Nonorthogonal Decomposition of Binary Matrices for Bounded-Error Data Compression and Analysis

MEHMET KOYUTÜRK and ANANTH GRAMA
Department of Computer Sciences, Purdue University
and
NAREN RAMAKRISHNAN
Department of Computer Sciences, Virginia Tech.

This article presents the design and implementation of a software tool, PROXIMUS, for error-bounded approximation of high-dimensional binary attributed datasets based on nonorthogonal decomposition of binary matrices. This tool can be used for analyzing data arising in a variety of domains ranging from commercial to scientific applications. Using a combination of innovative algorithms, novel data structures, and efficient implementation, PROXIMUS demonstrates excellent accuracy, performance, and scalability to large datasets. We experimentally demonstrate these on diverse applications in association rule mining and DNA microarray analysis. In limited beta release, PROXIMUS currently has over 300 installations in over 10 countries.

Categories and Subject Descriptors: G.4 [**Mathematics of Computing**]: Mathematical Software; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*

General Terms: Algorithms

Additional Key Words and Phrases: Compressing binary-valued vectors, nonorthogonal matrix decompositions, semidiscrete decomposition

1. INTRODUCTION

With the availability of large scale computing platforms for high-fidelity simulations and instrumentation for data gathering, increased emphasis is being placed on efficient techniques for analyzing large and high-dimensional datasets. These datasets may comprise discrete attributes such as those from business processes, information retrieval, and bio-informatics, as well as

This research was supported in part by NIH Grant R01 GM068959-01.
Authors' addresses: M. Koyutürk, A. Grama, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907; email: {koyuturk,ayg}@cs.purdue.edu; N. Ramakrishnan, Department of Computer Sciences, Virginia Polytechnic Institute and State University, 660 McBryde Hall, Blackburg, VA 24061; email: naren@cs.vt.edu.

continuous attributes such as those in scientific simulations, astrophysical measurements, and engineering design. An important subclass of applications with discrete datasets restricts data values to a binary subset. Such applications form the focus of this article.

Analysis of high-dimensional data typically takes the form of extracting correlations between data items, discovering meaningful information in data, clustering data items, and finding efficient representations for clustered data, classification, and event association. Since the volume (and dimensionality) of data is typically large, the emphasis of new algorithms must be on efficiency and scalability. Analysis of continuous attribute data generally takes the form of Eigenvalue/singular value problems (PCA/rank reduction), clustering, least squares problems, etc. Analysis of discrete datasets, however, generally leads to NP-complete/hard problems, especially when physically interpretable results in discrete spaces are desired. Consequently, the focus here is on effective heuristics for reducing the problem size. Two possible approaches to this problem are probabilistic subsampling and data compression. This article focuses on algorithms and heuristics for error-bounded compression of very large high-dimensional binary-attributed datasets.

Compression of binary data is a particularly challenging problem when compressed data is required to directly convey the underlying patterns in the data. Conventional techniques such as singular value decomposition (SVD), frequency transforms such as discrete cosine transforms (DCT) and wavelets, and others cannot be used here because the compressed data (orthogonalized vectors or frequency coefficients) are not directly interpretable as signals in noisy data. Techniques for clustering do not generalize easily to high-dimensions (10^4 or more) while yielding error-bounded cluster centroids. Unfortunately, the runtimes of all these methods are unacceptably large when scaled to millions of records (vectors), or more.

In this article, we present the design and implementation of a software tool for compressing binary matrices. This tool, PROXIMUS, is demonstrated to have excellent compression properties, compression times, and scalability in terms of dimensionality and dataset size. PROXIMUS implements a nonorthogonal matrix transform based on recursive partitioning of a dataset. The partitioning process extracts a representative pattern in the dataset and uses this pattern to divide the dataset into two, based on the distance of a relation from the representative pattern. The representative pattern is computed as a binary representative vector of the matrix of relations. PROXIMUS computes only the first binary representative vector and consequently, each representative pattern has a physical interpretation at all levels in the hierarchy of the recursive process. For the discovery of the representative binary vector, we adopt an iterative alternating heuristic. Due to the discrete nature of the problem, initialization of binary representative vectors is critical for convergence to desirable local optima. Taking this into account, we derive effective initialization strategies, along with algorithms, data structures, and efficient implementation schemes for a multiresolution representation of the dataset.

PROXIMUS provides several features that can be used to analyze binary attributed data. These include:

—discovering dominant and deviant patterns in the data in a hierarchical manner (deviant patterns are those that are largely orthogonal to all dominant patterns),
—clustering of data in an error-bounded and physically interpretable form,
—finding a concise representation for the data, and
—isolating signal from noise in a multiresolution framework.

It is important to note that the binary nature of data makes it critical to use appropriate data structures and algorithms. Naive storage, data movement, and computation schemes can easily overwhelm conventional computing platforms. As we demonstrate in our results, PROXIMUS couples fast algorithms with extremely efficient data structures and implementation to provide a powerful software framework. It is also useful to note that the techniques presented in this article for binary datasets can be extended to arbitrary discrete datasets (by allowing arbitrary singular values in conjunction with binary valued singular vectors) [Zyto et al. 2002].

In the next section, we discuss the use of matrix transforms in the context of data analysis and compression and review existing approaches. In Section 3, we present the mathematical underpinnings of PROXIMUS using representative examples. We present the implementation of PROXIMUS, emphasizing the design of suitable data structures and efficient implementation schemes, in Section 4. We demonstrate the effectiveness of PROXIMUS on both synthetic and experimental data and explore the effect of various parameters on the quality of approximations in Section 5. We also illustrate the scalability of PROXIMUS to extremely large datasets and present sample applications of PROXIMUS in diverse domains. Finally, in Section 6, we draw conclusions and outline some avenues for future research.

2. BACKGROUND AND RELATED WORK

Conventional approaches to analysis of large scale data focus on probabilistic subsampling and data compression. Data reduction techniques based on probabilistic subsampling have been explored by several researchers [John and Langley 1996; Provost and Kolluri 1999; Toivonen 1996; Zaki et al. 1996]. Data compression techniques are generally based on the idea of finding compact representations for data through discovery of dominant patterns or signals. A natural way of compressing data relies on matrix transforms, which have found various applications in large scale data analysis. Variants of orthogonal and nonorthogonal matrix transformations such as truncated singular value decomposition (SVD), semidiscrete decomposition (SDD), centroid decomposition (CD), and principal direction divisive partitioning (PDDP) have been widely used in information retrieval and data mining [Berry et al. 1995; Boley 1998; Chu and Funderlic 2002; Kolda and O'Leary 1998, 2000]. In the rest of this section, we summarize commonly used orthogonal and nonorthogonal matrix transformations and their applications in data analysis and explore alternative approaches for binary datasets.

2.1 Rank Reduction and the Singular Value Decomposition (SVD)

SVD transforms a matrix into two orthogonal matrices and a diagonal matrix of singular values, based on the Eigen-decomposition of matrices AA^T and A^TA. Specifically, an m by n rectangular matrix A can be decomposed into

$$A = X\Sigma Y^T, \tag{1}$$

where X is an $m \times r$ orthogonal matrix, Y is an $n \times r$ orthogonal matrix, and Σ is an $r \times r$ diagonal matrix of the singular values of A in descending order. Here, r denotes the rank of matrix A. The matrix $\tilde{A} = x_1\sigma_1 y_1^T$ is a rank-one approximation of A, where x_1 and y_1 denote the first columns of matrices X and Y, respectively. These vectors are the left and right singular vectors of A corresponding to the largest singular value.

If we think of a matrix as a multiattributed dataset with rows corresponding to relations and columns corresponding to attributes, we can say that each 3-tuple consisting of a singular value σ_k, k^{th} column in X, and k^{th} column in Y represents a pattern in A characterized by corresponding singular value σ_k. For larger singular values (relative to other singular values), the corresponding pattern is more dominant in the dataset. SVD forms the basis for latent semantic indexing (LSI) commonly used in information retrieval [Berry et al. 1995], which takes advantage of this property of SVD for rank reduction and noise elimination. LSI summarizes the underlying data represented by matrix A by truncating the SVD of A to an appropriate number of dominant singular values. In doing so, the insignificant patterns corresponding to small singular values are filtered.

2.2 Semidiscrete Decomposition (SDD)

SDD is a nonorthogonal matrix decomposition in which the values of the entries in matrices X and Y are constrained to be in the set $\{-1, 0, 1\}$ [Kolda and O'Leary 2000]. The main advantage of SDD is its lower storage requirement, which enables a higher rank representation for a given amount of memory. SDD applied to LSI is shown to do as well as truncated SVD, while using less than one-tenth of the storage [Kolda and O'Leary 1998]. SDD also finds application in image compression and pattern matching and has been shown to provide fast and accurate pattern matching, though performing slightly worse than DCT-based image compression [Zyto et al. 2002]. McConnell and Skillicorn [2001] show that SDD differs from SVD in that it is extremely effective in finding outlier clusters in datasets and works well in information retrieval for datasets containing a large number of small clusters.

Since the entries of the semidiscrete decomposition vectors are constrained to be in the set $\{-1,0,1\}$, computation of SDD becomes an integer programming problem which is NP-hard. Kolda and O'Leary [2000] propose an iterative alternating heuristic to solve the problem of finding rank-one approximations to a matrix in polynomial time. Each iteration of this heuristic has linear time complexity.

2.3 Centroid Decomposition (CD)

Centroid Decomposition (CD) is an approximation of SVD that is widely used in factor analysis. It has been shown empirically that CD provides an estimate of second order statistical information of the original data [Chu and Funderlic 2002]. CD represents the underlying matrix in terms of centroid factors that can be calculated without knowledge of the entire matrix; the computation only depends on the correlations between the rows of the matrix. Centroid factors are computed via the centroid method, which is a fast iterative heuristic for partitioning the data. This heuristic aims to modify the coordinate system to increase the eccentricity of the system variables with respect to the origin. The transformation aims to move the discovered centroid far away from the origin, so that it represents a better essential factor. Centroid method requires knowledge of correlations between all pairs of rows. This requires quadratic time and space in the number of rows. Thus, while adapting centroid method to binary data, an alternative for the correlation matrix must be provided that is much sparser and takes advantage of the discrete nature of data.

2.4 Principal Direction Divisive Partitioning (PDDP)

Principal direction divisive partitioning (PDDP) is a hierarchical clustering strategy for high-dimensional real-valued sparse datasets [Boley 1998]. PDDP splits documents (rows) into two parts, recursively, based on the principal direction of the document-term matrix. Here, principal direction corresponds to the first singular vector of the matrix obtained by moving the centroid of the original matrix to the origin. The idea of recursively partitioning the matrix based on the first singular vector is similar to that used by PROXIMUS. However, PROXIMUS is designed specifically for binary-attributed data and works on the original matrix rather than moving its centroid to the origin, in contrast to PDDP. For these reasons, PROXIMUS is significantly faster than PDDP.

2.5 Other Work on Summarizing Discrete-Attribute Datasets

Other work on summarizing discrete-attributed datasets has largely focused on clustering very large categorical datasets. A class of approaches is based on well-known techniques such as *vector-quantization* [Gray 1984] and *k-means clustering* [MacQueen 1967]. The *k-modes* algorithm [Huang 1997] extends k-means to the discrete domain by defining new dissimilarity measures. Another class of algorithms is based on similarity graphs and hypergraphs. These methods represent the data as a graph or hypergraph and apply partitioning heuristics to this representation. Graph-based approaches represent similarity between pairs of data items using weights assigned to edges and define cost functions on this similarity graph [Gibson et al. 1998; Guha et al. 2000; Gupta and Ghosh 2001]. Hypergraph-based approaches observe that binary-attributed datasets are naturally described by hypergraphs and directly define cost functions on the corresponding hypergraph [Han et al. 1998; Özdal and Aykanat 2004].

Our approach differs from these methods in that it discovers naturally occurring patterns with no constraint on cluster sizes or the number of clusters. Thus, it provides a generic interface to the problem which may be used in diverse

applications, as we demonstrate in our experimental results. Furthermore, the superior execution characteristics of our approach make it particularly suited to high-dimensional attribute sets.

3. PROXIMUS: MATHEMATICAL FOUNDATIONS

PROXIMUS relies on nonorthogonal decomposition of binary matrices to extract patterns. In many applications, the underlying data can be represented as a matrix with rows representing relations with binary attributes, or feature vectors with binary-valued features. For such a representation, the proposed nonorthogonal decomposition of binary matrices finds a minimal set of representative patterns for the rows and associates each row with a pattern. This results in a compact and error-bounded approximation for the given matrix.

The problem of extracting patterns from a given set of binary vectors can be stated as follows: Given m binary vectors defined over an n-dimensional space, construct a summary of $k \ll n$ (representative) vectors such that each of the given vectors is within given bounded distance from some representative vector. A variety of distance metrics can be used to measure the distance between binary vectors. PROXIMUS uses Hamming distance as its metric, since it is well suited to binary spaces. The Hamming distance between two binary vectors is defined as the number of entries that are different in the two vectors.

