Compression, Clustering, and Pattern Discovery in Very High-Dimensional Discrete-Attribute Data Sets

Mehmet Koyutürk, Ananth Grama, and Naren Ramakrishnan, *Member*, *IEEE Computer Society*

Abstract—This paper presents an efficient framework for error-bounded compression of high-dimensional discrete-attribute data sets. Such data sets, which frequently arise in a wide variety of applications, pose some of the most significant challenges in data analysis. Subsampling and compression are two key technologies for analyzing these data sets. The proposed framework, PROXIMUS, provides a technique for reducing large data sets into a much smaller set of representative patterns, on which traditional (expensive) analysis algorithms can be applied with minimal loss of accuracy. We show desirable properties of PROXIMUS in terms of runtime, scalability to large data sets, and performance in terms of capability to represent data in a compact form and discovery and interpretation of interesting patterns. We also demonstrate sample applications of PROXIMUS in association rule mining and semantic classification of term-document matrices. Our experimental results on real data sets show that use of the compressed data for association rule mining provides excellent precision and recall values (above 90 percent) across a range of problem parameters while reducing the time required for analysis drastically. We also show excellent interpretability of the patterns discovered by PROXIMUS in the context of clustering and classification of terms and documents. In doing so, we establish PROXIMUS as a tool for both preprocessing data before applying computationally expensive algorithms and directly extracting correlated patterns.

Index Terms—Clustering, classification, association rules, data mining, sparse, structured and very large systems, singular value decomposition.

1 INTRODUCTION

With the availability of large-scale computing platforms for high-fidelity design and simulations, and instrumentation for gathering scientific as well as business data, increased emphasis is being placed on efficient techniques for analyzing large and extremely high-dimensional data sets. These data sets may comprise discrete attributes, such as those from business processes, information retrieval, and bioinformatics, as well as continuous attributes such as those in scientific simulations, astrophysical measurements, and engineering design. 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 to large data sets. 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 data sets, 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 reduction. This paper focuses on algorithms and heuristics for error-bounded compression of very large high-dimensional discrete-attribute data sets.

Compression of discrete 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 do not apply 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 extremely 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 of very high dimension.

In order to overcome the computational requirements of the problem while providing efficient analysis of data, we propose a new technique—binary($\{0,1\}$) nonorthogonal matrix transformation to extract dominant patterns. In this technique, elements of singular vectors of a binary valued matrix are constrained to binary entries with an associated singular value of 1. Since this modification results in a heuristic approximation to a singular vector, we refer to these vectors as *approximation vectors* in the rest of this paper to avoid confusion. In contrast, in a related technique called Semi-Discrete Decomposition (SDD), elements of singular vectors are in the set $\{-1,0,1\}$ and the associated singular value is continuous. We show here that our variant results in an extremely efficient algorithm and powerful framework within which large data sets can be summarized.

- *M. Koyutürk and A. Grama are with the Department of Computer Sciences, Purdue University, West Lafayette, IN 47907. E-mail: {koyuturk, ayg}@cs.purdue.edu.*
- *N. Ramakrishnan is with the Department of Computer Science, Virginia Tech, Blacksburgh, VA 24061. E-mail: naren@cs.vt.edu.*

Manuscript received 22 Oct. 2003; revised 21 June 2004; accepted 22 Oct. 2004; published online 17 Feb. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0211-1003.

1041-4347/05/$20.00 © 2005 IEEE Published by the IEEE Computer Society

PROXIMUS is a nonorthogonal matrix transform based on recursive partitioning of a data set depending on the distance of a relation from the dominant pattern. The dominant pattern is computed as a binary approximation vector of the matrix of relations. PROXIMUS computes only the first approximation vector and, consequently, each discovered pattern has a physical interpretation at all levels in the hierarchy of the recursive process. For the discovery of the dominant approximation vector, we adopt an iterative alternating heuristic. Due to the discrete nature of the problem, initialization of approximation vectors is critical for convergence to desirable local optima. Taking this fact into account, we derive effective initialization strategies, along with algorithms for a multiresolution representation of the data set.

PROXIMUS provides several facilities to analyze discrete attributed data. These include:

- discovering dominant and deviant patterns in the data in a hierarchical manner,
- 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.

We also demonstrate the use of PROXIMUS for preprocessing data for subsequent analysis using conventional techniques. Using the apriori algorithm [1] for association rule mining, we clearly show PROXIMUS' ability to accurately represent data in a very compact form. Our experimental results show that use of the compressed data for association rule mining provides excellent precision and recall values (above 90 percent) across a range of support thresholds while reducing the time required for association rule mining by several orders of magnitude.

In the next section, we discuss the use of matrix transforms in the context of data analysis and compression and review existing approaches based on probabilistic subsampling, matrix decomposition, and latent structure analysis. In Section 3, we present the basic idea of PROXIMUS using representative examples, formulate the problem, and provide heuristics to solve the discrete rank-one approximation problem efficiently, and present our recursive algorithm for hierarchical discovery of patterns. In Section 4, we present an application of PROXIMUS in association rule mining. We demonstrate the effectiveness of PROXIMUS on both synthetic and experimental data in the context of a variety of applications and illustrate its scalability to large data sets in Section 5. 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 data sets focus on probabilistic subsampling and data compression. Data reduction techniques based on probabilistic subsampling have been explored by several researchers [2], [3], [4], [5], [6]. 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. From the pattern discovery and data analysis point of view, data reduction can also be regarded as discovery of latent structures in the data, which is closely related to matrix decomposition. There is also significant literature on the analysis of latent structure in continuous domain that are based on matrix decomposition, probability, and signal processing. In the rest of this section, we summarize commonly used orthogonal and nonorthogonal matrix transformations, latent structure analysis, and their applications in data analysis and explore alternate approaches for binary data sets.

2.1 Orthogonal and Nonorthogonal Matrix Decompositions

Singular Value Decomposition (SVD) is an orthogonal matrix decomposition that is used extensively in applications ranging from Principal Component Analysis (PCA) to dimensionality reduction. SVD transforms a matrix into two orthogonal matrices and a diagonal matrix of singular values. Specifically, an m by n rectangular matrix A can be decomposed into $A = U\Sigma V^T$, where U is an $m \times r$ orthogonal matrix, V is an $n \times r$ orthogonal matrix, and Σ is an $r \times r$ diagonal matrix of the singular values of A. Here, r denotes the rank of matrix A. The matrix $\tilde{A} = u_1\sigma_1 v_1^T$ is a rank-one approximation of A, where u_1 and v_1 denote the first columns of matrices U and V, respectively. This is the best rank-one approximation to A in minimum least squares sense. 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 data set with rows corresponding to relations and columns corresponding to attributes, we can say that each 3-tuple consisting of a singular value σ_k, kth column in U, and kth column in V represents a pattern in A characterized by σ_k. Larger singular values imply that the corresponding pattern is more dominant in the data set. A common algorithm in information retrieval, Latent Semantic Indexing (LSI) [7], exploits this property of SVD to summarize the underlying data represented by matrix A by truncating the SVD of A to an appropriate number of singular values so that the insignificant patterns corresponding to small singular values are filtered.

Semi-Discrete Decomposition (SDD) is a variant of SVD in which the values of the entries in matrices U and V are constrained to be in the set $\{-1, 0, 1\}$ [8]. The main advantage of SDD is its lower storage requirement, since each element only requires 1.5 bits, thus enabling a higher rank representation for a given amount of memory. Since the entries of the singular 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 [8] 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. Although PROXIMUS is closely related to SDD, it is different in the sense that it partitions data based on approximations rather than extracting the approximation.

Centroid Decomposition (CD) is an approximation to SVD that is widely used in factor analysis [9]. 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. CD runs in linear time in a number of rows of the matrix but 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 determined that takes advantage of the discrete nature of data and is much sparser.

Principal Direction Divisive Partitioning (PDDP) is a hierarchical clustering strategy for high-dimensional real-valued sparse data sets [10]. PDDP partitions documents (rows) into two parts, recursively, based on the principal direction of the document-term matrix. The idea of recursively partitioning the matrix based on the first singular vector is also used by PROXIMUS with a heuristic modification. However, PROXIMUS is designed specifically for binary-attributed data and always preserves the sparse and binary nature of the data in contrast to PDDP. This is advantageous in terms of computational resources (PROXIMUS has no FLOPs) and interpretability of the decomposition.

