting feasible solution is the optimal solution of Equation (5.12), which is 2-approximate to the optimal solution of the original problem. Their iterative heuristic can be viewed as a greedy heuristic, which greedily picks a neighboring solution at each iteration. One drawback with greedy heuristics is that it is easy to reach local optimums, and once reaching it the solution cannot be improved any more.

To address the local optimum issue, we present an adaptive tabu search heuristic. Tabu search is a mathematical optimization method. It essentially belongs to the class of local search techniques as well. But it enhances the performance of a local search method by using memory structures. When a potential solution has been determined, it is marked as "taboo" so that the algorithm does not visit that possibility repeatedly. In Section 5, we have shown that the dominant discrete pattern mining problem can be formulated as an unconstrained quadratic binary program. In literature, tabu search has been successfully employed to solve quadratic binary programming problems [5]. This is another reason why we choose tabu search.

The dominant discrete pattern mining problem is given a binary matrix A to maximize $\sum_i S(A_i, Y)$. It is easy to compute $S(A_i, Y)$ once Y is determined.

The adaptive tabu search approach taken here is a modification of the adaptive memory tabu search presented in [5]. The basic procedure of our algorithm is as following. It starts with an initial solution of Y and then goes through a series of alternative constructive phases and destructive phases. In the constructive phases, progressively set 0 components of Y to 1, while in the destructive phase, progressively set 1 components of Y to 0. At each iteration, one component of Y is chosen and its value is flipped.

A greedy rule would be choosing the component by changing which improves the objective function the most. However, to avoid visiting some solutions repeatedly, frequency and recency information is utilized. Frequency information is about *critical solutions* encountered to date. A critical solution is the solution at which the next move (either add 1 or drop 1) causes the objective function to decrease. Such an event is called a *critical event*. Recency information is about k critical solutions recently visited. Two pieces of information are stored in two tabu lists respectively. A greedy approach would stop at a critical event. But the tabu search heuristic taken here would keep moving. The number of further moving steps is based on *strategic oscillation*. Such a scheme would enable the solution to step out some local optimums. The amplitude of oscillation, in other words

<i>span</i>	number of further steps after a critical event
<i>span</i> *	maximum value of <i>span</i>
<i>Dir</i>	direction on changing the value of <i>span</i>
<i>p_r</i>	penalty weight from the recency tabu list
<i>p_f</i>	penalty weight from the frequency tabu list
<i>IterCount</i>	count of total iterations
<i>IterCount</i> *	maximum value of <i>IterCount</i>
<i>IterSpan</i>	count of iterations at the current value of <i>span</i>
<i>t</i>	scale parameter on the maximum value of <i>IterSpan</i>

Table 1: Parameter Descriptions

the number of further moving steps, is determined by a parameter, *span*. The span parameter is fixed for a certain number of iterations and then changed in a systematic fashion. To manage the value of *span*, an additional transitive phase is added between each constructive phase and each destructive phase. So technically speaking the adaptive tabu search approach consists of three phases.

ALGORITHM 7.1. Tabu Search Algorithm

Input: $A, IterCount^*, p_r, p_f, span^*, t;$

Output: $X, Y;$

- 1: $Y=0, Y_{best}=0, IterCount=0, span=1, Dir=increase, Count=0, IterSpan=0;$
- 2: **while** $IterCount < IterCount^*$ **do**
- 3: Conduct the constructive phase;
- 4: Conduct the transitive phase;
- 5: Conduct the destructive phase;
- 6: Conduct the transitive phase;
- 7: **end while**

The general structure of the tabu search approach is as outlined in Algorithm 7.1. Parameter descriptions are provided in Table 1. Variable *span* determines the further steps that can be made after encountering a critical solution. *span** is an input parameter, limiting the maximum value of *span*. The value of *Dir* indicates the direction of changing the value of *span*, either increasing or decreasing. *p_r* and *p_f* are penalty weights on information derived from recency and frequency tabu lists respectively. Its usage will be elaborated later. Variable *IterCount* counts the total iterations to date. *IterCount** is an input parameter limiting the maximum value of *IterCount* and hence the algorithm runtime. *IterSpan* counts iterations that have been made at the current value of *span*. *t* is an input scale parameter on the maximum value of *IterSpan*. We explain it explicitly later. The termination condition of Algorithm 7.1 is the total number of iterations less than *IterCount**. It could be replaced with a termination condition on the runtime. In each loop, three phases are traversed in order.