A binary rank-one approximation to a binary matrix is defined as an outer product of two binary vectors that is at minimum Hamming distance from the matrix over all outer products of the same size. In other words, the rank-one approximation problem for matrix A with m columns and n rows is one of finding two vectors x and y that maximize the number of zeros in the matrix $(A - xy^T)$, where x and y are of dimensions m and n, respectively. The following example illustrates this concept:

Example 3.1. Given a matrix A, we compute a rank-one approximation as follows:

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} = xy^T$$

Here, vector y can be thought of as the *pattern vector*, which is the best approximation for the objective (error) function specified. In our case, this vector is $[1\ 1\ 0]^T$. Vector x is the *presence vector* representing the rows of A that are well approximated by the pattern described by y. Since all rows contain the same pattern in this rank-one matrix, x is vector of all ones. We further clarify this discussion with a slightly nontrivial example.

Example 3.2. Consider now a binary matrix A, which does not have an exact rank-one representation (*i.e.*, the matrix is of higher rank).

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Table I. Summary of Notation

Symbol	Meaning	Symbol	Meaning
A	Binary matrix	$\bar{A}$	Representative matrix
m	Number of rows (context)	m_A	Number of rows in A
n	Number of columns (context)	n_A	Number of columns in A
N	Number of nonzeros (context)	N_A	Number of nonzeros in A
$A(i)$	i^{th} row of matrix A	$A(i, j)$	i, j^{th} entry of A
A_i	Submatrix of A		
x	Binary presence vector	y	Binary pattern vector
X	Binary presence matrix	Y	Binary pattern matrix
$x(i)$	i^{th} entry of x	k	Number of representative vector pairs
ϵ	Bound on Hamming radius	d	Height of recursion tree
$h(x, y)$	Hamming distance	$r(A, y)$	Hamming radius
$\|\|.\|\|_F$	Frobenius-norm of a matrix	$\|\|.\|\|_2$	2-norm of a vector
$O(.)$	Asymptotic upper bound	$\Theta(.)$	Asymptotic upper & lower bound

The pattern vector here is $[0\ 0\ 1\ 0\ 1]^T$ and the corresponding presence vector is $[1\ 1\ 0\ 1]^T$. This presence vector indicates that the pattern is dominant in the first, second, and fourth rows of A. A quick examination of the matrix confirms this. In this way, a rank-one approximation to a matrix can be thought of as decomposing the matrix into a pattern vector, and a presence vector that signifies the presence of the pattern.

Conventional singular value decompositions (SVDs) can be viewed as summations of rank-one approximations to a sequence of matrices. Starting with the given matrix, SVD computes a pair of singular vectors that are associated with the largest singular value of the matrix. The outer product of this pair, scaled by the corresponding singular value, provides the best rank-one approximation for the matrix in terms of minimizing the norm of the error. This approximation is subtracted from the given matrix to obtain a residual matrix, which in turn is the part of the matrix that cannot be represented by the first singular matrix, and the same procedure is applied to the residual matrix. Subsequent singular vectors are chosen to be orthogonal to all previous singular vectors. The number of singular vector pairs necessary to reach a zero residual matrix is equal to the rank of the matrix. Indeed, the procedure can be terminated earlier to obtain a "truncated SVD" for the matrix which provides the best possible approximation for a given number of singular vectors.

While SVD is useful in some applications involving discrete datasets (eg., latent semantic indexing (LSI)), the application of SVDs to binary matrices has two drawbacks. First, the resulting decomposition contains nonintegral vector values, which are generally hard to interpret for binary datasets. SDD partially solves this problem by restricting the entries of singular vectors to the set $\{-1, 0, 1\}$. However, the second drawback is associated with the idea of orthogonal decomposition, or more generally, extraction of singular vectors. If the underlying data consists of nonoverlapping (orthogonal) patterns only, SVD successfully identifies these patterns. However, if patterns with similar strengths overlap, then, because of the orthogonality constraint, the features contained in some of the previously discovered patterns are extracted from each pattern. In orthogonalizing the second singular vector with respect to

the first, SVD introduces negative values into the second vector. There is no easy interpretation of these negative values in the context of most postprocessing techniques, such as evaluating frequent itemsets. Since SDD is based on repeatedly finding rank-one approximations to a residual matrix obtained by extracting the information that is already contained in a previous approximation, SDD also suffers from the same problem. A simple solution to this problem is to cancel the effect of the first singular vector by removing this singular vector and introducing all subsets of this vector with appropriate weights. This can prove to be computationally expensive. What is required is a nonorthogonal transform that does not introduce negative values into the composing vectors.

Based on these observations, our modification to SDD for binary matrices has two major components:

—pattern and presence vectors are restricted to binary (0/1) elements, and
—the given matrix is partitioned based on the presence vector after each computation of a rank-one approximation, and the procedure is applied recursively to each partition. No orthogonalization is performed with respect to previously detected representative vectors. This method provides a hierarchical organization of representative patterns.

3.1 Discrete Rank-One Approximation of Binary Matrices

The problem of finding the optimal discrete rank-one approximation for a binary matrix can be stated as follows.

Definition 3.3. *Rank-One Approximation*
Given matrix $A \in \{0,1\}^m \times \{0,1\}^n$, find $x \in \{0,1\}^m$ and $y \in \{0,1\}^n$ that minimize the error:

$$h(A, \tilde{A}) = ||A - \tilde{A}||_F^2 = |\{i, j : A(i, j) \neq \tilde{A}(i, j)\}|, \tag{2}$$

where $\tilde{A} = x y^T$ is the representative matrix.

In other words, the error for a rank-one approximation is the number of nonzero entries in the residual matrix. This problem is closely related to one of finding maximal cliques in graphs. Although there is considerable literature on the maximum clique and biclique problems [Bron and Kerbosch 1973; Peeters 2003], we do not know of any approximation algorithms or effective heuristics in literature for this relaxed formulation of the problem. However, the main purpose here is to find a low-rank decomposition that approximates groups of rows with local patterns, rather than a globally optimal rank-one approximation. Since a locally optimal solution to the rank-one approximation problem is associated with a local pattern, it is adequate to apply an efficient heuristic to discover underlying local patterns in the matrix. Removing the nonorthogonality constraint and applying such an heuristic recursively, it is possible to find an approximation to the entire matrix while improving the local approximation as well. For this purpose, we adopt an alternating iterative heuristic for the computation of SDD vectors to binary matrices, with suitable initialization heuristics.

3.1.1 *Alternating Iterative Heuristics.* Since the objective (error) function can be written as

$$||A - x\,y^T||_F^2 = ||A||_F^2 - 2x^T Ay + ||x||_2^2||y||_2^2, \tag{3}$$

minimizing the error is equivalent to maximizing:

$$C_d(x, y) = 2x^T Ay - ||x||_2^2||y||_2^2. \tag{4}$$

For fixed y, it is possible to find vector x that maximizes this objective function in linear time in the number of nonzero entries of A. The solution to this problem is discussed in detail in Section 4.4.1. Similarly, the problem can be solved in linear time for vector y, given vector x. This leads to an alternating iterative algorithm based on the computation of SDD [Kolda and O'Leary 2000]; namely, initialize y, then solve for x; now, solve for y based on updated value of x. Repeat this process until there is no improvement in the objective function.

3.1.2 *Approximate Continuous Objective Function.* An alternate heuristic approach to the rank-one approximation problem is to replace the discrete objective function of Equation (4) with a continuous approximation, which follows from the analysis of the minimization problem of Equation (3) in the continuous domain. In the case of decomposing continuous valued matrices, it has been shown [O'Leary and Peleg 1983] that the objective function of rank-one approximation is equivalent to one of maximizing:

$$C_c(x, y) = \frac{(x^T Ay)^2}{||x||_2^2||y||_2^2}. \tag{5}$$

Although this function is not equivalent to the objective function in the case of binary matrices (*i.e.*, $C_d(x, y)$ and $C_c(x, y)$ do not have their global optimum at the same point) the behavior of these two functions is generally correlated. Thus, we can use $C_c(x, y)$ as a continuous approximation to $C_d(x, y)$. Although a local maximum of $C_c(x, y)$ does not necessarily correspond to a local maximum of the discrete objective function, it may correspond to a point that is close to a local maximum and has a higher objective value than many other undesirable local maxima. Consequently, it can be more effective and flexible in the iterative course of the algorithm, especially for very sparse matrices. In this case, fixing y and letting $s_y = Ay/||y||_2$ as above, the objective becomes one of maximizing $C_c(x) = \frac{(x^T s_y)^2}{||x||_2^2}$. This problem can also be solved in linear time, as discussed in Section 4.4.1. Both of the heuristics arising from the two objective functions are implemented in PROXIMUS.

3.2 Recursive Decomposition of Binary Matrices

We use the rank-one approximation of the given matrix to partition its rows into two submatrices. This is in contrast to conventional SVD-based techniques that compute the residual matrix and apply the transformation repeatedly. This difference manifests itself in the ability of PROXIMUS to find local (interpretable) patterns, whereas SVD finds global trends in the matrix.

Definition 3.4. *Partitioning Based on Rank-One Approximation:* Given rank-one approximation $\tilde{A} = xy^T \approx A$, a partition of A with respect to this approximation is defined by two submatrices A_1 and A_0, where

$$A(i) \in \begin{cases} A_1, \ if \ \ x(i) = 1 \\ A_0, \ otherwise \end{cases}$$

for $1 \leq i \leq m$. Here, $A(i)$ denotes the i^{th} row of A.

The intuition behind this approach is that the rows corresponding to 1's in the presence vector are the rows of a maximally correlated submatrix of A. Since the rank-one approximation for A gives no information about A_0, we further find a rank-one approximation and partition this matrix recursively. On the other hand, we use the representation of the rows in A_1 given by the pattern vector y and check if this representation is adequate via a stopping criterion. If so, we decide that matrix A_1 is adequately represented by matrix xy^T and stop; else, we discard x and y, and recursively apply the procedure for A_1 as for A_0.

The partitioning-and-approximation process continues until the matrix cannot be further partitioned and the resulting approximation adequately represents the entire matrix. We use the Hamming radius of the set of rows that are present in the approximation to measure the adequacy of the representation provided by a rank-one approximation, using the pattern vector as the centroid of this set of rows.

Definition 3.5. *Hamming Radius*
Given a matrix $A \in \{0, 1\}^{m \times n}$ and a binary vector $y \in \{0, 1\}^n$, the Hamming radius of A centered around y is defined as:

$$r(A, y) = \max_{1 \leq i \leq m_A} h(A(i)^T, y), \tag{6}$$

where $h(x, y) = ||x - y||_2^2$ is the Hamming distance between binary vectors x and y.

We use the Hamming radius as the major stopping criterion for the algorithm to decide whether the underlying pattern can represent all rows of the corresponding submatrix adequately. The recursive algorithm does not partition submatrix A_i further if the following conditions hold for the rank-one approximation $A_i \approx x_i y_i^T$.

—$r(A_{i1}, y_i) < \epsilon$, where ϵ is the prescribed bound on the Hamming radius of identified clusters; and
—$x_i(j) = 1 \ \forall j$ (i.e., all the rows of A_i are present in A_{i1}).

If the above conditions hold, the pattern vector y_i is identified as a representative pattern in matrix A_i and recorded along with its associated presence vector in the approximation of A. The resulting approximation for A is represented as $\tilde{A} = XY^T$. Here, X and Y are $m \times k$ and $n \times k$ matrices containing the presence and pattern vectors in their rows, respectively, and k is the number of identified patterns (i.e., the number of representative vector pairs). Note that

while the presence matrix X has orthogonal columns, since each pattern can be present in exactly one row, columns of the pattern vector Y are not necessarily orthogonal. This is because the resulting patterns may be overlapping, since the representative patterns are never extracted from the matrix. Thus, the resulting approximation is a binary *nonorthogonal* decomposition of A.

4. PROXIMUS: ALGORITHMS AND DATA STRUCTURES

PROXIMUS is a software package that uses the binary nonorthogonal decomposition described above for analysis of binary attributed datasets. It is available both as a stand-alone application for data analysis, or as a library of functions that can be used in diverse applications that involve binary datasets. Such applications include association rule mining, clustering, pattern discovery, and vector quantization.

4.1 Software Organization and Interface

PROXIMUS is implemented in the C programming language and its source code is available for free download at http://www.cs.purdue.edu/homes/koyuturk/proximus/. It can be compiled into an application that decomposes a given binary matrix into two low-rank binary matrices whose product approximates the given matrix based on various input parameters. These parameters can be tuned to obtain a desired approximation depending on the application. The software is composed of several modules that can serve independently as a library for computations on binary matrices.

4.1.1 *Input Parameters.* PROXIMUS can be executed as an application using the `bnd` command from the shell using the following prototype:

```
% bnd <filename> <optional arguments>.
```

Table II presents a description of the input parameters. The only mandatory argument for the application is the name of a file containing the given matrix. The file format used to store both given and representative matrices in PROXIMUS is described in Section 4.1.2. Table II also describes optional arguments that can be used to tune several parameters. The argument `algorithm` is used to specify the objective function to be maximized during rank-one approximation. Possible choices are discrete and continuous objective functions discussed in Section 3.1.1 and 3.1.2, respectively. The algorithm used for initializing the pattern vector in rank-one approximation is specified by the `init` argument. Various initialization methods implemented in PROXIMUS are discussed in Section 4.4.2. The argument `epsilon` sets the desired bound on the Hamming radius of the clusters of rows discovered by PROXIMUS. The final optional argument, `minclustersize`, can be used to limit the size of the clusters discovered by PROXIMUS. If this parameter is set to a value $c > 1$, PROXIMUS terminates with submatrices having number of rows less than c without checking the stopping criteria. The motivation for this parameter is that a submatrix that has c rows is small enough to be considered a good cluster of rows.