2.2 Latent Variable Analysis and Other Methods for Data Representation

Principal Component Analysis (PCA) [11] and Factor Analysis [12] are two common data analysis methods that are used to explore the latent structure in data. Both of these methods are based on orthogonal matrix decompositions and are closely related to each other. Recently proposed methods such as Probabilistic Latent Semantic Analysis (PLSA) are based on probabilistic modeling of the latent space [13]. PLSA assumes an underlying latent structure that generates the observed data and uncovers this latent structure using the EM Algorithm [14]. Although PROXIMUS is algorithmically similar to PLSA in terms of using iterative projections, it is based on the idea of optimization-based matrix decomposition rather than the assumption of an underlying latent structure. In addition, the recursive structure of PROXIMUS allows hierarchical analysis of the underlying patterns in the data. At the same time, patterns discovered by PROXIMUS can be regarded as latent variables as well. Another technique, Independent Component Analysis (ICA) [15], tries to find a representation for the observed data such that the statistical dependency between the components of representation is minimized. PROXIMUS is different from latent variable-based methods in the sense that it relates each row (document or data item) with exactly one pattern. This allows hierarchical analysis of the underlying cluster structure, taking advantage of the binary nature of data.

2.3 Other Work on Summarizing Discrete-Attribute Data Sets

Other work on summarizing discrete-attributed data sets is largely focused on clustering very large categorical data sets. A class of approaches is based on well-known techniques such as vector-quantization [16] and k-means clustering [17]. The k-modes algorithm [18] 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 to be partitioned and apply partitioning heuristics on this representation. Graph-based approaches represent similarity between pairs of data items using weights assigned to edges and cost functions on this similarity graph [19], [20]. Hypergraph-based approaches are based on the fact that discrete-attribute data sets are naturally described by hypergraphs and directly define cost functions on the corresponding hypergraph [21], [22].

Our approach differs from these methods in that it discovers naturally occurring patterns with no constraint on cluster sizes or number of clusters. Thus, it provides a generic interface to the problem, which may be used in diverse applications. Furthermore, the superior execution characteristics of our approach make it particularly suited to extremely high-dimensional attribute sets.

3 Nonorthogonal Decomposition of Binary Matrices

PROXIMUS is a collection of novel algorithms and data structures that rely on a variant of SDD to determine error-bounded approximations to binary attributed data sets. While relying on the idea of matrix transforms, PROXIMUS provides a framework that captures the properties of discrete data sets more accurately and takes advantage of their binary nature to improve both the quality and efficiency of the analysis. We formulate the problem of error-bounded approximation of binary matrices as follows.

Definition 3.1. *Given m binary vectors $a_1, a_2, \ldots, a_m$ in n-dimensional space, find k $n \times 1$ binary approximation vectors $y_1, y_2, \ldots, y_k$ such that*

$$\forall\, 1 \le i \le m, \exists\, j \text{ s.t. } \|a_i - y_j\|_2^2 \le \epsilon \tag{1}$$

to minimize k, where ϵ is a prescribed error bound.

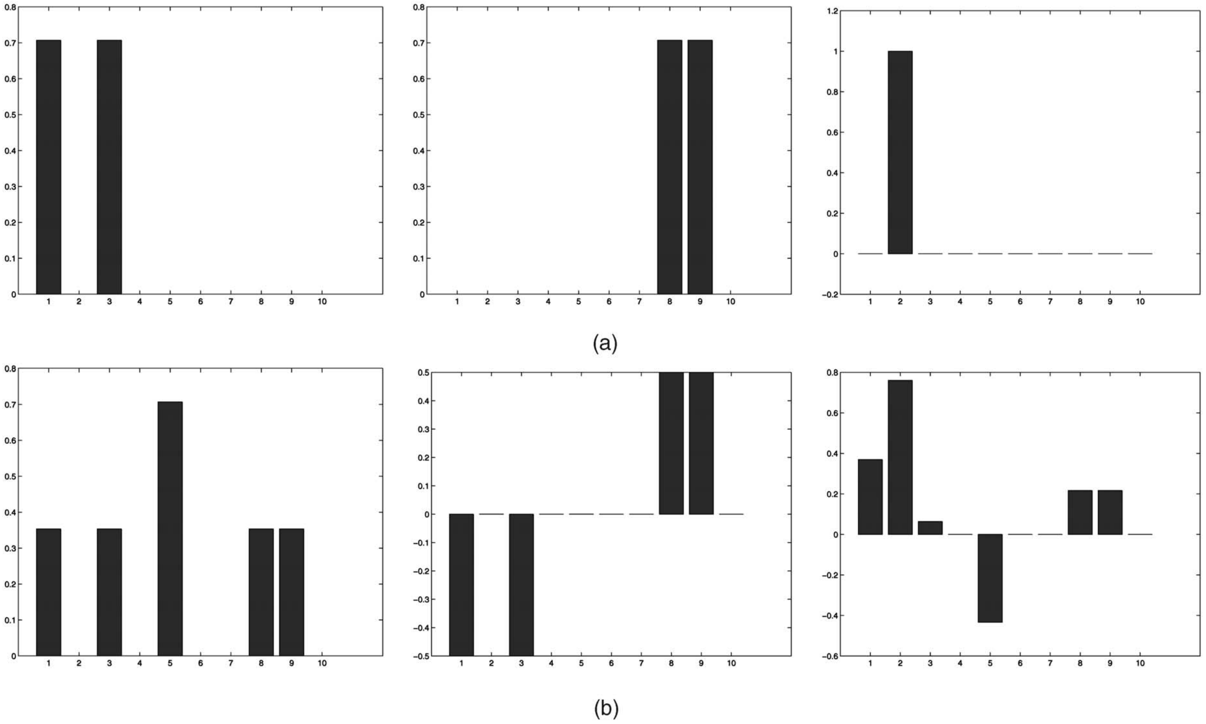
Letting $A = [a_1 a_2 \ldots a_m]^T$ and $Y = [y_1 y_2 \ldots y_m]^T$, this becomes a minimum-rank matrix decomposition problem where $\|A - XY^T\|_\infty \le \epsilon$ and X is an $m \times k$ binary matrix with $x_i = 1$ if and only if y_j is the approximation vector that is of minimum Hamming distance from row a_i and satisfies (1).

Our approach to solving this problem is based on recursively computing discrete rank-one approximations to the matrix to extract dominant patterns hierarchically [23]. This simplifies the problem algorithmically while providing a framework for interpretability and applicability of the approximation. Relying on the fact that rows and columns have different conceptual meanings in many applications (e.g., rows being items and columns being features), and one is generally interested in the underlying patterns spread across the rows, we develop an algorithm that is based on recursively partitioning the set of rows.

The problem of error-bounded approximation can also be thought of as finding dense patterns in sparse matrices. A binary rank-one approximation for a 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 1. Given a matrix A, we compute a rank-one approximation as follows:

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



(a)

(b)

Fig. 1. SVD examples that illustrate difficulty of interpreting results. In each panel, three figures show the most significant singular vectors in the item space of a transaction matrix in decreasing order of dominance from left to right. (a) Nonoverlapping item sets and (b) overlapping item sets.

Here, vector y is 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 2. Consider now a binary matrix A, which does not have an exact rank-one representation (i.e., the matrix is of higher rank):

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

Consider the following rank-one approximation for A:

$$\tilde{A} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} [0 \quad 0 \quad 1 \quad 0 \quad 1] = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

The pattern vector here is $[0\ 0\ 1\ 0\ 1]^T$ and 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 input 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. Then, the approximation is subtracted from the input 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 vectors that are necessary to compute in order 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 the given number of singular vectors. While SVD is useful in some applications involving discrete data sets such as 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 data sets. One such application is illustrated in Section 5.3. 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 the 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. Fig. 1 illustrates this fact. We construct a transaction matrix by assigning elements t_{ij} to the number of instances of item j in transaction i. In Fig. 1a, we show the three dominant singular vectors (rank reduction to three) derived from a synthetic transaction matrix. It is clear from this figure that items 1 and 3 form the most dominant cooccurring set of items followed by items 8 and 9, followed by item 2. However, in the case of overlapping frequent sets, as in the example of Fig. 1b, the orthogonality constraint poses difficulties. In this example, the first vector indicates that items 1, 3, 5, 8, and 9 are most significant. However, 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 or association rules as illustrated in Section 4. Since SDD is based on repeatedly finding rank-one approximations to a residual matrix which is 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 here 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 elements, and
- the matrix is partitioned based on the presence vector after each computation of rank-one approximation, and the procedure is applied recursively to each partition. This method provides a hierarchical representation of dominant 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 3.2: Rank-one approximation. *Given matrix $A \in \{0,1\}^m \times \{0,1\}^n$, find $x \in \{0,1\}^m$ and $y \in \{0,1\}^n$ to minimize the error:*

$$||A - xy^T||_F^2 = |\{a_{ij} \in (A - xy^T) : |a_{ij}| = 1\}|. \qquad (2)$$

In other words, the error for a rank-one approximation is the number of nonzero entries in the residual matrix. This 0-1 integer programming problem with 2^{m+n} feasible points is NP-hard [8]. Indeed, it is closely related to finding maximum cliques in graphs. Although there is considerable literature on the maximum clique and biclique problems [24], [25], 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. As a locally optimal solution for the rank-one approximation problem will be 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 for the entire matrix, while improving the local approximation as well. For this purpose, we adopt an alternating iterative heuristic for computation of approximation vectors for binary matrices, with suitable initialization heuristics.