7.1 Constructive Phase In the constructive phase, we progressively pick a 0 component of Y and flip it to 1. The intuitive component-picking policy is to choose the component by flipping which contributes the largest net

increase to the objective function value $\sum_i S(A_{i:}, Y)$. But we want to avoid repeatedly visiting some solutions. To this end, two tabu lists are utilized to influence the search process. The recency tabu list is a vector denoted by V_r , which is the sum of the most recent k critical solutions. At each critical event, it is updated as below:

$$V_r = V_r + Y(current) - Y(current - k),$$

where $Y(current)$ denotes the current critical solution and $Y(current - k)$ denotes the critical solution encountered before k critical events. The frequency tabu list is the sum of all critical solutions encountered so far. At each critical event, it is updated as below:

$$V_f = V_f + Y(current).$$

To avoid repeatedly visiting some critical solutions, we give certain penalty on a 0 component which was 1 in recent critical solutions or was 1 frequently in the past critical solutions. So the final evaluation on a 1 component at the j th position is determined by the following measure:

$$(7.16) \quad f_1(j) = \sum_i S(A_{i:}, Y + e_j) - \sum_i S(A_{i:}, Y) - p_r * V_r(j) - p_f * V_f(j)$$

where e_j is a unit vector with the j th entry of 1, p_r and p_f are penalty weights, and $V_r(j)$ and $V_f(j)$ are the j th entry of V_r and V_f respectively.

The details of the constructive phase are as described in Algorithm 7.2. It consists of two parts. The first part is to iteratively flip the component which maximizes the evaluation formula of Equation (7.16). The second part is to make *span* more iterations after a critical event occurs.

ALGORITHM 7.2. Constructive Phase

- 1: **while** $|Y| < n$ **do**
- 2: Find j' maximizing $f_1(j)$ subject to $Y(j) = 0;$
- 3: **if** $f_1(j') > 0$ **then**
- 4: $Y = Y + e_{j'};$
- 5: $IterCount = IterCount + 1;$
- 6: Update Y_{best} if necessary;
- 7: **end if**
- 8: **end while**
- 9: **while** $CountSpan \leq span$ and $|Y| > 0$ **do**
- 10: $j' = \operatorname{argmax}_{Y(j)=1} (f_2(j));$
- 11: **if** $f_1(j') \leq 0$ **then**
- 12: Update V_r and $V_f;$
- 13: **end if**
- 14: $Y = Y + e_{j'};$
- 15: $CountSpan = CountSpan + 1;$
- 16: $IterCount = IterCount + 1;$
- 17: Update Y_{best} if necessary;
- 18: **end while**
- 19: $CountSpan = 0;$

7.2 Destructive Phase In the destructive phase, we progressively pick a 1 component in Y and change it to 0. The evaluation measure for component picking is as below. It is similar to Equation (7.16) except that we add penalty terms.

$$(7.17) \quad f_2(j) = \sum_i S(A_{i:}, Y - e_j) - \sum_i S(A_{i:}, Y) + p_r * V_r(j) + p_f * V_f(j)$$

The details of the destructive phase is as shown in Algorithm 7.3. It is same as Algorithm 7.2 except that the component picking rule is changed.