Table II. Description of Input Parameters

Argument	Description	Default value
`<filename>`	Name of the file containing input matrix.	N/A
`-a <algorithm>`	Algorithm to be used for rank-one approximation. (1) *Discrete* (2) *Continuous*	*Discrete*
`-i <init>`	Initialization strategy. (1) *Allones* (2) *Center* (3) *Maximum* (4) *Partition* (5) *Greedy Graph Growing* (6) *Neighbor* (7) *Random-row* (8) *Random*	*Random-row*
`-e <epsilon>`	Bound on Hamming radius.	0
`-w`	Write representative vectors. If used, presence and pattern vectors will be written in files `<filename>.X.out` and `<filename>.Y.out`, respectively.	*False*
`-c <minclustersize>`	Minimum size of a cluster of rows.	1

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

(a)

$$\begin{matrix} 4 & 5 & 8 \\ 1 & 4 & \\ 0 & 3 & 4 \\ 0 & 3 & \\ 2 & & \end{matrix}$$

(b)

Fig. 1. Storage of sparse binary matrices in PROXIMUS: (a) sample matrix, (b) its representation in row-major format.

4.1.2 *Data Representation and Storage.* In PROXIMUS, binary matrices are stored in sparse row-major format as `ascii` files. An $m \times n$ matrix containing N nonzero entries is stored in a file containing $m+1$ lines. Each of the last m lines contain a list of column indices of nonzero entries (ones) in the corresponding row of the matrix. Indexing of columns starts from zero in PROXIMUS. The first line is reserved for the declaration of matrix parameters—it contains the number of rows, columns, and nonzeros in the matrix, respectively. A sample sparse binary matrix and its representation are shown in Figure 1.

4.2 Overview of Algorithms

We illustrate the recursive structure of the hierarchical decomposition of binary matrices with an example.

Example 4.1. Figure 2 illustrates the recursive structure of PROXIMUS in the decomposition of the matrix (A) of Figure 1(a) with bound on Hamming radius $\epsilon = 1$. Starting with matrix A, a rank-one approximation to A is computed. Matrix A is then partitioned into A_1 and A_0 based on the presence vector $x = [0\ 1\ 1\ 0]^T$. Here, the pattern vector is $y = [1\ 0\ 0\ 1\ 1]^T$. The rank-one approximation to A_1 returns a presence vector of all 1's ($x_1 = [1\ 1]^T$) and the

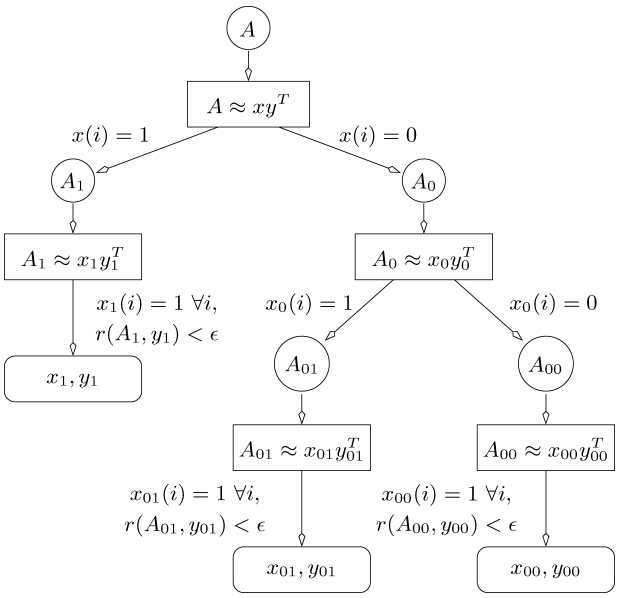


Fig. 2. Recursive structure of PROXIMUS. The tree shows the recursive decomposition of the matrix in Figure 1(a) with $\epsilon = 1$. Each rectangular internal node is a rank-one approximation and two circular children of these nodes are the matrices that result from partitioning of parent matrix based on this approximation. Leaves of the recursion tree correspond to final decomposition.

approximation is adequate ($r(A_1, x_1) \leq 1$), so the recursion stops at that node and $y_1 = [1\ 0\ 0\ 1\ 1]^T$ is recorded as a representative pattern. Note that while recording presence vectors, we extend the dimensions of the vector to the number of rows in the vector by filling in zeros into all entries that correspond to the rows of the given matrix that are not contained in the present submatrix (i.e., x_1 is recorded as $[0\ 1\ 1\ 0]^T$ since A_1 contains only the second and third rows of A). Matrix A_0 is further partitioned since the approximation $\tilde{A}_0 = x_0 y_0^T$, where $y_0 = [0\ 1\ 0\ 0\ 1]^T$ and $x_0 = [1\ 0]^T$, does not cover all rows of A_0. The overall decomposition of A into three representative vector pairs is $\tilde{A} = XY^T$, where

$$X = [x_1, x_{01}, x_{00}] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and } Y^T = [y_1, y_{01}, y_{00}]^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

A more detailed example for recursive decomposition on a larger matrix is shown in Figure 8.

An outline of the recursive algorithm and its components is presented in Figure 3. Procedure INITIALIZEPATTERNVECTOR initializes the pattern vector based on the given matrix. Possible methods for initialization of the pattern vector are discussed in Section 4.4.2.

```
procedure RankOneApproximation(Matrix A: matrix to be approximated)
    ▷returns Binary Vector x, y : approximation vectors
    ▷Integer Vector z_x, z_y: vectors to record x^T A, Ay
    ▷Integer n_x, n_y: number of 1's in vectors x, y
    y ← InitializePatternVector(A)
    repeat
        y' ← y
        z_y ← Ay, n_y ← ||y||_2^2
        solve for x to maximize 2x^T z_y − n_y||x||_2^2
        z_x ← x^T A, n_x ← ||x||_2^2
        solve for y to maximize 2z_x y − n_x||y||_2^2
    until y' = y
end

procedure Partition(Matrix A: matrix to be partitioned, Binary Vector x: presence vector)
    ▷returns Matrix A_0, A_1: submatrices of A characterized by presence vector x
    Initialize A_0 and A_1 to empty matrices
    for each row A(i) of A do
        if x(i) = 1
          then put A(i) in A_1
          else put A(i) in A_0
    endfor
end

procedure Decompose (Matrix A: matrix to be decomposed)
    ▷Binary Vector x, y : representative vectors
    ▷Matrix A_0, A_1 : submatrices of A
    {x, y} ← RankOneApproximation(A)
    {A_0, A_1} ← Partition(A, x)
    if A_1 meets stopping criteria
      then record x and y
      else Decompose(A_1)
    if A_0 is not empty
      then Decompose(A_0)
end
```

Fig. 3. Outline of the algorithms for recursive decomposition of binary matrices.

4.3 Data Structures

As seen in Figure 3, three major types of data are involved in the decomposition of binary matrices. The first component corresponds to the binary matrices themselves, which act as the input and output for the decomposition. Each rank-one approximation is associated with two binary vectors that need to be saved throughout the computation since they constitute the final approximation and help in tracking the hierarchical structure of the resulting decomposition. Finally, integer vectors are used in the computation as the result of multiplication of binary matrices with binary vectors. Binary matrices and vectors are sparse by nature and the selection of appropriate storage schemes for these data types is important for space utilization as well as computational cost, since they generally tend to be very large and high-dimensional. The only integer vectors that appear during the course of the algorithm are z_x and z_y in procedure RankOneApproximation, which is shown in Figure 3. Since these vectors result from a binary matrix-vector multiplication, they are not necessarily

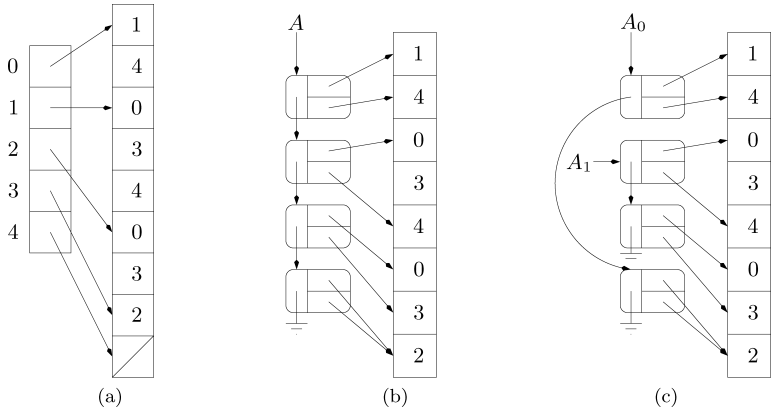


Fig. 4. Sparse representation of the matrix in Figure 1 in (a) traditional CSR; (b) modified CSR format; (c) original matrix partitioned into two.

sparse. Therefore, they are stored in a dense array of integers. However, since these integer vectors are temporary variables that are consumed immediately after being produced, it is possible to recycle these vectors within and across rank-one approximations.

4.3.1 *Binary Matrices.* Several binary matrices appear through the recursive process of decomposition. The rows of each matrix are partitioned into two submatrices based on rank-one approximation and each resulting matrix is further decomposed recursively. It would require excessive space and data movement if each matrix that appears in the course of the recursive process is stored separately. This process can be handled very efficiently based on the following observations: First, the two children of a matrix that result from partitioning the matrix are disjoint (row) subsets of the original matrix. Second, it is easy to split the rows of a matrix into two if the matrix is stored in compressed sparse row (CSR) format. Therefore, entire decomposition can be performed in-place on the original matrix with an appropriate modification of CSR format.

Traditionally, CSR representation of sparse matrices consists of two arrays; one storing the column indices of nonzero entries and the other storing pointers to this array, one for each row, to point the first nonzero entry of the row [Saad 1996]. End-pointers are not needed since the start pointer of each row also marks the end of the previous row. Figure 4(a) illustrates the CSR representation of the matrix in Figure 1(a). The last element of the pointer array is necessary to mark the end of the list of nonzero elements in the matrix. Note that these arrays are usually accompanied by a third array that stores the value of each entry. This is not necessary in our case since each nonzero entry of the matrix is equal to one.

In PROXIMUS, row pointers are stored in a linked list rather than in a contiguous array, in order to handle the partitioning of rows efficiently. Figure 4(b) shows this representation. The matrix can be partitioned by simply splitting the linked list of row pointers. This operation can be performed in a single pass over the rows without any extra storage. The partitioning of matrix A of

Example 4.1 into A_1 and A_0 is shown in Figure 4(c). Note that it is now necessary to keep pointers to not only the first entry but also to the last entry of each row, since the list of nonzero entries of the matrix is not contiguous anymore. With this representation, the space required to store all binary matrices that appear during the course of decomposition is no more than the space required for the matrix.

4.3.2 *Binary Vectors.* It is possible to store binary vectors in either compressed sparse format or in bit-arrays. While compressed sparse format is highly efficient in terms of space, it is necessary to use a full-vector to perform sparse matrix-vector multiplication (mat-vec) in time linear in the number of nonzero entries [Duff et al. 1987]. Since the core computation in recursive matrix decomposition is a mat-vec, it is crucial for overall performance to perform this operation in linear time. For this reason, while the binary vectors are stored in sparse representation throughout the decomposition process, they are expanded into full vectors (bit-arrays) during mat-vecs. Note that bit-arrays are also recycled in the same manner as integer vectors, requiring storage of only two full vectors for the entire decomposition process.

4.4 Algorithmic Issues

4.4.1 *Major Computations.* As seen in Figure 3, the core computations in the decomposition of matrices are those during each iteration of a rank-one approximation, namely, a matrix-vector multiplication followed by solving the maximization problem of Equation (4). Right matrix-vector multiplication (*i.e.*, the computation of $z_y = Ay$) can be easily performed in time linear in the number of nonzeros of A. Similarly, left-vector multiplication ($z_x = x^T A$) can also be performed in linear time by visiting each element of the binary vector and if this element is nonzero, incrementing the corresponding entry of z_x for each nonzero element of the corresponding row [Loan 2000].

If the objective function to be maximized is the discrete function of Equation (4), then the solution for vector x for a fixed vector y is given by the following equation:

$$x(i) = \begin{cases} 1, & \textit{if } 2z_y(i) \geq n_y \\ 0, & \textit{otherwise} \end{cases} \tag{7}$$

This equation follows from the following idea: If we set the i^{th} entry of vector x, then the i^{th} row of the representative matrix xy^T will have $n_y = ||y||_2^2$ nonzero elements. Then, the contribution of this entry to $C_d(x)$ will be positive only if at least half of these elements match those of the original matrix. Note that two entries match if the j^{th} entry of the row of interest of A and the binary vector y are both nonzero. Clearly, $z_y(i)$ is equal to the number of nonzeros on the i^{th} row of A that match the nonzeros in y. Therefore, Equation (7) follows. This equation can be evaluated in $\Theta(m)$ time, once z_y is computed. Similarly, we can compute vector y that maximizes $C_d(x, y)$ for a fixed x in $\Theta(n)$ time.