3.1.1 Alternating Iterative Heuristic

Since the objective (error) function can be written as

$$||A - xy^T||_F^2 = ||A||_F^2 - 2x^T Ay + ||x||_2^2 ||y||_2^2,$$

minimizing the error is equivalent to maximizing

$$C_d(x,y) = 2x^T Ay - ||x||_2^2 ||y||_2^2. \qquad (3)$$

If we fix y and set $s = Ay$, the corresponding x that maximizes this function is given by the following equation:

$$x(i) = \begin{cases} 1, & \textit{if } 2s(i) \geq ||y||_2^2 \\ 0, & \textit{otherwise}. \end{cases} \qquad (4)$$

This equation follows from the idea that a nonzero element of x can have a positive contribution to $C_d(x,y)$ if and only if at least half of the nonzero elements of y match with the nonzero entries on the corresponding row of A. Clearly, this equation leads to a linear time algorithm in the number of nonzeros of A to compute x, as computation of s requires $O(N)$ time and (4) can be evaluated in $O(m)$ time. Here, m is the number of rows and N is the number of nonzeros (ones) in the matrix. Similarly, we can compute vector y that maximizes $C_d(x,y)$ for a fixed x in linear time. This leads to an alternating iterative algorithm based on the computation of SDD [8], 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.

3.2 Recursive Decomposition of Binary Matrices

We use a rank-one approximation of the input matrix to partition the rows into two submatrices. This is in contrast to conventional SVD-based techniques that compute the residual matrix and apply the transformation repeatedly.

Definition 3.3: Partitioning based on rank-one approximation. *Given rank-one approximation $A \approx xy^T$, a partition of A with respect to this approximation results in two submatrices A_1 and A_0, such that*

$$a_i \in \begin{cases} A_1, & \textit{if } x(i) = 1 \\ A_0, & \textit{otherwise} \end{cases} \qquad (5)$$

for $1 \leq i \leq m$. Here, a_i denotes the ith row of A.

The intuition behind this approach is that rows corresponding to 1s in the presence vector are the rows of a maximally connected submatrix of A. Therefore, these rows have more similar nonzero structures among each other compared to the rest of the matrix. 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 some stopping criterion. If so, we decide that matrix A_1 is adequately represented by matrix xy^T and stop; else, we recursively apply the procedure for A_1 as for A_0.

The partitioning-and-approximation process continues until the matrix cannot be further partitioned or 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

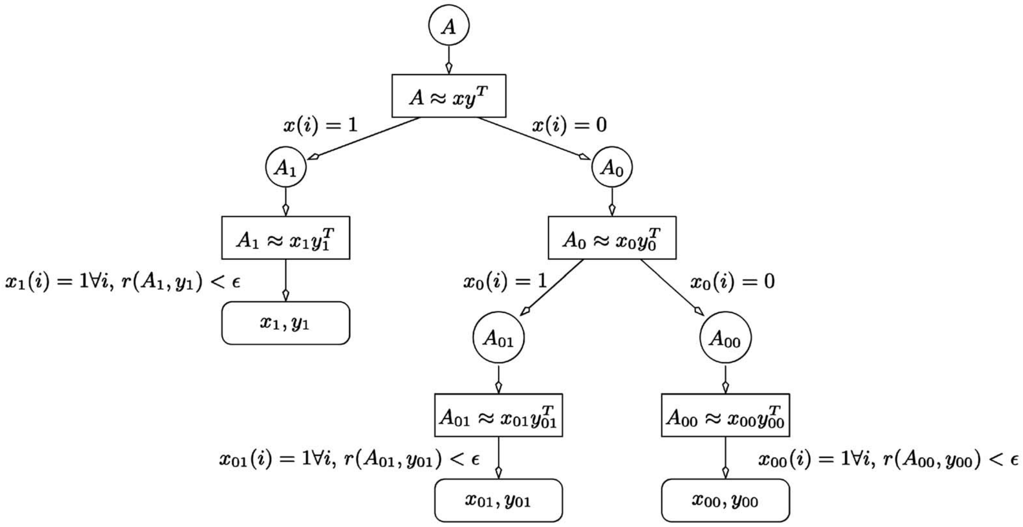

A
$A \approx xy^T$
$x(i) = 1$
$x(i) = 0$
A_1
A_0
$A_1 \approx x_1 y_1^T$
$A_0 \approx x_0 y_0^T$
$x_1(i) = 1 \forall i,\ r(A_1, y_1) < \epsilon$
$x_0(i) = 1$
$x_0(i) = 0$
x_1, y_1
A_{01}
A_{00}
$A_{01} \approx x_{01} y_{01}^T$
$A_{00} \approx x_{00} y_{00}^T$
$x_{01}(i) = 1 \forall i,\ r(A_{01}, y_{01}) < \epsilon$
$x_{00}(i) = 1 \forall i,\ r(A_{00}, y_{00}) < \epsilon$
x_{01}, y_{01}
x_{00}, y_{00}

Fig. 2. Recursive structure of PROXIMUS. 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.

of the representation provided by a rank-one approximation, regarding pattern vector as the centroid of this set of rows.

Definition 3.4: Hamming radius. *Given a set of binary vectors $R = \{x_1, x_2, \ldots, x_n\}$ and a binary vector y, the Hamming radius of R centered around y is defined as:*

$$r(R, y) = \max_{1 \le i \le n} h(x_i, y), \tag{6}$$

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.
- $x_i(j) = 1\ \forall j$, 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 dominant 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 $\tilde{A} = XY^T$, where 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.

Fig. 2 illustrates the recursive structure of PROXIMUS. Starting with matrix A, a rank-one approximation to A is computed. The matrix A is then partitioned into A_1 and A_0 based on the presence vector x. The rank-one approximation to A_1 returns a presence vector of all 1s and the approximation is adequate so the recursion stops at that node and y_1 is recorded as a dominant pattern. On the other hand, matrix A_0 is further partitioned as the approximation $A_0 \approx x_0 y_0^T$ does not cover all rows of A_0. The overall decomposition is $A \approx XY^T$, where $X = [x_1, x_{01}, x_{00}]^T$ and $Y = [y_1, y_{01}, y_{00}]^T$.

3.3 Initialization of Iterative Process

While finding a rank-one approximation, initialization is crucial not only for the rate of convergence but also the quality of the solutions since a wrong choice can result in poor local minima. 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 the pattern vector must not require more than $\Theta(N)$ operations, since it will otherwise dominate the runtime of the overall algorithm. Possible procedures for finding an initial pattern vector include:

- **Partition**. Select a separator column and identify the rows that have a nonzero at 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**. Based on the idea of iterative improvement heuristics in graph partitioning [26], this scheme starts with a randomly selected row in one part and grows the part by including rows that share a nonzero with that part until a balanced partition is obtained. The initial pattern vector is set to the centroid of rows in this part.
- **Random-row**. Observing that a balanced partition of rows is not necessary due to the nature of the problem, we select one row randomly and initialize the pattern vector to that row with the expectation that it shares some nonzeros with the rows that share the same pattern with itself.