ALGORITHM 7.3. Destructive Phase

```

1: while  $|Y| > 0$  do
2:   Find  $j'$  maximizing  $f_2(j)$  subject to  $Y(j) = 1$ .
3:   if  $f_2(j') > 0$  then
4:      $Y = Y - e_{j'}$ ;
5:      $IterCount = IterCount + 1$ ;
6:     Update  $Y_{best}$  if necessary;
7:   end if
8: end while
9: while  $CountSpan \leq span$  and  $|Y| > 0$  do
10:   $j' = \operatorname{argmax}_{Y(j)=1} (f_2(j))$ ;
11:  if  $f_2(j') \leq 0$  then
12:    Update  $V_r$  and  $V_f$ ;
13:  end if
14:   $Y = Y - e_{j'}$ ;
15:   $CountSpan = CountSpan + 1$ ;
16:   $IterCount = IterCount + 1$ ;
17:  Update  $Y_{best}$  if necessary;
18: end while
19:  $CountSpan = 0$ ;

```

7.3 Transitive Phase If $span$ is an unchanged constant, the constitutive phase and the destructive phase constitute a complete tabu search algorithm. A large $span$ would enlarge the search space at the expense of increasing runtime. But with a small $span$ the algorithm might not be able to find a satisfactory solution. To address the issue, adaptive tabu search adjusts the value of $span$ in a systematic fashion. We fix the value of $span$ for a certain number of iterations and then increase it by 1. Keep doing this until $span$ reaches $span^*$, the maximum value predefined. After that, we gradually decrease the value of $span$ by 1. Therefore, the value of $span$ will transverse back and forth between 1 and $span^*$. The complete procedure of transitive phase is as described in Algorithm 7.4.

ALGORITHM 7.4. Transitive Phase

```

1: if  $Dir = increase$  then
2:   if  $span > span^*$  then
3:      $span = span^*$ ;
4:      $Dir = decrease$ ;
5:      $IterSpan = 0$ ;
6:   else
7:     if  $IterSpan > t * span$  then
8:        $span = span + 1$ ;
9:        $IterSpan = 0$ ;
10:    end if
11:  end if
12: else
13:  if  $span = 0$  then
14:     $span = 1$ ;
15:     $Dir = increase$ ;
16:     $IterSpan = 0$ ;
17:  else
18:    if  $IterSpan > t * span$  then
19:       $span = span - 1$ ;
20:       $IterSpan = 0$ ;
21:    end if
22:  end if
23: end if

```

8 Experiments

In this section, we illustrate the properties of weighted rank-one binary matrix approximation by implementing it on synthetic data. The synthetic data are created as following: (i) Generate eight random binary row vectors, such that each vector is of size 1×50 and has exactly $50 * \rho$ components

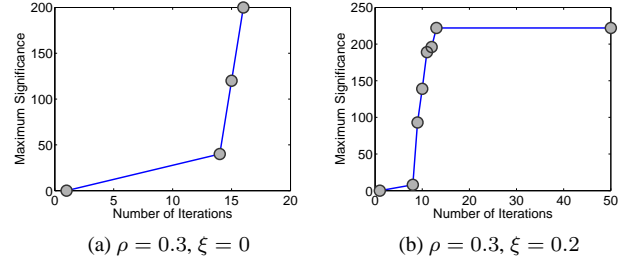


Figure 3: Speed of Convergence

of 1, where ρ is the parameter determining data sparseness and within $[0, 1]$; (ii) Generate 200, 200, 150, 150, 100, 100, 50, and 50 copies respectively for each of the eight vectors, and put them together to constitute a binary matrix with size 1000×50 . Therefore the resultant binary matrix contains eight discrete patterns;(iii) Flip the value for each cell in the binary matrix with probability $\rho * \xi$, where ξ is the parameter determining noise ratio and is within $[0, 1]$.