In the case of the continuous objective function, the algorithm is based on the following observation: If the solution to $C_c(x)$ contains l nonzero entries, these entries correspond to the largest l entries of s_y. This is indeed a generalization of

```
procedure MAXIMIZEXS(Integer Vector s_y: Ay for fixed y)
    ▷returns Binary Vector x: that maximizes x^T s_y/||x||_2^2
    Find Π = {π_1,π_2,...,π_m} s.t. s_y(π_i) ≥ s_y(π_j) if i > j
    Initialize x(i) ← 0 for 1 ≤ i ≤ m
    C_c(x) ← 0 , sum ← 0
    for i ← 1 to m do
        sum ← sum + s_y(π_i)
        if sum^2/(||x||_2^2 + 1) ≥ C_c(x)
          then x(π_i) ← 1
               C_c(x) ← sum^2/||x||_2^2
          else stop
    endfor
end
```

Fig. 5. Computation of vector x to maximize $C_c(x)$ for fixed vector y.

the statement of the previous paragraph and was proved by O'Leary and Peleg [1983]. Thus, the problem can be solved in linear time by sorting elements of s_y and visiting elements of x in the resulting order until no improvement in the objective function is possible. Note that although entries of s_y are normalized in the mathematical formulation, it can be omitted in implementation since normalization doesn't affect the ordering of the elements. In addition, s_y being an integer vector makes it possible to use counting sort, which proves to be efficient for this problem since the entries of s_y are bounded by number of columns (rows) of A. The algorithm for the maximization of $C_c(x)$ for fixed vector y is given in Figure 5.

4.4.2 *Initialization of Iterative Process.* While finding a rank-one approximation, initialization is critical not only for fast convergence but also for the quality of the solution since a poor choice can result in poor local maxima. In order to have a feasible solution, the initial pattern vector should have magnitude greater than zero (*i.e.*, at least one of the entries in the initial pattern vector should be equal to one). It is important that the initialization of pattern vector must not require more than $\Theta(N_{A_i})$ operations, where A_i is the submatrix being approximated, since it will otherwise dominate the runtime of the overall algorithm. The following initialization algorithms are implemented in PROXIMUS.

—*Allones:* A vector of all ones. This scheme is very fast but generally converges to local maxima that correspond to large clusters with poor proximity.

—*Center:* Center of all rows on the n-dimensional hypercube. This scheme leads to patterns that characterize the overall structure of the matrix rather than capturing individual patterns.

—*Maximum:* Set only the entry of pattern vector that corresponds to the column of the matrix with largest number of nonzeros.

—*Partition:* Select a separator column and identify the rows that have a nonzero in that column. Initialize the pattern vector to the centroid of these rows. The idea is to partition the rows of the matrix along one dimension expecting that such a partition will include rows that contain a particular pattern.

—*Greedy Graph Growing* (GGG): Based on the idea of iterative improvement heuristics in graph partitioning [Karypis and Kumar 1998], this scheme starts with a randomly selected row in one part and grows that part by including rows that share the maximum number of nonzeros with that part until a balanced partition is obtained. The initial pattern vector is set to the center of rows in this part.

—*Neighbor:* Observing that a balanced partition of rows is not necessary due to the nature of the problem, we select a row randomly and initialize the pattern vector to the centroid of the neighbors of that row (*i.e.*, the set of rows that share a nonzero with that particular row).

—*Random-Row:* Initial pattern vector is a randomly selected row. The expectation is that the selected row is part of a cluster, so the iterative algorithm will capture the representative pattern in that cluster.

—*Random:* Some randomly selected entries of initial pattern vector are set. In PROXIMUS, the number of entries set is equal to the average number of nonzeros per row.

All of the above initialization schemes require $O(N_{A_i})$ time to compute, where N_{A_i} is the number of nonzero entries in the matrix being approximated, therefore, they do not introduce any overhead on the asymptotic complexity of the underlying algorithm.

4.5 Time and Space Complexity

4.5.1 *Time Complexity.* In the alternating iterative heuristic for computing rank-one approximations, each solution to the optimization problem of Equation (4) takes $O(N_{A_i})$ time, where A_i is the submatrix being approximated. The number of iterations required to compute a rank-one approximation is a function of the initialization vector and strength of associated local minima. In general, if the underlying pattern is strong, we observe very fast convergence. In our experiments, we observe the computation time of a rank-one approximation to be linear in the number of nonzeros of the matrix for all instances.

If we view the recursive process as a tree with each node being a rank-one approximation to a matrix, we can observe that the total number of nonzeros of the matrices at each level of the recursion tree is at most equal to the number of nonzeros in the original matrix. Thus, the overall time complexity of the algorithm is bounded by $O(d \times N)$, where d denotes the height of the recursion tree. If the resulting decomposition has k pattern vectors (which is equal to the number of leaves) in the recursion tree, then $d \leq k - 1$. Note that this is a loose bound, since $d \ll k$ in general. More specifically, d and k are functions of the underlying pattern structure of the given matrix and the prescribed bound on Hamming radius.

4.5.2 *Space Complexity.* As discussed in Section 4.3.1, all rank-one approximations are performed in-place on the original matrix, therefore, the total space required for the binary matrices is $O(m + N)$. All binary and integer vectors that appear in the computation of a rank-one approximation are recycled across rank-one approximations. A vector can be of length at most $O(m + n)$,

therefore, the space required for storing vectors is $O(m + n)$. If we are only interested in the final decomposition, all the pattern and presence vectors that appear at the leaves of the recursion tree are recorded in pattern and presence matrices, respectively. The final presence matrix is a sparse binary matrix of size $k \times m$, with m nonzero entries since each row is linked to exactly one pattern. The pattern matrix contains the representative patterns in its k columns, and a pattern vector can contain at most twice the number of nonzeros in the row of the original matrix that contains this pattern, by Equation (7). Thus, there can be at most $2N$ nonzeros in the pattern matrix. Note that this is a loose bound since the number of pattern vectors are much smaller than the number of rows in the given matrix. Since $k \leq m$ and we may assume that $m, n \leq N$, the overall space requirement is linear in the number of nonzeros of the matrix. We conclude from this discussion that PROXIMUS is well-suited for use in large-scale problems.

If we need to keep track of the hierarchical structure of the decomposition, then it is only necessary to store all pattern vectors that appear at intermediate nodes of the recursion tree, since intermediate presence vectors can be reconstructed from the tree. Since the recursion tree has at most $2k$ nodes and each pattern vector contains approximately $2N/m$ nonzeros, the additional space required for saving hierarchical cluster information is proportional to Nk/m, on an average.

5. EXPERIMENTAL RESULTS

In this section we present detailed experimental results on various performance aspects of PROXIMUS. We first compare the performance of PROXIMUS with that of traditional matrix decomposition and clustering methods. We then present results on synthetic datasets with a view to exploring various program parameters and their impact on performance. The IBM Quest dataset generator is particularly useful in this regard, since it enables us to generate datasets with specific characteristics. We also present two real applications built on the PROXIMUS library. These applications relate to mining association rules in relational data and extracting coregulated genes from microarray data.

5.1 Use of PROXIMUS in Clustering and Pattern Extraction

In this section, we report two experiments that illustrate the superior characteristics of PROXIMUS in approximating and clustering binary datasets compared to other state-of-the-art clustering and approximation techniques that work particularly well on continuous data. We generate two sample matrices by implanting uniform patterns into groups of rows on a background of uniform white noise.

The first matrix, shown in Figure 6(a), contains four overlapping patterns with uniform distribution. This matrix is generated as follows. For the background noise, any entry of the 80×52 matrix is set to 1 with probability p_b. If the ith row contains the kth pattern, then the (i, j)th entry of the matrix is set to 1 with probability p_p, where $(k - 1)l + 1 \leq j \leq kl + r$. Here, the leading columns of two successive patterns are l apart and r denotes the number of

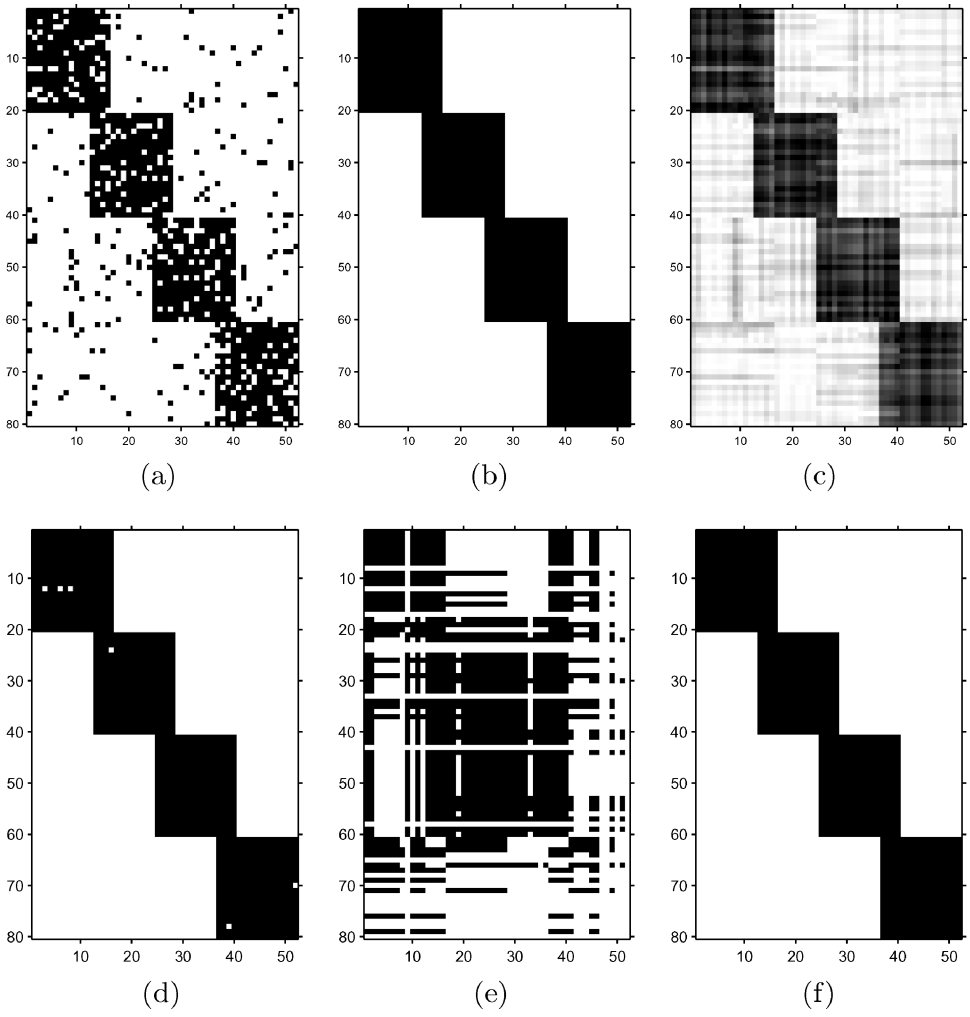


Fig. 6. Approximation of a sample binary matrix that contains four overlapping uniform patterns. (a) original matrix, (b) rank-4 approximation provided by PROXIMUS, (c) rank-4 approximation provided by SVD, (d) rank-8 approximation obtained by quantizing SVD approximation, (e) approximation (sum of four rank-one matrices) obtained by quantizing most dominant singular vectors, (f) rank-4 approximation provided by K-means clustering.

columns shared by two neighboring patterns. While generating the matrix of Figure 6, pattern length parameters l and r are set to 12 and 4, respectively, probability parameters p_b and p_p are set to 0.01 and 0.8, respectively, and the number of rows that contain the same pattern is set to 20. Note that the rows and columns that belong to a particular pattern are shown to be adjacent in the figures just for illustration purposes. In other words, for any of the algorithms whose performance is reported here, the ordering of rows or columns is not important. Indeed, if we reorder the rows and the columns of the matrix randomly, it is possible to recover the block-diagonal structure of the matrix using the hierarchical clustering of rows provided by PROXIMUS.

The rank-4 approximation provided by binary nonorthogonal decomposition of the matrix is shown in Figure 6(b). As seen in the figure, PROXIMUS is able to capture the four underlying patterns in the matrix and associate each row with the pattern that it contains. The Frobenius norm of the error of this approximation is 19.7, which is the square root of the Hamming distance of 388 between the given and representative matrices.