All of the above initialization schemes require $O(N)$ time. Our observations indicate that the *Random-row* scheme tends to initialize the pattern vector close to a desired local minimum, i.e., the resulting rank-one

T_1 : {beer, snacks}
T_2 : {beer, snacks, bread}
T_3 : {milk, bread}
T_4 : {milk, bread, butter}
T_5 : {milk, butter}
T_6 : {bread, butter}

(a)

$$T = \begin{array}{cccccc} & \text{beer} & \text{snacks} & \text{bread} & \text{milk} & \text{butter} \\ T_1 & 1 & 1 & 0 & 0 & 0 \\ T_2 & 1 & 1 & 1 & 0 & 0 \\ T_3 & 0 & 0 & 1 & 1 & 0 \\ T_4 & 0 & 0 & 1 & 1 & 1 \\ T_5 & 0 & 0 & 0 & 1 & 1 \\ T_6 & 0 & 0 & 1 & 0 & 1 \end{array}$$

(b)

Fig. 3. (a) A sample transaction set of six transactions on five items and (b) its corresponding transaction matrix.

approximation includes a specific pattern that represents a small set of rows adequately. On the other hand, *Greedy Graph Growing* provides hierarchical extraction of patterns, the resulting rank-one approximation generally contains a combination of patterns, which can be further decomposed in the recursive course of the algorithm. The *Partition* scheme lies somewhere between the first two schemes as the balance of the partition depends on the selection of the dimension. In our implementation of this scheme, we select the dimension that yields the most balanced partition in order to increase the probability of partitioning along a significant dimension.

3.4 Generalization of Proposed Framework

Throughout the discussion of the proposed framework, we have considered rows of a matrix as data items and columns as features and assumed that patterns of interest lie in rows. While this assumption is valid in many applications, it might be necessary to consider patterns in other dimensions as well, in some cases. PROXIMUS is easily extendible to such instances as follows:

- If we are interested in column patterns, PROXIMUS is directly applicable on the transpose of the matrix. Specifically, decomposition in each dimension (rows or columns) also reveals some interpretable pattern structure on the other dimension since both pattern and presence vectors are binary. This property is illustrated on document-term matrices in Section 5.3.
- PROXIMUS can also be modified to capture pattern structure in both row and column spaces. This can be done by computing a binary residual to the matrix by extracting the rank-one approximation from the matrix ($A_r = A \& \overline{xy^T}$, where & and — denote binary AND and NOT operations) and decomposing this residual matrix recursively as in SDD, until the residual matrix is sparse enough to be neglected. In this decomposition, a row or a column may contain more than one pattern. However, this formulation does not provide a hierarchical clustering information as PROXIMUS does.

3.5 Computational Complexity

In the alternating iterative heuristic for computing rank-one approximations, each solution to the optimization problem of (3) takes $O(N)$ time. 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 see 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 $O(h \times N)$, where h 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 $h \leq k - 1$. Therefore, we can conclude that the time complexity of overall algorithm is $O(k \times N)$. Note that k is a function of the underlying pattern structure of the input matrix and the prescribed bound on Hamming radius.

4 Application to Association Rule Mining

In this section, we show a simple application of PROXIMUS to accelerate association rule mining, a well-known and extensively studied problem in data mining [1]. Given a set of transactions and a set of items, transactions being subsets of the entire item set, association rule mining aims to discover association rules between itemsets that satisfy the minimum support and confidence constraints prescribed by the user. An association rule is an assertion of kind "{bread, milk} ⇒ {butter}" meaning that if a transaction contains bread and milk, then it is also likely to contain butter. Support of a rule in a transaction set is defined as the fraction of the transactions that contain all items in the rule. Confidence of a rule is the ratio of the number of transactions that contain both sides of the rule to the number of all transactions that contain the left-hand-side of the rule.

Given a transaction set on a set of items, we can construct a binary transaction matrix by mapping transactions to rows and items to columns and setting entry t_{ij} of transaction matrix T to 1 if item j is in transaction T_i. Figs. 3a and 3b illustrate a sample transaction set of six transactions on the item set {beer, snacks, bread, milk, butter} and its corresponding transaction matrix, respectively. A locally optimal rank-one approximation to T is $x_1 y_1^T$ with pattern vector $y_1 = [0\ 0\ 1\ 1\ 1]^T$ and presence vector $x_1 = [0\ 0\ 1\ 1\ 1\ 1]^T$. This

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

(a)

Virtual transaction	Weight
T'_1 : {bread, milk, butter}	4
T'_2 : {beer, snacks, bread}	2

(b)

Fig. 4. (a) Decomposition of transaction matrix of the transaction set in Fig. 3 and (b) the corresponding approximate transaction set.

means that the pattern {bread, milk, butter} is present in transactions T_3, T_4, T_5, and T_6. Based on this pair of approximation vectors, we can create a virtual transaction $T'_1 = \{\mathrm{bread, milk, butter}\}$ that represents all these transactions. Partitioning T with respect to x_1 and finding a locally optimal rank-one approximation to the resulting matrix, we end up with pattern and presence vectors $y_2 = [1\ 1\ 1\ 0\ 0\ 0]^T$ and $x_2 = [1\ 1\ 0\ 0\ 0\ 0]^T$, respectively. Based on these approximation vectors, we can create a second virtual transaction $T'_2 = \{\mathrm{beer, snacks, bread}\}$, which represents transactions T_1 and T_2. We associate weights $w(T'_1) = 4$ and $w(T'_2) = 2$ representing the number of transactions that each virtual transaction represents. Finally, we end up with a transaction set of two transactions that is an approximation to the original transaction set. We can mine this smaller approximate transaction set for association rules on behalf of the original transaction set. This will clearly be faster than mining the original transaction set as the cardinality of the approximate transaction set is one third of the original set. Figs. 4a and 4b show the decomposition of T into two pairs of approximation (presence and pattern) vectors and the resulting approximate transaction set, respectively.

In general, in order to reduce the time required for association rule mining, we decompose the corresponding transaction matrix of the original transaction set and create an approximate transaction set based on the set of identified pattern vectors. We associate a weight with each virtual transaction that is defined as the number of nonzeros in the corresponding presence vector, i.e., the number of transactions that contain the corresponding pattern. We then mine the approximate transaction set. Extension of the association rule mining algorithms to the case of weighted transactions is straightforward; we consider transaction T'_i as occurring $w(T'_i)$ times in the transaction set while counting the frequencies of itemsets. Compression of transaction sets might be particularly useful in data mining applications where data is distributed and sites are loosely coupled or privacy is a concern [27].

5 Experimental Results

In this section, we illustrate the desirable properties of PROXIMUS in terms of effectiveness in clustering and discovering patterns, application to association rule mining, semantic classification of terms and documents, and runtime scalability.

5.1 Effectiveness of Analysis

In this section, we report two experiments that illustrate the superior characteristics of PROXIMUS in approximating and clustering binary data sets 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.

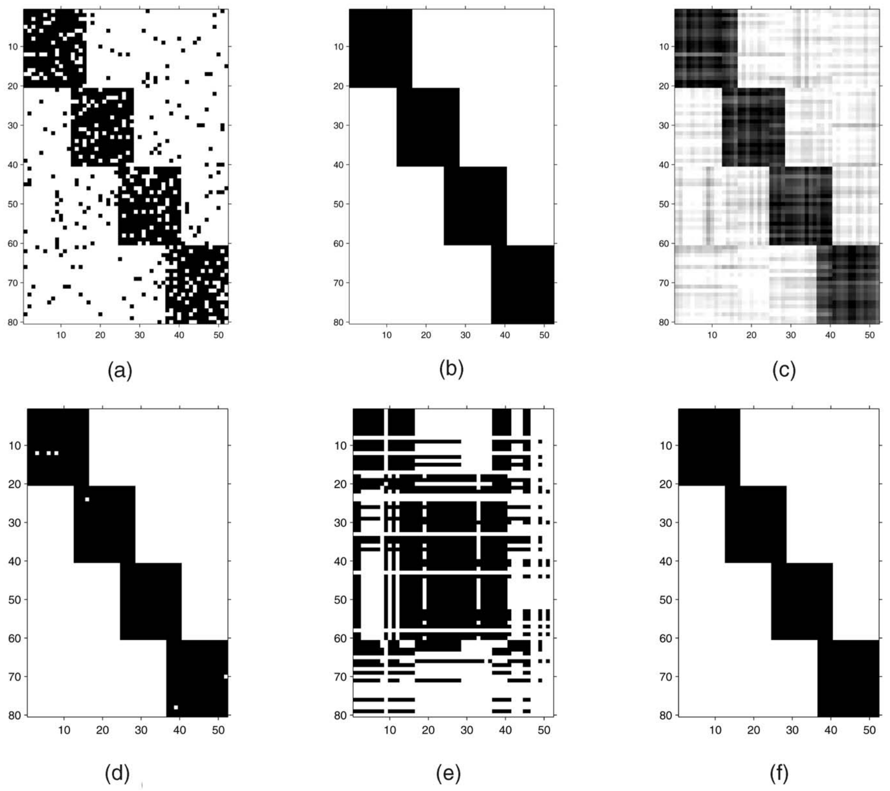
The first matrix that is shown in Fig. 5a contains four overlapping patterns of uniform distribution. This matrix is generated as follows: For the background noise, any entry of the 80×52 matrix is set to 1 with probability p_b. If the ith row contains the kth pattern, then the $(i, j)\mathrm{th}$ entry of the matrix is set to 1 with probability p_p, where

$$(k-1)(l+r)+1 \leq j \leq kl+(k+1)r.$$

Here, l denotes the number of columns that are specific to a single pattern, and r denotes the number of columns shared by two neighboring patterns. While generating the matrix of Fig. 5, pattern length parameters l and r are set to 10 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 of 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 Fig. 5b. 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 input and approximation matrices.