We first study the performance of the adaptive tabu search heuristic with respect to its speed of convergence. Parameter settings for the adaptive tabu search heuristic are as: $p_r = 1$, $p_f = \frac{IterCount}{50}$, $span^* = 3$, $IterCount^* = 500$, and $t = 3$. We run it on two synthetic binary matrices with generating parameter values as $\{\rho = 0.3, \xi = 0.2\}$ and $\{\rho = 0.3, \xi = 0\}$ respectively. The experimental results are as shown in Figures 3a and 3b. In Figure 3a, it shows that the adaptive tabu search heuristic only takes about 15 iterations to reach a discrete pattern with significance value of 200. The pattern is the global optimum solution, because according to the data generating process with the noise parameter $\xi = 0$ the dominant discrete pattern covers 200 row vectors. In Figure 3b, it shows that the adaptive tabu search heuristic only takes about 15 iterations to reach a pattern with significance value of about 220. As no efficient way to check if the found pattern is dominant, but according to the data generating process, it should be nearly dominant at least. Both graphs verify the high convergence rate of our adaptive tabu search heuristic.

We then evaluate the performance of the adaptive tabu search heuristic by comparing it to the alternating iterative approach proposed in [13] with respect to approximation ratio. The approximation ratio measure is defined as

$$\frac{f_{11} - f_{10} - f_{01}}{\|A\|_1}$$

where f_{11} is the number of matching entries with the value of 1, f_{10} is the number of 1-becoming-0 errors, f_{01} is the number of 0-becoming-1 errors, and $\|A\|_1$ is the number of 1 entries in the original data. For a fair comparison, we let penalty weighs w_1 and w_2 for weighted rank-one binary matrix approximation be 1. Specific parameter setting for the

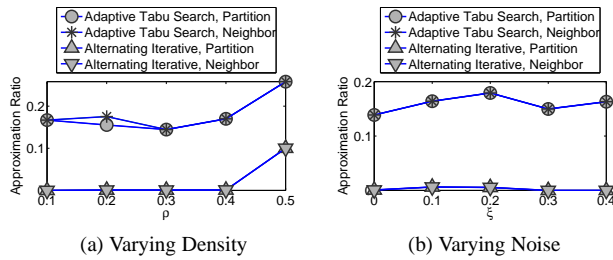


Figure 4: Comparison w.r.t Approximation Ratio on Synthetic Data

adaptive tabu search heuristic is: $p_r = 1$, $p_f = \frac{IterCount}{50}$, $span^* = 3$, $IterCount^* = 500$, and $t = 3$. Regarding p_f , $\frac{IterCount}{50}$ means that the frequency tabu list has more and more impact when the algorithm proceeds.

We compare our algorithm with the alternating iterative approach on various datasets. We first generate data by fixing the noise ratio ξ to be 0.2 and varying the density parameter value ρ from 0.1 to 0.5. As both algorithms require an initial solution. To be favorable to the alternating iterative approach, we employ the *Partition* and *Neighbor* procedures designed for the alternating iterative approach proposed in [13] to generate initial solutions. The experimental results are as shown in Figures 4a. The observation is that the tabu search heuristic performances significantly better than the alternating iterative approach in any case. According to the data-generating process, the dominant discrete pattern should cover about 20 percent of the whole binary matrix. So the maximum approximation ratio is around 0.2. Experimental results show that the adaptive tabu search heuristic produces nearly optimal solutions, while the performance of the alternating iterative approach is unsatisfactory. We then generate data by fixing the density parameter value ρ to be 0.3 and varying noise from 0 to 0.4. The results is as shown in Figure 4b. They again confirm the conclusion that the adaptive tabu search heuristic has significantly better performance.

Next we investigate the impact of penalty weights on the quality and properties of mined patterns. Two measures could be employed to evaluate the quality of mined patterns. The first measure is the number of patterns. The objective of rank-one matrix approximation is to reduce the size of original data by replacing individual records with their corresponding patterns. So less patterns means more succinct approximate data. The second measure is approximation ratio. It is always desirable to retain the original data information in the approximate data, because severely contaminated data would hardly produce any convincing data analysis result.

Again we run our adaptive tabu search heuristic on synthetic data with the same data-generating process as before.