The rank-4 approximation provided by the four most significant singular vectors of SVD is shown in Figure 6(c). This approximation is optimal in the sense of minimum least squares, with an error of 17.2. Although this is less than the binary approximation provided by PROXIMUS, it is not very useful in applications involving binary data for several reasons, as discussed earlier. Although we can see in the figure that SVD approximation is able to reveal the underlying patterns on the diagonal blocks of the matrix once the matrix is reordered, it is not possible to capture these patterns just by analyzing the real-valued singular vectors provided by SVD. On the other hand, binary pattern and presence vectors of PROXIMUS reveal this structure clearly, regardless of ordering. In order to address the interpretability problem of SVD, it is necessary to quantize the SVD approximation. This can be done in two ways. The first method is to quantize the rank-4 SVD approximation matrix, obtaining the binary approximation of Figure 6(d) with an error of 19.7, which is the same as that of PROXIMUS. However, the rank of this approximation is 8, since quantization of individual entries does not preserve the rank of the matrix. In order to preserve the rank of the matrix, it is possible to quantize the dominant singular vectors rather than the representative matrix. This makes it possible to represent the representative matrix as the sum of four rank-one matrices. However, quantization of singular vectors is problematic since these vectors may contain large negative values. The only way to quantize these vectors is rounding the absolute value of each singular vector amplified by the associated singular value, relying on the assumption that a large negative value in a left singular vector accompanied by another negative in the corresponding right singular vector may be associated with a pattern in the matrix. However, this assumption does not always hold, since a negative value combined with a positive value in the corresponding singular vector may be associated with the correction of an error introduced by more dominant singular vectors. Consequently, binary quantization amplifies such errors because of misinterpretation of negative values. Indeed, the rank-4 approximation obtained by quantizing singular vectors has an error of 45.2 that is more than 100% worse than that of other techniques. As seen in Figure 6(e), this method is unable to reveal the underlying pattern structure.

We also compare the performance of PROXIMUS with that of K-means. We obtain an approximation through K-means clustering by approximating each row by the centroid of the cluster that it is assigned to. For the matrix of Figure 6, four-way K-means clustering provides the same approximation as PROXIMUS, as shown in Figure 6(f). However, for harder instances K-means is not able to separate clusters with significant overlap, as will be discussed in the next example.

The approximation provided by the methods of interest on a harder instance is shown in Figure 7. The 134×64 matrix shown in Figure 7(a) consists of

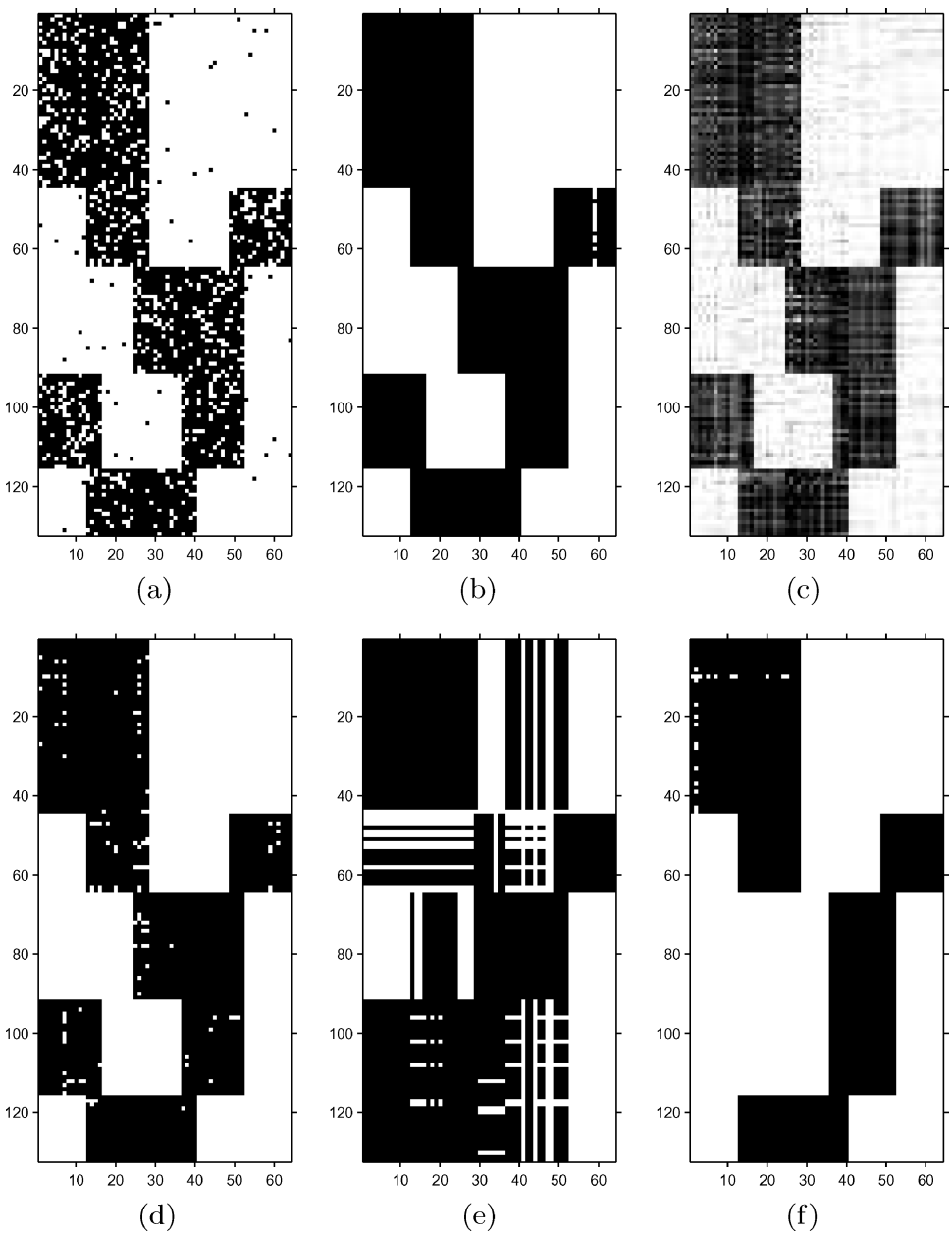


Fig. 7. Approximation of a sample binary matrix that contains five row clusters, each containing a randomly chosen pair of five overlapping uniform patterns. (a) original matrix, (b) rank-6 approximation provided by PROXIMUS, (c) rank-6 approximation provided by SVD, (d) rank-29 approximation obtained by quantizing SVD approximation, (e) approximation (sum of 6 rank-one matrices) obtained by quantizing most dominant singular vectors, (f) rank-6 approximation provided by K-means clustering.

five groups of rows, each of which contain two patterns randomly drawn from five uniform overlapping patterns. These patterns are generated as described above with the same pattern length parameters ($l = 12$, $r = 4$) and density parameters $p_b = 0.005$ for background noise, and $p_p = 0.8$ for patterns. In this experiment, the number of rows in each group is also chosen randomly from a normal distribution.

As evident in Figure 7(b), PROXIMUS is able to provide a rank-6 approximation for this matrix, which reveals the underlying pattern structure well, with an error of 27.3. The only redundancy in this approximation is the division of the second row group into two parts, which adds an additional rank for the approximation. This is caused by the outlying sparsity of some columns in the fifth pattern. On the other hand, as seen in Figures 7(c) and (d), although SVD provides a rank-6 approximation with an error of 22.9 and the error of the quantized SVD approximation is 26.2, which is better than that of PROXIMUS, this approximation is of rank 29! If we rather quantize the SVD approximation at the singular vector-level as a sum of six rank-one matrices, the approximation totally deviates from the original matrix with an error of 68.7, which is shown in Figure 7(e).

The approximation provided by six-way K-means clustering is shown in Figure 7(f). The error of this approximation is 34.1. Although this approximation is able to capture the patterns in the first, second, and fifth row groups, it clusters the significantly overlapping third and fourth row groups together. If we try five-way clustering taking into account that there are five implanted row groups, K-means is still not able to distinguish these two row groups as separate clusters.

The recursion tree for the decomposition of the matrix in Figure 7 is shown in Figure 8. This tree provides an illustrative example of the hierarchical nature of PROXIMUS. The figure also illustrates the suboptimality of the heuristic in face of the NP-hardness of the underlying problem. Observe that the submatrices that are at the right-most two leaves of the tree contain very similar patterns. However, since they are separated because of the dominance of another pattern in the course of decomposition, they are represented by different pattern vectors. In general, this problem can be stated as follows. Once rows are partitioned into two different submatrices based on one pattern vector, they cannot subsequently come together to be represented by the same pattern vector. While this problem might be addressed to a certain extent by rollbacks in recursion, this would require significant compromise in terms of efficiency and simplicity of the algorithm. In the implementation of PROXIMUS, we attempt to minimize the effect of this problem at the leaves of the recursion tree. When an approximation meets the stopping criteria, the pattern vector is compared to previously recorded pattern vectors. If a close approximation exists, all of the corresponding rows are candidates for merging into one representative vector.

5.2 Quantitative Evaluation of PROXIMUS

5.2.1 *Performance Evaluation Metrics.* Nonorthogonal decomposition of a matrix A provides an approximation to the matrix, namely, $\tilde{A} = XY^T$, where

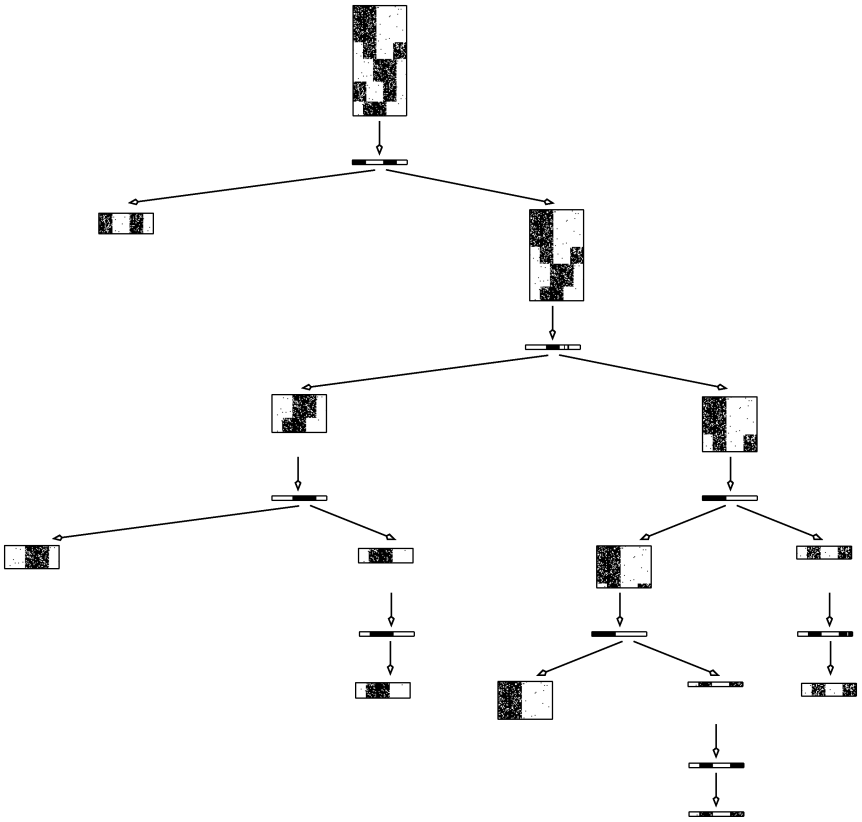


Fig. 8. Decomposition of the matrix in Figure 7 via PROXIMUS. A rank-one approximation is computed for each matrix (A_i) at the internal nodes of the tree. Resulting pattern vector (y_i) is shown at the single child node of each matrix. The left child of a pattern vector is the submatrix that contains the pattern (A_{i1}) and the right child is the submatrix that is composed of remaining rows of the parent matrix (A_{i0}). The representative vector for the set of rows at each leaf of the PROXIMUS tree is the one at its parent node.

X and Y are the presence and pattern matrices, respectively. A metric that immediately follows from the definition of the problem is the Hamming distance between the given and representative matrices, denoted $h(A, \tilde{A})$. Since this metric is highly dependent on the size of the given matrix, we rather use the average Hamming error per row, given by $h(A, \tilde{A})/m_A$. This metric provides an understanding for the average deviation of all rows from their representative pattern vectors.

We use two other normalized metrics, namely, *precision* and *recall*, which are used frequently in the data mining community for the evaluation of mining, clustering, classification, and learning algorithms. These two metrics together provide a two-sided understanding of how well the information in the given matrix is captured and what extra information is introduced by the approximation. Precision measures the fraction of ones in the representative matrix that also exist in the original matrix. It is defined by the following

Table III. Description of Generated Data in Terms of Number of Rows, Columns, Patterns, and Nonzeros

Dataset	# Rows	# Cols	# Patterns	# Nonzeros
M10K	7513	472	100	95048
L100K	76025	178	20	965210
M100K	75070	852	100	955555
H100K	74696	3185	500	958733
M1M	751357	922	100	9557237

formula:

$$precision = \frac{||A\&\tilde{A}||_F^2}{||\tilde{A}||_F^2} = \frac{|\{i, j : A(i, j) = \tilde{A}(i, j) = 1\}|}{|\{i, j : \tilde{A}(i, j) = 1\}|} \tag{8}$$

Recall, on the other hand, measures the fraction of the ones in the original matrix that are also captured by the decomposition. It is defined by the following formula:

$$recall = \frac{||A\&\tilde{A}||_F^2}{||A||_F^2} = \frac{|\{i, j : A(i, j) = \tilde{A}(i, j) = 1\}|}{|\{i, j : A(i, j) = 1\}|} \tag{9}$$

To measure the compactness of the approximation provided by PROXIMUS, we adapt the commonly used *compression ratio* metric [Cover and Thomas 1991] to our problem. Considering the number of nonzeros as the matrix size, we define compression ratio as follows:

$$compression = \frac{N_X + N_Y}{N_A} = \frac{|\{i, j : X(i, j) = 1\}| + |\{i, j : Y(i, j) = 1\}|}{|\{i, j : A(i, j) = 1\}|}. \tag{10}$$

Note that one may also consider the number of rows as the matrix size in some applications.