The rank-4 approximation provided by the four most significant singular vectors of SVD is shown in Fig. 5c. 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 before. 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



(a) (b) (c)

(d) (e) (f)

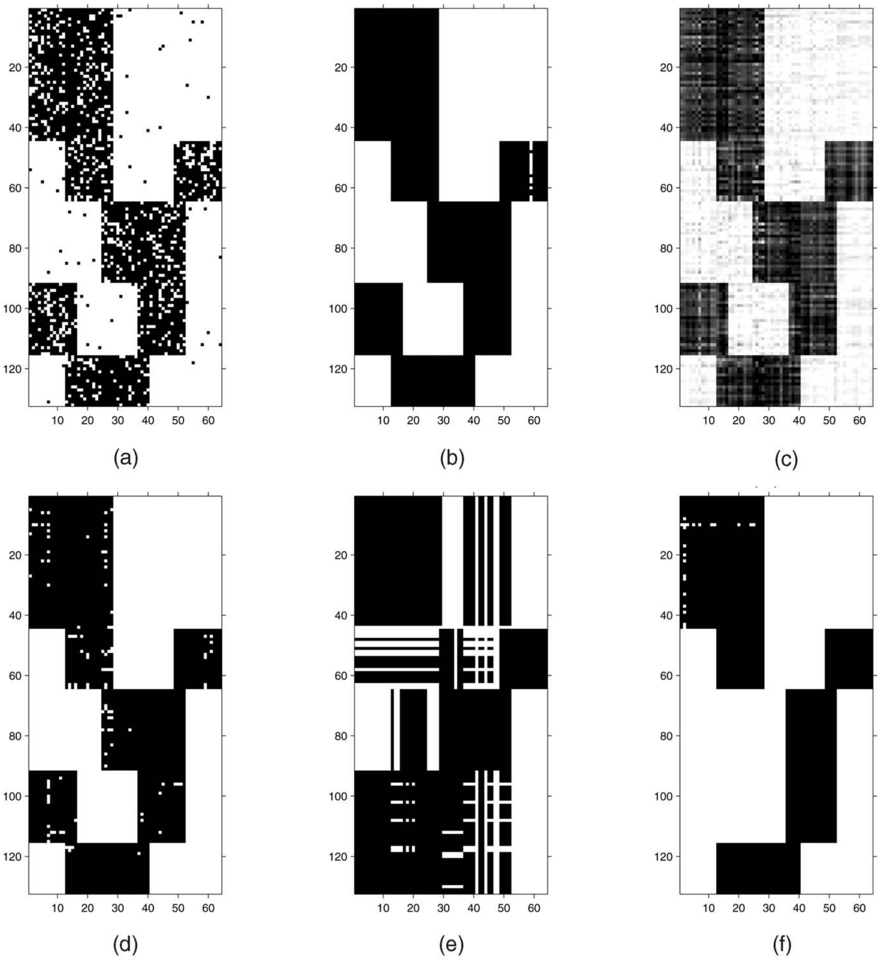
Fig. 5. Approximation of a sample binary matrix that contains four overlapping uniform patterns. (a) Original matrix, (b) rank-4 approximation provided by PROXIMUS, (c) rank-4 approximation provided by SVD, (d) rank-8 approximation obtained by quantizing SVD approximation, (e) approximation (sum of four rank-1 matrices) obtained by quantizing most dominant singular vectors, and (f) rank-4 approximation provided by K-means clustering.

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 Fig. 5d 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 approximation matrix. This makes it possible to represent the approximation as the sum of four rank-one matrices, although the sum may have a larger rank due to loss of orthogonality. 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 the singular vector, accompanied with another negative in the corresponding 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. However, 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 percent worse than that of other techniques. As seen in Fig. 5e, 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 Fig. 5, 4-way K-means clustering provides the same approximation as PROXIMUS, as shown in Fig. 5f. 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 Fig. 6. The 134×64 matrix shown in Fig. 6a consists of 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 ($l = 10$, $r = 4$) 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 are also chosen randomly from a normal distribution.

As seen in Fig. 6b, PROXIMUS is able to provide a rank-6 approximation for this matrix, which reveals the underlying pattern structure reasonably 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



(a) (b) (c)

(d) (e) (f)

Fig. 6. Approximation of a sample binary matrix that contains five row clusters each contain 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 six rank-1 matrices) obtained by quantizing most dominant singular vectors, and (f) rank-6 approximation provided by K-means clustering.

sparsity of some columns in the fifth pattern. On the other hand, as seen in Figs. 6c and 6d, 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 looses track of the original matrix with an error of 68.7, which is shown in Fig. 6e.

The approximation provided by 6-way K-means clustering is shown in Fig. 6f. 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 5-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.

While the computational complexity of SVD is $O(mn \times \min\{m, n\})$ in general, sparse implementations of truncated SVD computations can run in $O(kNI)$ time [7], where k is the number of computed singular vectors and I is the number of iterations in the computation of a single singular vector. Recall that N is the number of nonzeros in the matrix. Similarly, while a general implementation of K-means requires $O(kmnI)$ time, its complexity can be improved to $O(kNI)$ by taking advantage of the sparse and binary nature of the input data sets. Although these algorithms appear to have asymptotically similar time complexity, we note three observations about their runtime performances. First, the factor that relates to the number of approximation vectors or clusters is not k itself in PROXIMUS, rather it is the height of the recursion tree, which is sublinear in most cases. Second, while no fill-in is introduced by PROXIMUS into any submatrix during the computation, SVD may introduce fill-in into the residual matrix. Finally, the number of iterations in PROXIMUS is less than that in the other two methods and floating point operations are completely avoided due to the discrete nature of the algorithm.

5.2 Performance of PROXIMUS in Association Rule Mining

In this section, we illustrate the desirable properties of PROXIMUS in the context of association rule mining using the method described in Section 4. In our implementation, we use the well-known apriori algorithm [1] as the benchmark algorithm for association rule mining. While improved algorithms that reduce the number of passes over

TABLE 1
Description of Data Sets and Results of Preprocessing via PROXIMUS

	#	#	#	# Approximation	Preprocessing
Dataset	Transactions	Items	Non-zeros	vectors	time (s)
connect	67558	129	2904994	6703	1192
pumsb	49047	2113	3629478	4443	1264
pumsb_star	49047	2088	2475997	5416	526

TABLE 2
Time Spent and Number of Discovered Rules in Mining Original and Approximate Transaction Sets

Dataset	Confidence	ARM time	ARM Time	# Rules	# Rules	# Rules
	(%)	orig. (s)	appx. (s)	orig.	appx.	common
connect	50	4766	447	31237901	29342663	28044087
	70	3988	388	25174099	23977423	22545595
	90	3335	333	17297192	17885346	15588014
pumsb	50	3818	317	56412765	56333542	52147969
	70	3187	269	47350093	48920776	44271385
	90	2708	235	36750896	41146376	34796814
pumsb_star	50	4152	329	53468258	50788639	48137472
	70	3315	284	48255192	49015788	44846212
	90	2665	191	38066956	42939526	36234688

data have been developed, these improved algorithms can also be applied to the output of PROXIMUS. We use an efficient implementation of the apriori algorithm[1] [28] for our experiments. We create a second version of the software which is capable of mining weighted transaction sets by slightly modifying the original software. For each data instance, we mine the original transaction set with the original software as well as the approximate transaction set with the modified software and compare the results in terms of both precision and recall rates and the runtime of the software on these two transaction sets.