First, we let $w_1 = w_2$ and increase their values gradually. We run our pattern mining algorithm on various data with different data-generating parameter settings. The experimental results are as shown in Figures 5a-5d. In these four graphs, the same observation is that by increasing penalty weights approximation ratio increases consistently. However, with conventional rank-one binary matrix factorization, approximation ratio is not adjustable. When penalty weights are increased to 4, the approximation ratio can be improved to be nearly 1, while the required number of patterns is still significantly less than the number of original records.

We then fix one of penalty weights and vary the other to investigate the impact of penalty weights on error type distribution. First, we fix w_2 to be 1 and then vary w_1 from 1 to 4. The experimental result is as shown in Figure 5e. It is clearly observed that type I error rate decreases consistently with increasing w_1 . When $w_1 = 4$, the type I error rate is as low as 0.1. We then fix w_1 and vary w_2 . The experimental result is as shown in Figure 5f. The same fact is observed. It supports the second main advantage of weighted rank-one binary matrix factorization. By adjusting the value of w_1 and w_2 , users can easily reflect their preferences on error type distribution.

9 Conclusion

In this paper, we propose weighted rank-one binary matrix approximation. In addition to all advantages of conventional rank-one binary matrix approximation, it provides much flexibility to users. By varying the values of penalty weights, users can adjust the size and the accuracy of resulting approximate data and impose their preferences on error type distribution. Theoretical properties of the weighted rank-one binary matrix approximation problem are analyzed. Its relations to other known problems are also investigated. As the decision problem is proven to be NP-complete, an efficient and effective adaptive tabu search heuristic is given. Experimental results show that the presented heuristic performs significantly better than the existing approach, and match all claimed properties of weighted rank-one binary matrix approximation.

References

- [1] B. ALIDAEI, F. GLOVER, G. K. H. W. Solving the maximum edge weight clique problem via unconstrained quadratic programming. *Eur. J. Oper. Res.* 181, 2 (2007), 592–597.
- [2] BERRY, M. W., DUMAIS, S. T., AND O'BRIEN, G. W. Using linear algebra for intelligent information retrieval. *SIAM Rev.* 37, 4 (1995), 573–595.
- [3] DANTZIG, G. *Linear Programming and Extensions*. Princeton University Press, 1998.