5.2.2 *Description of Input Matrices.* The input matrices for this specific study are generated using the IBM Quest synthetic data generator, which allows us to control the number of patterns in the data as well as the correlations between them [IBM]. We generate two sets of data, one for varying the number of rows and the other for varying the number of patterns. In the first set, the number of patterns is fixed at 100 (medium), and three instances, named M10K, M100K, and M1M, containing $\approx 10K$ (low), $\approx 100K$ (medium), and $\approx 1M$ (high) rows, respectively, are generated. In the second set, the number of rows is fixed at $\approx 100K$ (medium) and three instances, named L100K, M100K, and H100K, containing 20 (low), 100 (medium), and 500 (high) patterns, respectively, are generated. The average number of nonzeros per row is set to 10. The average correlation between each pair of patterns is set to 0.1, while the average confidence of a pattern is set to 90%. Table III presents a general description of the five instances in terms of number of rows, columns, patterns, and nonzeros.

5.2.3 *Performance of* PROXIMUS *on Different Instances.* A summary of the results obtained from the decomposition of five synthetic matrices is shown in Table IV. We show results of decomposing the five matrices with both

Table IV. Performance on PROXIMUS on Five Instances (in terms of runtime, number of representative vectors, compression ratio, error per-row, precision, and recall)

Obj. Func.	Init. Scheme	Runtime (secs.)	# Appx. Vectors	Comp. Ratio	Error Per-row	% Prec.	% Recall
			M10K				
Disc.	Partition	0.3	1307	0.33	0.18	99.2	99.4
Cont.	Partition	0.2	1309	0.33	0.18	99.2	99.3
Disc.	Randomrow	0.3	1306	0.33	0.18	99.2	99.4
Cont.	Randomrow	0.2	1319	0.33	0.17	99.2	99.5
			L100K				
Disc.	Partition	1.5	771	0.10	0.38	98.7	98.4
Cont.	Partition	1.6	772	0.09	0.37	98.7	98.4
Disc.	Randomrow	1.0	737	0.09	0.31	98.6	98.9
Cont.	Randomrow	1.2	783	0.10	0.39	98.6	98.3
			M100K				
Disc.	Partition	5.0	4208	0.17	0.27	99.0	98.8
Cont.	Partition	4.3	4272	0.17	0.30	98.9	98.7
Disc.	Randomrow	3.4	4256	0.17	0.25	99.0	99.1
Cont.	Randomrow	3.3	4155	0.16	0.29	99.0	98.7
			H100K				
Disc.	Partition	33.6	16717	0.40	0.14	99.2	99.7
Cont.	Partition	28.0	16799	0.40	0.14	99.2	99.7
Disc.	Randomrow	26.8	16828	0.40	0.14	99.2	99.8
Cont.	Randomrow	26.7	16737	0.40	0.14	99.2	99.7
			M1M				
Disc.	Partition	87.8	12218	0.10	0.37	99.0	98.1
Cont.	Partition	95.0	12375	0.10	0.40	99.0	97.8
Disc.	Randomrow	52.0	11958	0.10	0.33	99.0	98.5
Cont.	Randomrow	52.3	12014	0.10	0.37	99.0	98.1

discrete and continuous objective functions for the rank-one approximation heuristic using two initialization schemes, *partition* and *random-row*. In all experiments, the bound on Hamming radius is set to 3. All experiments reported in this section are performed on a Pentium-IV 3.0 GHz server with 512 MB RAM.

The columns in Table IV show the time spent in decomposing the matrix in seconds, compression ratio, number of representative vectors (k), percentage accuracy, precision, and recall, respectively. As evident in the table, PROXIMUS is able to provide compression up to a factor of ten (*i.e.*, compression ratio of 0.10) while maintaining above 98% precision and recall for the hardest instance. More impressively, the average Hamming error per row is less than 0.4 for all instances, although the bound on Hamming radius is set to 3.

A comparison of the results on L100K, M100K, and H100K matrices reveals that the more the number of underlying patterns in the matrix, the more the number of representative vectors required for decomposition. In other words, it is possible to provide more compression in less time for instances that have fewer underlying patterns. Therefore, like any other compression scheme or matrix decomposition, PROXIMUS is also more successful and efficient on instances with less entropy. Note also that the approximation quality is not significantly affected by the number of underlying patterns once ϵ is fixed. Therefore, it is

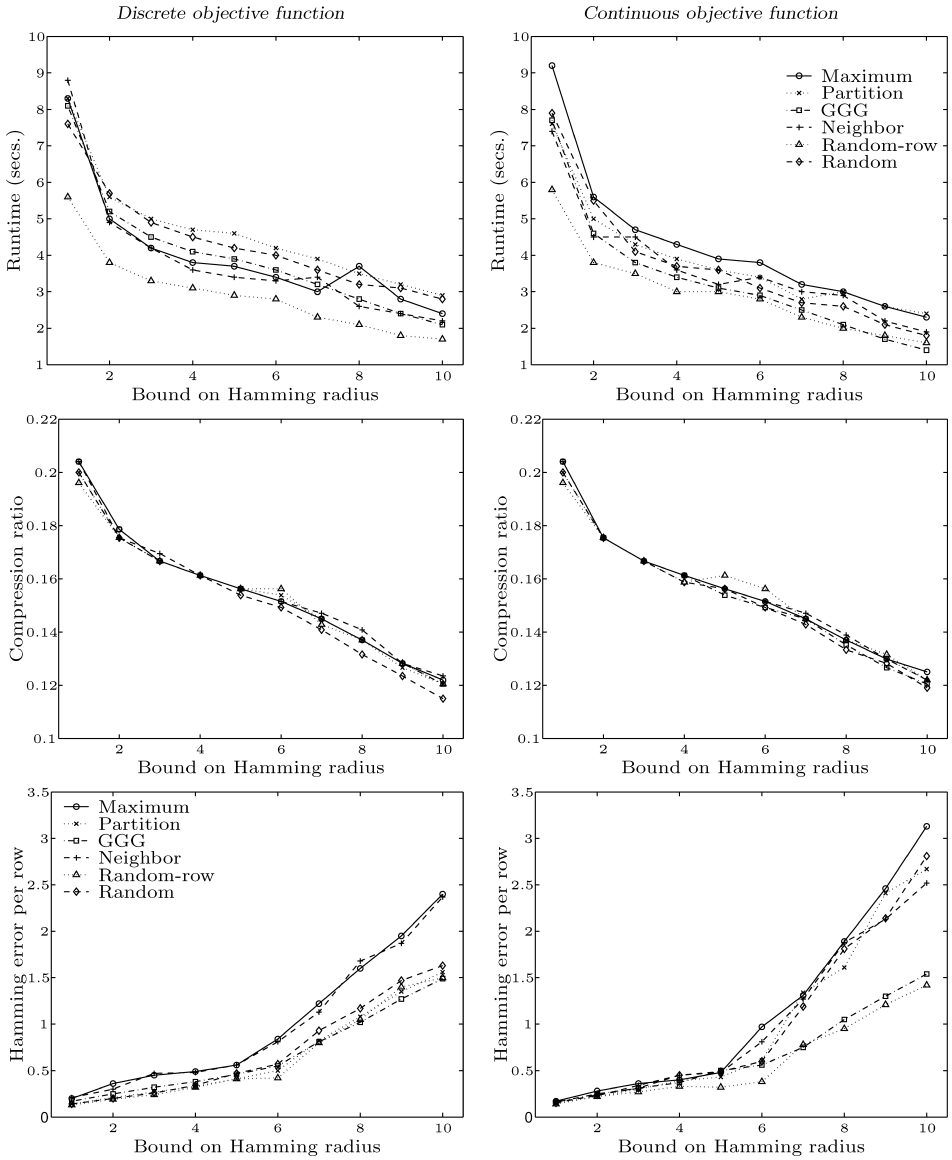


Fig. 9. Effects of bound on Hamming radius (ϵ), initialization scheme and the heuristic objective function on the performance of PROXIMUS on the M100K matrix. The graphs on first and second rows show the variance in runtime, compression ratio, and Hamming error per-row, respectively, for ϵ ranging from 1 to 10, for each initialization scheme. The results for the discrete and continuous objective functions are shown on the left and right panels, respectively.

possible to achieve better compression by trading off quality. More strikingly, our results on the M100K dataset show that the degradation in quality for better compression is not very significant, namely, it is possible to achieve a compression ratio of 0.12 for this matrix while keeping the precision and recall above 90%, as shown in Figures 9 and 10.

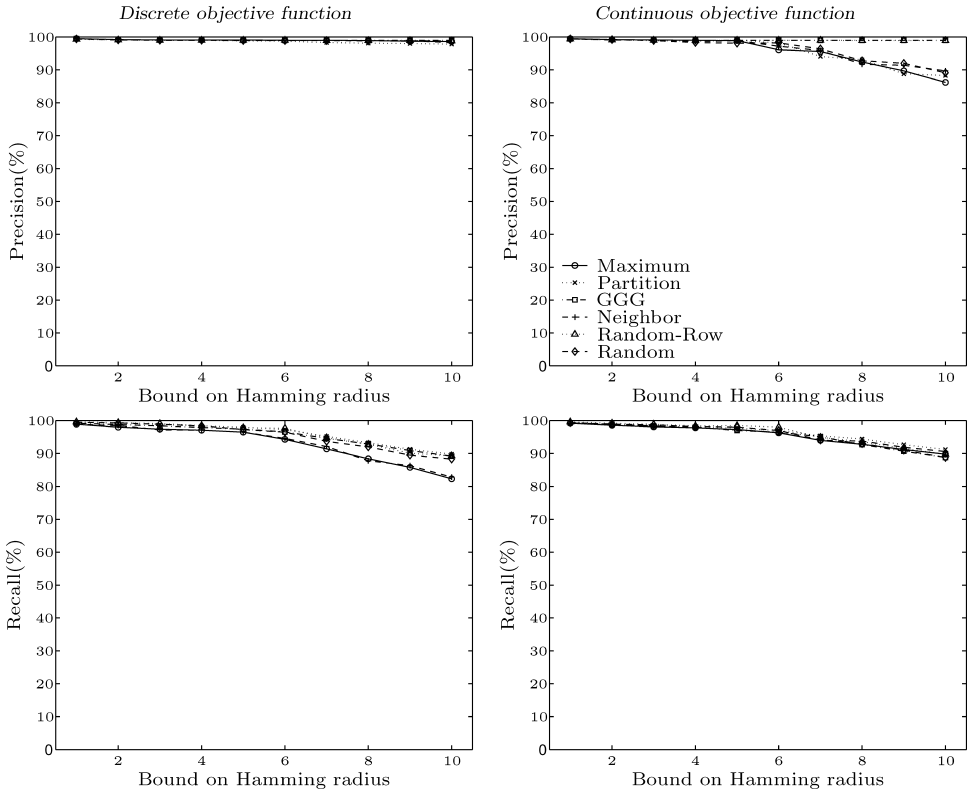


Fig. 10. Effect of bound on Hamming radius (ϵ), initialization scheme, and the heuristic objective function on the precision and recall of approximation provided by PROXIMUS. The graphs in first and second rows show the variance in precision and recall, respectively, for ϵ ranging from 1 to 10 for each initialization scheme. The results for the discrete and continuous objective functions are shown in the left and right panels, respectively.

Similar to the number of patterns, once ϵ is fixed, matrix size also effects PROXIMUS's performance in terms of runtime and compression ratio, rather than quality of approximation. Therefore, much faster decompositions with relatively lower accuracy are possible for large matrices. We discuss the runtime scalability of PROXIMUS in terms of matrix size and number of patterns in Section 5.2.5.

5.2.4 *Effect of Various Parameters.* The performance of PROXIMUS on the M100K dataset in terms of runtime and compression ratio, and the quality of approximation in terms of error per-row, percentage accuracy, precision, and recall for varying bound on Hamming radius (ϵ) are shown in Figures 9 and 10, respectively. In each figure, the results for the discrete and continuous objective functions are shown on the left and right panels, respectively.

Effect of Bound on Hamming Radius. All experiments in Figures 9 and 10 are performed for bound on Hamming radius ranging from 1 to 10, where $\epsilon = 10$ provides a pretty loose bound, since the average number of nonzeros in a row is 10. However, as seen in Figure 10, the precision and recall provided by

PROXIMUS' decomposition remains well above 80% even for such a loose bound on Hamming radius. Furthermore, precision remains above 98% for the discrete objective function, and recall remains above 98% for *random-row* and *GGG* initialization schemes. This is due to the nature of the algorithm we adopt in PROXIMUS. Namely, each rank-one approximation provides a fairly accurate approximation, which ensures that a row that is present in the representative matrix shares at least half of the nonzeros in the pattern vector, as stated by Equation (4). This approximation is further improved recursively during the course of decomposition, with bound on Hamming radius determining the depth of this recursion. Note also that although the average Hamming error per-row increases almost quadratically with increasing ϵ, this is not reflected in the cumulative accuracy of approximation because even at its highest point ($\epsilon = 10$), the average error per-row is about 3, which is only one-third of the average number of nonzeros per-row.