We evaluate the performance of PROXIMUS in association rule mining on three FIMI workshop data sets.[2] These data sets are described in Table 1. We decompose the matrices corresponding to these data instances using PROXIMUS with $\epsilon = 5$. The resulting number of approximation vectors and the time spent for obtaining this approximation are also shown in the table. As seen in the table, PROXIMUS approximates the transaction set using about one-tenth of the original number of transactions for all three instances.

The results of mining the original and approximate transaction sets for association rules on these three instances are shown in Table 2. We mine these transaction sets for rules of cardinality 6 for a constant support threshold of 20 percent, 20 percent, and 10 percent for data sets connect, pumsb and pumsb_star, respectively. These rule cardinalities and support thresholds are selected large enough to be interesting. While the performance of PROXIMUS for different values of these parameters is generally conserved, the speed-up provided by compressing transaction sets increases with decreasing support threshold and increasing rule size. The table shows the runtime of apriori algorithm on both original and approximate transaction sets along with the number of discovered rules on each transaction set, and the number of rules that are common to these transaction sets for varying confidence threshold. For all three instances, the number of discovered rules is in the order of 10M, and the time spent on mining the original transaction sets is much larger than the time spent for compressing these transaction sets via PROXIMUS.

The performance figures derived from these results are shown in Fig. 7. Each figure displays speed-up, precision, and recall values for varying confidence for all three data sets. Speed-up is calculated as the ratio of the runtime of a priori software on original transaction set to that on approximate transaction set. Precision and recall correspond to the percentage of the rules discovered on both transaction sets among the ones that are discovered on the approximate transaction set and original transaction set, respectively.

As seen in the figure, PROXIMUS provides a speed-up of at least 10 for all data sets for all confidence levels, which is consistent with the rate of compression. While providing this speed-up, PROXIMUS almost always keeps the precision

[1] Borgelt's implementation of the apriori algorithm is available as open source at http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori.html.

[2] FIMI workshop data sets are available at http://fimi.cs.helsinki.fi/data/.

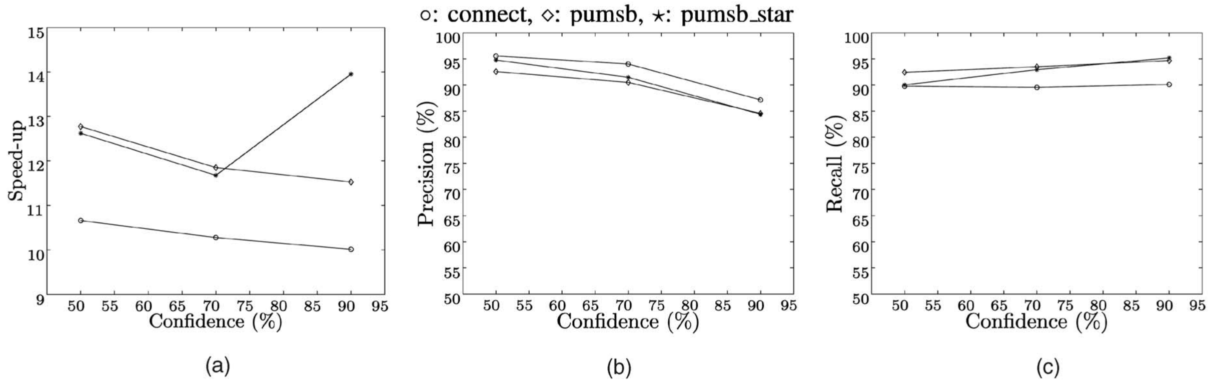


Fig. 7. (a) Speed-up, (b) precision, and (c) recall obtained by performing association rule mining on approximate transaction set for varying confidence threshold.

and recall values above 90 percent. As seen in Figs. 7a and 7b, precision decreases with increasing confidence, while recall shows an opposite trend. This observation is consistent with the fact that PROXIMUS "fills in" the lacking items in all transactions that have the same pattern, while it "filters out" the items that are too rare to be included in the pattern. Therefore, although these rare items can come together to form low-confidence rules, they cannot be discovered for higher confidence thresholds even in the original transaction set. Similarly, by filling in the items for all transactions that belong to a particular pattern, PROXIMUS increases the confidence of the rules that are derived from this pattern. The effects of several other parameters such as bound on Hamming radius (ϵ), initialization scheme for rank-one approximation, rule size, and support threshold are discussed in detail in [29].

It is important to note that meaningful association rules are mined by repeatedly varying confidence and support values until a suitable rule set is determined. This implies that the cost of applying PROXIMUS is amortized over several runs of the apriori algorithm. What is impressive is the fact that, even for a single run, the cost of compression followed by a single a priori run is less than the cost of running a priori on the original data set for all instances in Table 2. It is also important to note that these data sets are all dense. PROXIMUS is specially designed for high-dimensional sparse data sets. Its performance on sparse data sets is even more impressive.

5.3 Semantic Classification of Terms and Documents

In this section, we use PROXIMUS to cluster terms in a document database to extract semantic information, which allows fine grain classification of terms. All experiments in this section are performed on a document database that consists of a collection of articles from the Los Angeles Times newspaper from the late 1980's. The data set consists of 26,799 terms and 3,204 documents, each of which contain a small subset of these terms.

It is possible to analyze the LA Times data set in two different ways. First, we can regard documents as binary vectors in the term space and cluster/classify them based on the intuition that similar documents should have many terms in common. On the other hand, it is also possible to consider terms as binary vectors in the document space and cluster/classify them observing that terms that are semantically similar should occur together in several documents. PROXIMUS provides a framework that allows analyzing both dimensions simultaneously, since each pattern is associated with a pattern and presence vector that characterize row and column spaces, respectively. For example, in the former case, if we represent each document vector as a row of a matrix, presence vectors in the decomposition of this matrix will provide a disjoint clustering of documents while each pattern vector will associate the corresponding cluster with a set of terms that characterize the cluster. Note that a term can be associated with more than one cluster/class in this formulation. This also allows easy semantic interpretation of discovered clusters. Although this formulation is common and very appropriate since distinct document clusters and overlapping term clusters make sense, we consider the later formulation in this paper to illustrate an alternative view point for the analysis of such data sets.

We represent the data set as a binary term-document matrix by mapping terms to rows and columns to documents, so that a nonzero entry in the matrix indicates the existence of a word in the corresponding document. This results in a $26,799 \times 3,204$ term-document matrix that contains 109,946 nonzeros. Observe that the matrix is highly sparse, with each term occurring in about four documents and each document containing about 35 terms on the average. We decompose this matrix via PROXIMUS, setting $\epsilon = 0$. This provides a hierarchical clustering of terms where each leaf cluster is a set of terms that occur exactly in same documents. For the LA Times data set, we obtain a tree with 16,324 leaf clusters. This number is indeed too large for effective analysis, but it is possible to tune the ϵ parameter to obtain a minimum number of clusters with desired quality. However, because of space limitations, we present sample clusters that are chosen from the internal nodes of the perfect ($\epsilon = 0$) hierarchical clustering tree for the purpose of illustration.

A cluster of words discovered by PROXIMUS in LA Times data set is shown in Fig. 8. This cluster is composed of terms *becker*, *bonk*, *bori*, *edberg*, *graf*, *ivan*, *lendl*, *martina*, *mate*, *mecir*, *melbourn*, *miloslav*, *navratilova*, *pam*, *seed*, *semifin*, *shriver*, *stefan*, *steffi*, *sweden*, and *wiland*. This cluster is clearly related to tennis. The pattern vector that corresponds to this cluster is shown at the top of the figure, while the vectors that correspond to the terms in this cluster are shown in the following rows. The LA Times data set also includes categorical information about the documents, where each document is associated with one of six categories. These