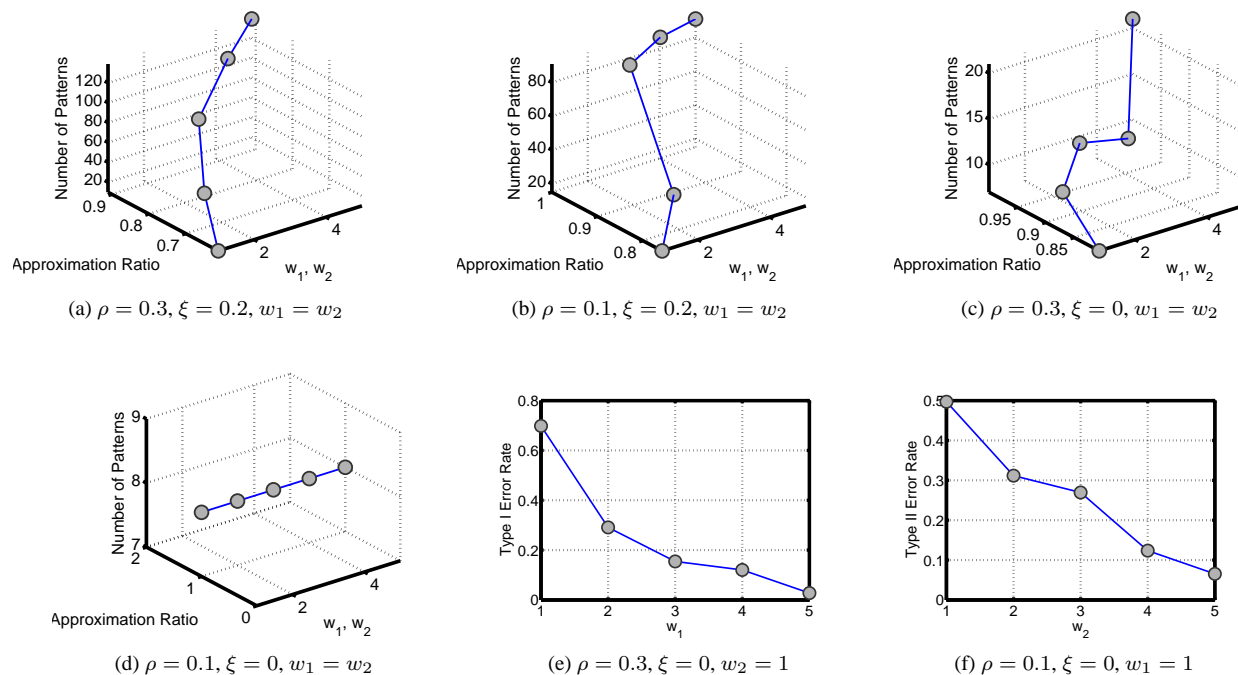


Figure 5: Impact of Penalty Weights on Approximation Ratio, Number of Patterns, and Error Type Distribution

- [4] GEERTS, F., GOETHALS, B., AND MIELIKAINEN, T. Tiling databases. In *Discovery Science* (2004), Springer, pp. 278–289.
- [5] GLOVER, F., KOCHENBERGER, G. A., AND ALIDAEI, B. Adaptive memory tabu search for binary quadratic programs. *Management Science* 44, 3 (1998), 336–345.
- [6] GOLUB, G., AND LOAN, C. V. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996.
- [7] GOLUB, G., AND REINSCH, C. Singular value decomposition and least squares solutions. *Numerische Mathematik* 14, 5 (1970), 403–420.
- [8] HOCHBAUM, D. S. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.* 23, 6 (1994), 1179–1192.
- [9] HOCHBAUM, D. S. Approximating clique and biclique problems. *J. Algorithms* 29 (1998), 174–200.
- [10] HOCHBAUM, D. S., MEGIDDO, N., NAOR, J., AND TAMIR, A. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming* 62 (1992), 69–83.
- [11] HORN, R. A., AND JOHNSON, C. R. *Matrix Analysis*. 1997.
- [12] KARMARKAR, N. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing* (New York, NY, USA, 1984), ACM, pp. 302–311.
- [13] KOYUTÜRK, M., AND GRAMA, A. Proximus: a framework for analyzing very high dimensional discrete-attributed datasets. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2003), ACM, pp. 147–156.
- [14] KOYUTURK, M., GRAMA, A., AND RAMAKRISHNAN, N. Compression, clustering, and pattern discovery in very high-dimensional discrete-attribute data sets. *IEEE Trans. on Knowl. and Data Eng.* 17, 4 (2005), 447–461.
- [15] LEE, D. D., AND SEUNG, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature* 401, 6755 (October 1999), 788–791.
- [16] LU, H., VAIDYA, J., AND ATLURI, V. Optimal boolean matrix decomposition: Application to role engineering. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 297–306.
- [17] LU, H., VAIDYA, J., ATLURI, V., AND HONG, Y. Extended boolean matrix decomposition. In *ICDM '09: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 317–326.
- [18] M. DAWANDE, P. K., AND TAYUR, S. On the biclique problem in bipartite graphs. GSIA working paper 1996-04, Carnegie-Mellon University, 1997.
- [19] PEETERS, R. The maximum edge biclique problem is np-complete. *Discrete Appl. Math.* 131, 3 (2003), 651–654.
- [20] SHEN, B.-H., JI, S., AND YE, J. Mining discrete patterns via binary matrix factorization. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2009), ACM, pp. 757–766.