On the other hand, relaxing the bound on Hamming radius improves compression where the growth in compression ratio with respect to the growth in ϵ is sublinear for $\epsilon < 5$ and superlinear for $\epsilon > 5$. Similarly, a loose bound on Hamming radius provides a significantly faster decomposition. Therefore, since the loss in accuracy for this speedup, and improvement in compression is relatively less significant, PROXIMUS may be used to obtain fast and compact approximations with sufficient quality in many applications.

Effect of Initialization Scheme. The behavior of each variable with varying ϵ is plotted for all initialization schemes in Figures 9 and 10. The *allones* and *center* initialization schemes are not shown in the table to save space, since their performance is relatively poor compared to the other schemes for the reasons discussed in Section 4.4.2. An important observation revealed by these figures is that the choice of initialization scheme affects the runtime significantly, while the obtained compression ratio is relatively independent of the initialization scheme. It is also interesting to note that the recall value of the decomposition is more dependent on the initialization scheme when the discrete objective function is used, while precision is more initialization-dependent with the continuous objective function. More specifically, we can derive the following conclusions on the performance of each initialization strategy:

—*Maximum:* The quality of approximation decays significantly for looser bounds on Hamming radius with this initialization scheme. Moreover, it provides the slowest decomposition when coupled with continuous objective function.

—*Partition:* While being relatively robust to loose bounds on Hamming radius, it is generally slow, especially with the discrete objective function.

—*Greedy Graph Growing (GGG):* One of the best initialization schemes in terms of both runtime performance and approximation quality. It keeps recall above 90% and precision around 99% independent of the value of ϵ and the objective function used.

—*Neighbor:* While being one of the fastest initialization schemes, it is not very robust for loose bounds on Hamming radius.

—*Random-Row:* This scheme is extremely fast compared to other schemes. Moreover, it provides the best accuracy with the continuous objective function and is among the best with the discrete objective function. It is also as robust to loose bounds on Hamming radius as GGG. Therefore, it is the method of choice for many practical applications.

—*Random:* This initialization scheme provides the best compression for most values of ϵ, although the difference is not very significant. While it maintains quality for higher ϵ when used with the discrete objective function, it is does not couple well with the continuous objective function.

Effect of Objective Function. A comparison of the left and right panels of Figures 9 and 10 shows that the discrete objective function is better in terms of runtime, compression ratio, and accuracy. On the other hand, the continuous objective function provides slightly better quality in approximation for tighter bounds on Hamming radius. Based on these observations, we can speculate that the discrete objective function provides better rank-one approximations with more localized pattern vectors and smaller Hamming radius. On the other hand, the continuous objective function provides rank-one approximations with more general pattern vectors present in more rows. Hence, the continuous objective function is associated with deeper recursion trees, providing slightly better approximation at the cost of longer running time and less compression.

Another interesting observation is that precision remains around 99% for any value of ϵ with the discrete objective function. This is true for all test matrices. We observe the same pattern when the continuous objective function is coupled with the *greedy graph growing* or *random-row* initialization schemes. This is because PROXIMUS tends to prune out nonzeros in the matrix that do not significantly belong to any pattern. These nonzero entries can be considered as noise, taking into account the fact that the data generator adds some random noise based on between-pattern correlation and pattern confidence parameters. Therefore, a decomposition for a higher error value (*i.e.*, smaller recall) may also be regarded as a noise filtering decomposition rather than a low-quality decomposition, to a certain extent. This is similar to the application of truncated SVD in information retrieval.

5.2.5 *Runtime Scalability of* PROXIMUS. We perform two experiments to illustrate the scalability of PROXIMUS in terms of matrix size and number of patterns. The settings for these experiments are as follows:

—The number of patterns is kept constant at 100. The number of rows ranges from approximately 10K to 3M. Note that number of nonzeros grows linearly with number of rows ranging from 100K to 30M, while the number of columns remains constant at about 1000.

—The number of rows is kept constant at 100K. The number of patterns range from 20 to 1000. Note that the number of columns grows linearly from 200 to 10K with increasing number of patterns, while number of nonzeros remains constant at about 1M.

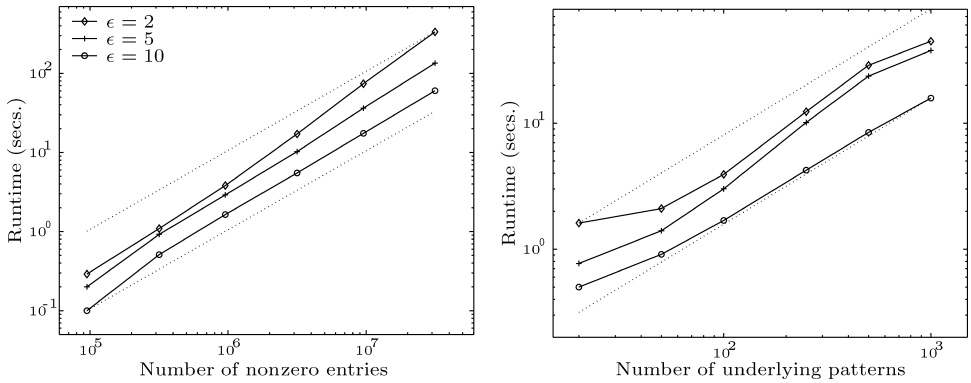


Fig. 11. Runtime of PROXIMUS with respect to matrix size and number of underlying patterns. On the left is the log-log plot of number of nonzero entries vs runtime for three values of ϵ. The dotted lines show two linear functions of number of nonzeros. Similarly, on the right is the log-log plot of number of underlying patterns vs runtime in comparison with two dotted linear functions of number of patterns.

We run PROXIMUS for data instances that correspond to six sample points for each parameter by setting the initialization scheme to *random-row* and using the discrete objective function. We perform experiments for tight ($\epsilon = 2$), moderate ($\epsilon = 5$), and loose ($\epsilon = 10$) bounds on Hamming radius. Each experiment is repeated 10 times on a Pentium-IV 3.0 GHz server with 512 MB RAM.

The log-log plots of matrix size versus runtime in seconds are shown on the left in Figure 11. We also plot two dotted lines that correspond to two linear functions of matrix size, with slopes 1.0×10^{-6} and 1.0×10^{-5} for reference. As evident in the figure, the lines for loose and moderate bounds on Hamming radius are almost parallel to the reference lines, showing that the behavior of runtime with respect to matrix size is linear. For tighter bounds on Hamming radius, on the other hand, the line has a larger slope than the reference lines, showing slight superlinearity, which is expected since the number of representative patterns increases significantly for tighter bounds on Hamming radius.

The log-log plots of number of underlying patterns versus runtime in seconds are shown on the right in Figure 11. Similar to the previous case, we plot two dotted lines that correspond to two linear functions of number of patterns, with slopes 0.015 and 0.08 for reference. As seen in the figure, the line for $\epsilon = 10$ is sublinear with respect to the number of patterns, while the behavior of runtime is sublinear at the two ends of the sampling range and slightly superlinear at the middle for the other values of ϵ. Note that, generally, the number of identified vectors is slightly superlinear in terms of the number of underlying patterns.

5.3 Application Studies

We present the use of PROXIMUS in two diverse applications with a view to demonstrating the versatility of the library.

5.3.1 *Association Rule Mining.* Our first application uses PROXIMUS as a preprocessing tool for accelerating conventional association rule mining (ARM)

algorithms. The idea is to first reduce the dataset using PROXIMUS and then to apply conventional association rule mining algorithms to the reduced data. In ARM, the input is a set of items and a set of transactions, each of which is a subset of the global set of items. Association rules relate co-occurrences of itemsets based on support and confidence metrics. For instance, a rule like $\{bread, butter\} \Rightarrow \{milk\}$ with 20% support and 90% confidence indicates that 20% of the transactions in the database contain these three items together, and 90% of the transactions that contain bread and butter also contain milk. ARM aims at finding all association rules that satisfy user-defined thresholds on support and confidence, which is a computationally challenging problem since the space of itemsets grows exponentially and the number of transactions tends to be very large (in the order of millions) in general [Hipp et al. 2000].

Transaction sets are naturally modeled by sparse binary matrices, where rows correspond to transactions, columns correspond to items, and a one indicates the occurrence of an item in a transaction. Since PROXIMUS returns a minimal set of vectors that are close to the entire set of rows in the matrix within a certain Hamming distance, we can use a single pattern vector discovered by PROXIMUS to represent all transactions in which it is present. Associating each representative transaction (pattern vector) with a weight that corresponds to the number of transactions that contains this pattern, it is possible to mine the set of representative transactions using conventional mining algorithms to speedup the mining process. Since the number of transactions is a major factor for the runtime performance of mining algorithms, compressing the transaction set is expected to speedup the process significantly [Koyutürk and Grama 2003].

In our experiments, we use an efficient implementation [Borgelt 1996] of the well-known *a-priori* algorithm [Agrawal and Srikant 1994] as the benchmark algorithm for association rule mining. We mine a benchmark transaction database named *connect* obtained from the FIMI workshop data pages.[1] This dataset contains 67558 transactions over 129 items. Decomposing the binary matrix that corresponds to this dataset using PROXIMUS in 1192 seconds, we are able to represent this database with only 6703 representative transactions. Comparison of the performances of the a-priori algorithm on original and representative transaction sets in terms of runtime and discovered rules is shown in Table 5.3.1. In our experiments, we fix the support threshold to 20% and the confidence threshold varies from between 50% and 90%. For a confidence threshold of 50%, a-priori is able to discover 3.12 million rules in 4766 seconds on the original transaction set. On the other hand, the same algorithm discovers 2.93 million rules in only 447 seconds on the representative transaction set, 2.8 million of which are rules that are also discovered on the original transaction set. Hence, PROXIMUS is able to achieve a speedup of about 10, which is consistent with the compression ratio in terms of number of transactions. Note that the cost of a single run of a-priori on the original transaction set is about four times the runtime of PROXIMUS! As meaningful association rules are mined by repeatedly varying confidence and support values until a suitable rule set is

[1] http://fimi.cs.helsinki.fi/data/

Table V. Comparison of the Performance of A-Priori Algorithm on the *Connect* Dataset and the Representative Transaction Set (obtained by decomposing the original set using PROXIMUS, in terms of runtime and discovered rules)

Confidence	ARM Time		Speed	# Rules ($\times 10^6$)			Recall	Precision
(%)	Orig. (s)	Appx. (s)	Up	Orig.	Appx.	Common	(%)	(%)
50	4766	447	10.7	3.12	2.93	2.80	89.8	95.6
70	3988	388	10.3	2.52	2.40	2.25	89.6	94.0
90	3335	333	10.0	1.73	1.79	1.56	90.1	87.1

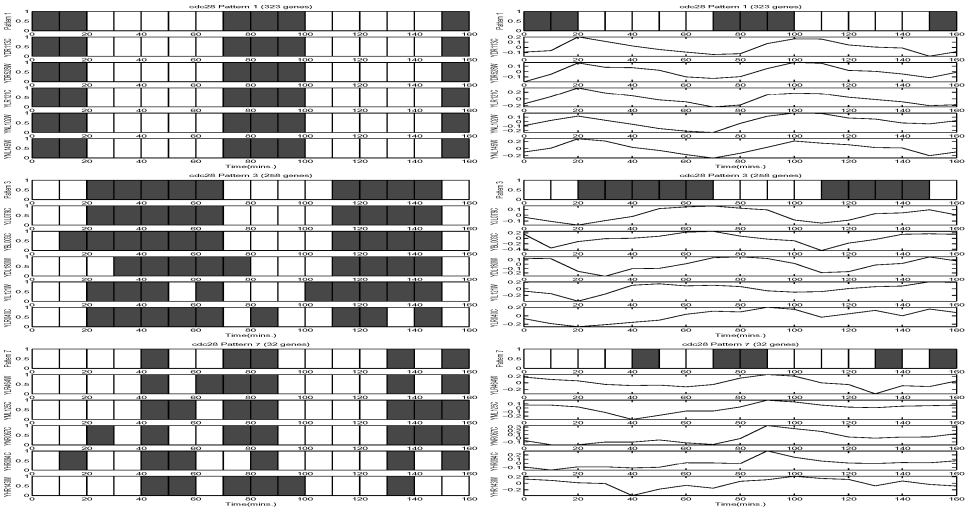
determined, the speedup provided by PROXIMUS as a preprocessor becomes even more impressive.

The recall and precision values measuring the quality of approximation are also presented in the table. Recall is the percentage of rules discovered on the representative transaction set among all rules discovered on the original set. Precision, on the other hand, is the percentage of rules discovered on the original transaction set among all rules discovered on the representative set. As seen in the table, both precision and recall values are around 90% for all values of confidence threshold. The performance of PROXIMUS on association rule mining along with the effects of the properties of the input dataset, initialization schemes, and bound on Hamming radius are discussed in detail in Koyutürk and Grama [2003].