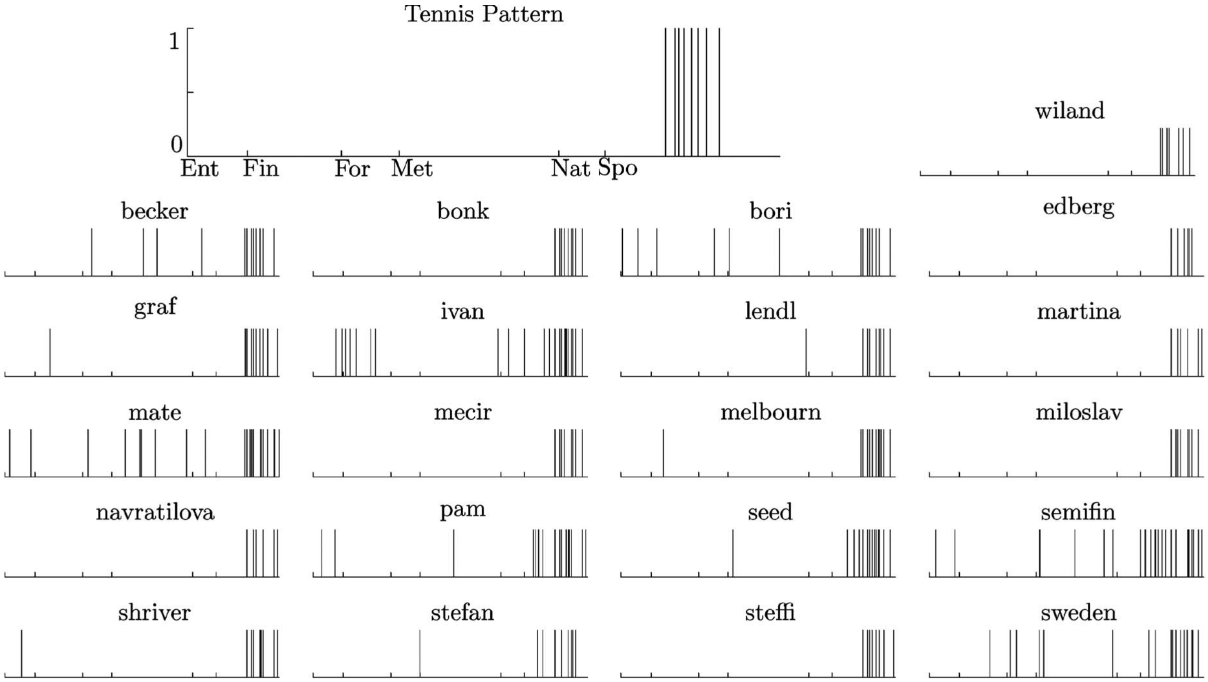


Fig. 8. A cluster of words discovered by PROXIMUS in LA Times data set. Each figure is a binary vector in document space associated with a word, signifying the existence of the word in corresponding documents. Ticks on the x-axis divide the document space into six document classes. The pattern vector associated with this cluster is shown at the top. We associate this cluster with tennis.

categories are *Entertainment*, *Financial*, *Foreign*, *Metro*, *National*, and *Sports*. Note that PROXIMUS does not use this categorical information. In the figures, the x-axis is divided into six regions, where each region corresponds to a category. As seen in the Fig. 8, the term vectors in the cluster discovered by PROXIMUS are generally dense in the *Sports* region and this is captured by the pattern vector provided by PROXIMUS. This pattern vector contains 10 nonzeros, all of which belong to the *Sports* category. These nonzeros correspond to 10 documents, which can clearly be classified as tennis-related documents along with the terms in the cluster. This example illustrates that PROXIMUS is able to provide classification of documents and terms at an adjustable resolution, which is much finer than the available categorical information in this example. Note also that PROXIMUS can also be used for filtering out noise, as in LSI, with an additional advantage of removing the noise completely rather than reducing its magnitude as in the case of orthogonal decompositions like SVD. On the other hand, while SVD-based methods such as LSI can be used for text categorization in order to improve accuracy, PROXIMUS provides a hierarchical clustering associated with directly interpretable pattern vectors.

Other pattern vectors detected by PROXIMUS from the LA Times data set show the same level of accuracy as shown in Table 3. In this table, each cluster is associated with a dominant class, which is the document category that holds the majority in the pattern vector. We also note our interpretation for this cluster, based on the terms in the cluster. Observe that these interpretations provide semantic classification at a finer resolution than the available categorical information, while being consistent with them. Pattern length is the number of documents in pattern vector. As seen in the table, it is easy to interpret these patterns since presence vectors provide discrete sets of terms and pattern vectors provide discrete sets of documents. In addition, the number of documents in the corresponding pattern for each cluster provides a clue about the dominance of the cluster in the data set. Pattern length increases with the depth of the node in the clustering tree as would be expected. Most clusters are associated with at most a couple of documents, while some clusters are more dominant in the data set. Therefore, it is possible to rank clusters of terms to identify popular topics of the time. It is also interesting to note that PROXIMUS captures patterns that are on the border of actual categories. For instance, the dining-related pattern on the fourth row of the table contains three documents that belong to *Metro* and *Entertainment* categories each, which definitely makes sense.

5.4 Runtime Scalability

The results displayed in Fig. 9 demonstrate the scalability of PROXIMUS in terms of number of rows, number of nonzeros, and number of patterns. We generate a series of binary matrices for three settings using the IBM Quest data generator.[3] The settings for these three experiments are as follows:

1. Number of patterns and average number of nonzeros per row are kept constant at 100 and 10, respectively. The number of rows ranges from $\approx 1K$ to $\approx 1M$. Note that number of nonzeros grows linearly with number of rows while number of columns remains constant.
2. The number of rows and number of patterns are kept constant at $\approx 100K$ and 100, respectively, while the average number of nonzeros per row ranges from 5 to 400. Note that the number of nonzeros and the number of columns grow linearly with average row density.

[3] IBM's Quest data generator is available as open source at http://www.almaden.ibm.com/software/quest/Resources/index.shtml.

TABLE 3
Sample Clusters Discovered by PROXIMUS on the LA Times Data Set

	Dominant		Pattern
Terms in cluster	class	Interpretation	length
commod corn crop grain mercantil soybean wheate	Financial	Commodities	14
alysheba anita bred breeder derbi eclips filli jockei mare mccarron santo turf undef whittingham	Sports	Horse racing	7
azing birdi birdie bogei calcavecchia chrysler crenshaw kite lanni lyle mal nabisco par pga wadkin wedge	Sports	Golf	7
bak beef cheese cream dessert dishe menu pasta roast salad sauce steak tomato veget	Metro Entertainment	Dining	7
cambridg chanceri delawar eastman infring kodak patent photographi polaroid shamrock upheld	Financial	Photography	5
schwarzenegg stallon sylvest	Entertainment	Action movies	3

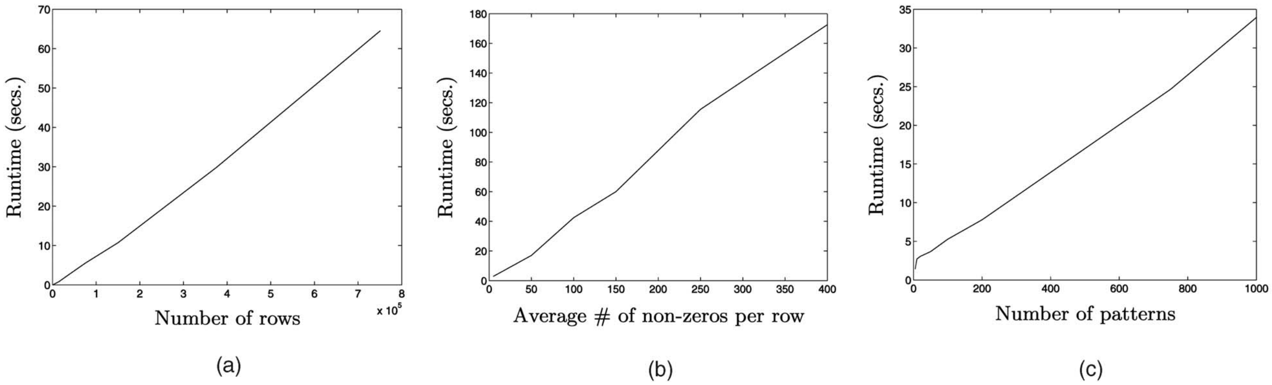


Fig. 9. Runtime of PROXIMUS (secs.) with respect to (a) number of rows, (b) average number of nonzeros per row, and (c) number of patterns.

3. The number of rows and average row density are kept constant at $\approx 100K$ and 10, respectively, while the number of patterns range from 5 to 1,000. Note that the number of columns grows linearly with the number of patterns, while the number of nonzeros remains constant.