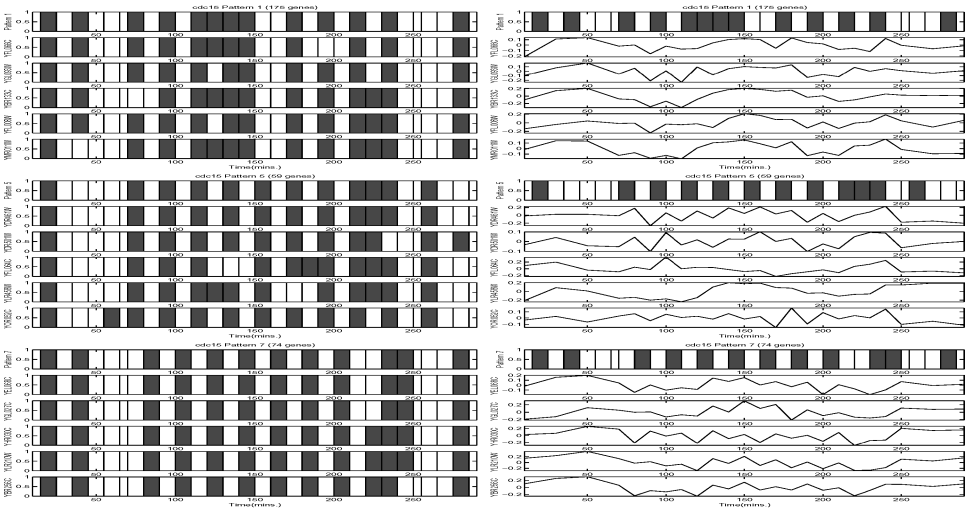
Based on these results, we establish PROXIMUS as an effective preprocessor for data mining, which can be used for compressing large datasets before applying conventional data mining algorithms. The speed and quality of the approximation provided by PROXIMUS can be also useful in distributed data mining algorithms, where sites are loosely-coupled, or privacy is an issue [Chi et al. 2004].

5.3.2 *Discovery of Regulation Patterns in DNA Microarray Experiments.* We also demonstrate the use of PROXIMUS in the context of microarray data analysis [Koyutürk et al. 2003]. Our objective in this application is to examine coregulation (up- and down-regulation) in groups of genes. With this goal, we convert expression data from a selected microarray experiment for each gene into a binary vector of length equal to number of samples. Each component of the vector is assigned a value 0 if expression was down-regulated during the period and 1 otherwise. Considering the binary regulation vector of each gene as a row, this gives us a binary regulation matrix for the underlying microarray experiment. We then decompose this matrix using PROXIMUS to determine a suitable set of representative regulation patterns characterized by pattern vectors along with a partitioning (and assignment) of the genes to these patterns characterized by presence vectors. Each partition represents a set of genes that are coregulated to within specified tolerance, which corresponds to the bound on Hamming radius in the decomposition. This partitioning can then be used to identify motifs in genes that control regulation.

We apply our method to microarray data from four experiments on yeast cultures synchronized by the following methods: α-factor arrest (dataset alpha), elutriation (dataset elu), and arrest of cdc15/cdc28 (datasets cdc15/cdc28) temperature-sensitive mutants [Spellman et al. 1998; Cho et al. 1998]. Dataset



Clusters 1, 3, 7 of dataset CDC28.



Clusters 1, 5, 7 of dataset CDC15.

Fig. 12. Selected clusters from datasets CDC28 and CDC15, the representative patterns of these clusters, and some members of the clusters illustrating excellent coregulation properties. Binary regulation vectors are shown on the right panel, where dark and white boxes indicate up- and down-regulation, respectively. Actual expression profiles of the corresponding genes are shown on the right panel.

alpha corresponds to samples taken at 7-minute intervals for 140 minutes, dataset cdc15 contains samples taken every 10 minutes for 300 minutes, dataset cdc28 contains samples taken every 10 minutes for 160 minutes, and dataset elu contains samples taken every 30 minutes for 330 minutes.

In Figure 12, we select some of the patterns from each dataset and demonstrate the excellent clustering properties of PROXIMUS. In the figure, the left

panel shows binary vectors with dark and white boxes indicating up- and down-regulation, respectively. In the right panel, we display the original gene expression vectors that correspond to the binary regulation vectors on the left.

In the top panel of Figure 12, we illustrate three patterns from dataset cdc28. The top pattern in each case is the representative pattern, which is identified by PROXIMUS as a pattern vector in the decomposition of the binary regulation matrix. The following five rows correspond to five sample genes from the data, which are selected from the set of genes that contain this pattern (*i.e.*, the set of rows that have a one in the corresponding presence vector of this pattern). The left panel illustrates the pattern in comparison to up- and down-regulation data (0/1 discretized expression data). As seen in the figure, the first pattern discovered by PROXIMUS on the cdc28 dataset exactly represents the regulation characteristics of the five genes whose binary regulation vectors are displayed under itself. This pattern is present in 322 of the genes, which also contain this regulation pattern with a tolerance distance of, at most, 3 in terms of Hamming distance to the representative pattern. On the right of each binary vector is the actual gene expression data for the corresponding gene. Observe that the five sample genes shown in the figure display exactly the same regulation characteristics (*i.e.*, their expression profiles in the cdc28 experiment are highly correlated).

Decomposition of the binary regulation matrix for the cdc28 dataset reveals that the next pattern in the figure is present in 257 genes. The binary regulation vectors and expression profiles of five of these genes are also shown in the figure. As seen in the figure, although the first sample shows a regulation pattern that is exactly the same as the representative pattern, the second gene has a regulation pattern that is at Hamming distance 1 from the representative pattern. However, examination of the actual expression profiles of these two genes reveals that the general regulation characteristics of these two genes are highly correlated in the cdc28 experiment.

In general, PROXIMUS is able to discover 13 regulation patterns in the alpha dataset, 10 in cdc15 dataset, 8 in cdc28 dataset, and 7 in the elu dataset.

6. CONCLUSIONS AND ONGOING WORK

This article presents the design principles, data structures, implementation details, and use of PROXIMUS, an efficient software tool for error-bounded compression of large, high-dimensional binary attributed datasets. We also present detailed experimental results examining various performance characteristics and provide sample applications in diverse areas. PROXIMUS finds applications in many other areas, including clustering, classification, and dominant and deviant pattern detection. Our experiments illustrate that PROXIMUS is scalable to very large datasets of high-dimensions, making it suitable for a variety of applications. The data structures and implementation schemes described in this article can also be used in several applications involving sparse binary datasets, exploiting the advantages of the sparse and binary nature of such datasets.

PROXIMUS can be further improved by exploring more effective initialization strategies and refinement heuristics. It can also be generalized to discrete

datasets that are not necessarily binary. Such datasets appear frequently in a range of applications, such as image compression and pattern matching.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contribution of anonymous referees and associate editor John Reid, whose comments and suggestions strengthened this manuscript significantly.

REFERENCES

AGRAWAL, R. AND SRIKANT, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*. 487–499.

BERRY, M. W., DUMAIS, S. T., AND O'BRIEN, G. W. 1995. Using linear algebra for intelligent information retrieval. *SIAM Rev. 37,* 4, 573–595.

BOLEY, D. 1998. Principal direction divisive partitioning. *Data Mining Knowl. Discovery 2,* 4, 325–344.

BORGELT, C. 1996. Finding association rules/hyperedges with the apriori algorithm. http://fuzzy.cs.Uni-Magdeburg.de/ borgelt/apriori/apriori.html.

BRON, C. AND KERBOSCH, J. 1973. Finding all cliques in an undirected graph. *Commun. ACM 16*, 575–577.

CHI, J., KOYUTÜRK, M., AND GRAMA, A. 2004. CONQUEST: A distributed tool for constructing summaries of high-dimensional discrete-attributed datasets. In *Proceedings of the 4th SIAM International Conference on Data Mining (SDM 2004)*. 154–165.

CHO, R. J., CAMPBELL, M. J., WINZELER, E. A., STEINMETZ, L., CONWAY, A., WODICKA, L., WOLFSBERG, T. G., GABRIELIAN, A. E., LANDSMAN, D., LOCKHART, D. J., AND DAVIS, R. W. 1998. A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular Cell 2,* 1, 65–73.

CHU, M. T. AND FUNDERLIC, R. E. 2002. The centroid decomposition: Relationships between discrete variational decompositions and SVDs. *SIAM J. Matrix Anal. Appl. 23,* 4, 1025–1044.

COVER, T. M. AND THOMAS, J. A. 1991. *Elements of Information Theory*. Wiley & Sons, New York.

DUFF, I. S., ERISMAN, A. M., AND REID, J. K. 1987. *Direct Methods for Sparse Matrices*. Clarendon Press, New York.

GIBSON, D., KLEINGBERG, J., AND RAGHAVAN, P. 1998. Clustering categorical data: An approach based on dynamical systems. In *Proceedings of the 24th Very Large Data Bases Conference.*

GRAY, R. M. 1984. Vector quantization. *IEEE ASSP 1,* 2, 4–29.

GUHA, S., RASTOGI, R., AND SHIM, K. 2000. ROCK: A robust clustering algorithm for categorical attributes. *Inf. Syst. 25,* 5, 345–366.

GUPTA, G. AND GHOSH, J. 2001. Value balanced agglomerative connectivity clustering. In *SPIE Proceedings*.

HAN, E., KARYPIS, G., KUMAR, V., AND MOBASHER, B. 1998. Hypergraph-based clustering in high-dimensional datasets: A summary of results. *Bull. Tech. Committee Data Eng. 21,* 1, 15–22.

HIPP, J., GÜNTZER, U., AND NAKHAEIZADEH, G. 2000. Algorithms for association rule mining—A general survey and comparison. *SIGKDD Explorations 2,* 1, 58–64.

HUANG, Z. 1997. A fast clustering algorithm to cluster very large categorical data sets in data mining. In *Research Issues on Data Mining and Knowledge Discovery*.

IBM. Quest synthetic data generation code. http://www.almaden.ibm.com/cs/quest/syndata.html.

JOHN, G. H. AND LANGLEY, P. 1996. Static versus dynamic sampling for data mining. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, KDD*, E. Simoudis et al., eds. AAAI Press, 367–370.

KARYPIS, G. AND KUMAR, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20,* 1, 359–392.

KOLDA, T. G. AND O'LEARY, D. P. 1998. A semidiscrete matrix decomposition for latent semantic indexing information retrieval. *ACM Trans. Inf. Syst. 16,* 4, 322–346.

KOLDA, T. G. AND O'LEARY, D. P. 2000. Algorithm 805: Computation and uses of the semidiscrete matrix decomposition. *ACM Trans. Math. Softw. 26,* 3, 415–435.

KOYUTÜRK, M. AND GRAMA, A. 2003. PROXIMUS: A framework for analyzing very high-dimensional discrete attributed datasets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003)*. 147–156.

KOYUTÜRK, M., GRAMA, A., AND SZPANKOWSKI, W. 2003. Algorithms for bounded-error correlation of high dimensional data in microarray experiments. In *Proceedings of the 2nd IEEE Computational Systems Bioinformatics Conference (CSB 2003)*. 575–580.

LOAN, C. F. V. 2000. *Introduction to Scientific Computing*. Prentice Hall, Englewood Cliffs, N.J.

MACQUEEN, J. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium*, vol. 1. 281–297.

MCCONNELL, S. AND SKILLICORN, D. B. 2001. Outlier detection using semi-discrete decomposition. Tech. Rep. 2001-452, Dept. of Computing and Information Science, Queen's University.

O'LEARY, D. P. AND PELEG, S. 1983. Digital image compression by outer product expansion. *IEEE Trans. Commu. 31*, 441–444.

ÖZDAL, M. AND AYKANAT, C. 2004. Hypergraph models and algorithms for data-pattern based clustering. *Data Mining and Know. Discovery 9,* 1, 29–57.

PEETERS, R. 2003. The maximum edge biclique problem is NP-complete. *Discrete Appl. Math. 131,* 3, 651–654.

PROVOST, F. J. AND KOLLURI, V. 1999. A survey of methods for scaling up inductive algorithms. *Data Mining Knowl. Discovery 3,* 2, 131–169.

SAAD, Y. 1996. *Iterative Methods for Sparse Linear Systems*. PWS.

SPELLMAN, P. T., SHERLOCK, G., ZHANG, M. Q., IYER, V. R., ANDERS, K., EISEN, M. B., BROWN, P. O., BOTSTEIN, D., AND FUTCHER, B. 1998. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell 9*, 3273–3297.

TOIVONEN, H. 1996. Sampling large databases for association rules. In *Proceedings of the 22th International Conference on Very Large Databases (VLDB'96)*. 134–145.

ZAKI, M. J., PARTHASARATHY, S., LI, W., AND OGIHARA, M. 1996. Evaluation of sampling for data mining of association rules. Tech. Rep. TR617.

ZYTO, S., GRAMA, A., AND SZPANKOWSKI, W. 2002. Semi-discrete matrix transforms (SDD) for image and video compression. In *Process Coordination and Ubiquitous Computing*, D. Marinescu and C. Lee, eds. Kluwer, Amsterdam, 249–259.

Received January 2004; revised June 2005; accepted June 2005