All experiments are repeated with different randomly generated matrices 10 times for all values of the varying parameter. The reported values are the average runtimes over these 10 experiments on a Pentium-IV 2.0 GHz workstation with 512 MB RAM. In the first case, the number of nonzeros grows linearly with the number of rows, while the number of patterns is constant. Therefore, we expect the runtime to grow linearly with the number of rows as discussed in Section 3.5. As seen in Fig. 9a, the runtime of PROXIMUS grows linearly with the number of rows. In the second case, we expect runtime to grow linearly with average row density since the number of patterns remains constant, while the number of nonzeros grows linearly. We see this expected behavior of runtime in Fig. 9b. Finally, in the third case, it is important to note that the runtime depends on the number of identified vectors, and not directly on the number of patterns in the matrix. As we expect the number of vectors to be linear in the number of patterns, we expect a linear behavior of runtime with a growing number of patterns since the number of nonzeros remains constant. Fig. 9c shows that the behavior of runtime with respect to the number of patterns is almost linear as expected. Note that, generally, the number of identified vectors is slightly superlinear in terms of the number of underlying patterns.

6 CONCLUSIONS AND ONGOING WORK

In this paper, we have presented a powerful new technique for analysis of large high-dimensional binary valued attribute sets. Using a range of innovative algebraic techniques and data structures, this technique achieves excellent performance and scalability. The application of the method to association rule mining shows that compression of transaction sets via PROXIMUS accelerates the association rule mining process significantly while being able to discover association rules that are consistent with those discovered on the original transaction set. Another sample application on clustering of term-document matrices illustrates that the binary and hierarchical nature of PROXIMUS makes it easy to interpret and annotate the output of decomposition to obtain semantic information. The results reported for these applications show that use of the method is promising in various applications, including dominant and deviant pattern detection, collaborative filtering, clustering, bounded error compression, and classification. The method can also be extended beyond binary attributed data sets to general discrete positive valued attribute sets.

PROXIMUS is available for free download at http://www.cs.purdue.edu/homes/koyuturk/proximus/.

ACKNOWLEDGMENTS

This research was supported in part by NIH Grant R01 GM068959-01. The authors would also like to acknowledge the constructive comments and recommendations of anonymous reviewers, which considerably strengthened this manuscript. A shorter version of this paper appeared in the *Proceedings of the Ninth ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2003).*

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94),* pp. 487-499, 1994.

[2] G.H. John and P. Langley, "Static versus Dynamic Sampling for Data Mining," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (KDD '96),* pp. 367-370, 1996.

[3] F.J. Provost, D. Jensen, and T. Oates, "Efficient Progressive Sampling," *Knowledge Discovery and Data Mining,* pp. 23-32, 1999.

[4] F.J. Provost and V. Kolluri, "A Survey of Methods for Scaling Up Inductive Algorithms," *Data Mining and Knowledge Discovery,* vol. 3, no. 2, pp. 131-169, 1999.

[5] H. Toivonen, "Sampling Large Databases for Association Rules," *Proc. 22th Int'l Conf. Very Large Databases (VLDB '96),* pp. 134-145, 1996.

[6] M.J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara, "Evaluation of Sampling for Data Mining of Association Rules," *Proc. Seventh Int'l Workshop Research Issues Data Eng. (RIDE '97),* p. 42, 1997.

[7] M.W. Berry, S.T. Dumais, and G.W. O'Brien, "Using Linear Algebra for Intelligent Information Retrieval," *SIAM Rev.,* vol. 37, no. 4, pp. 573-595, 1995.

[8] T.G. Kolda and D.P. O'Leary, "Computation and Uses of the Semidiscrete Matrix Decomposition," *ACM Trans. Information Processing,* 1999.

[9] M.T. Chu and R.E. Funderlic, "The Centroid Decomposition: Relationships between Discrete Variational Decompositions and SVDs," *SIAM J. Matrix Analysis and Applications,* vol. 23, no. 4, pp. 1025-1044, 2002.

[10] D. Boley, "Principal Direction Divisive Partitioning," *Data Mining and Knowledge Discovery,* vol. 2, no. 4, pp. 325-344, 1998.

[11] I.T Joliffe, *Principal Component Analysis.* Springer-Verlag, 1986.

[12] H.H. Harman, *Modern Factor Analysis.* Univ. of Chicago Press, 1967.

[13] T. Hofmann, "Probabilistic Latent Semantic Analysis," *Proc. 15th Conf. Uncertainty in Artificial Intelligence (UAI '99),* 1999.

[14] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc. Series B (Methodological),* vol. 39, no. 1, pp. 1-38, 1977.

[15] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis.* John Wiley & Sons, 2001.

[16] R.M. Gray, "Vector Quantization," *IEEE ASSP Magazine,* vol. 1, no. 2, pp. 4-29, 1984.

[17] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. Fifth Berkeley Symp.,* vol. 1, pp. 281-297, 1967.

[18] Z. Huang, "A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining," *Proc. SIGMOD Workshop Research Issues Data Mining and Knowledge Discovery,* 1997.

[19] D. Gibson, J. Kleingberg, and P. Raghavan, "Clustering Categorical Data: An Approach Based on Dynamical Systems," *The VLDB J.,* vol. 8, nos. 3-4, pp. 222-236, 2000.

[20] S. Guha, R. Rastogi, and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes," *Information Systems,* vol. 25, no. 5, pp. 345-366, 2000.

[21] E.H. Han, G. Karypis, V. Kumar, and B. Mobasher, "Hypergraph-Based Clustering in High-Dimensional Data Sets: A Summary of Results," *Bull. IEEE Technical Comm. Data Eng.,* vol. 21, no. 1, pp. 15-22, 1998.

[22] M. Özdal and C. Aykanat, "Hypergraph Models and Algorithms for Data-Pattern Based Clustering," *Data Mining and Knowledge Discovery,* vol. 9, no. 1, pp. 29-57, 2004.

[23] M. Koyutürk, A. Grama, and N. Ramakrishnan, "Algebraic Techniques for Analysis of Large Discrete-Valued Data Sets," *Proc. Sixth European Conf. Principles of Data Mining and Knowledge Discovery (PKDD '02),* pp. 311-324, 2002.

[24] C. Bron and J. Kerbosch, "Finding All Cliques in an Undirected Graph," *Comm. ACM,* vol. 16, pp. 575-577, 1973.

[25] R. Peeters, "The Maximum Edge Biclique Problem is NP-Complete," *Discrete Applied Math.,* vol. 131, no. 3, pp. 651-654, 2003.

[26] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM J. Scientific Computing,* vol. 20, no. 1, pp. 359-392, 1998.

[27] J. Chi, M. Koyutürk, and A. Grama, "Conquest: A Distributed Tool for Constructing Summaries of High-Dimensional Discrete-Attributed Data Sets," *Proc. Fourth SIAM Int'l Conf. Data Mining (SDM '04),* pp. 154-165, 2004.

[28] C. Borgelt and R. Kruse, "Induction of Association Rules: Apriori Implementation," *Proc. 15th Conf. Computational Statistics,* 2002.

[29] M. Koyutürk and A. Grama, "Proximus: A Framework for Analyzing Very High Dimensional Discrete-Attributed Data Sets," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '03),* pp. 147-156, 2003.

Mehmet Koyutürk received the BS degree in 1998 and the MS degree in 2000 from Bilkent University, Turkey, in electrical and electronics engineering and computer engineering, respectively. He has been working toward his PhD degree in the Computer Sciences Department at Purdue University since 2001. His research interests include parallel algorithms, pattern discovery and data mining in molecular biology, and algebraic/graph theoretical algorithms with applications to optimization and data analysis.

Ananth Grama received the PhD degree in computer sciences from the University of Minnesota in 1996. Thereafter, he joined the Department of Computer Sciences at Purdue University, where he is currently a University Faculty Scholar and associate professor. Dr. Grama's research interests span the areas of parallel and distributed computing architectures, algorithms, and applications. His recent work on distributed systems has focused on resource location and allocation mechanisms in peer-to-peer networks. His research on applications has focused on particle dynamics methods, their applications to dense linear system solvers, and algorithms for data compression and analysis. He has authored several papers on these topics and coauthored a textbook *Introduction to Parallel Computing* with Drs. Vipin Kumar, Anshul Gupta, and George Karypis. He is an Us National Science Foundation CAREER awardee, and a member of the American Association for Advancement of Sciences and Sigma Xi.

Naren Ramakrishnan received the PhD degree in computer sciences from Purdue University in Aug 1997. He is an associate professor of computer science at Virginia Tech. His research interests are problem solving environments, mining scientific data, and information personalization. Dr. Ramakrishnan is the recipient of a 2000 US National Science Foundation CAREER grant, the 2001 New Century Technology Council Innovation award, and a corecipient of a 2002 DARPA Early Contributor Appreciation award, towards the BioSPICE project. He currently serves on the editorial board of IEEE *Computer*. He is a member of the IEEE Computer Society.