PROXIMUS: A Framework for Analyzing Very High Dimensional Discrete-Attributed Datasets [*]

Mehmet Koyutürk and Ananth Grama
Department of Computer Sciences, Purdue University
West Lafayette, IN, 47907, USA.
{koyuturk, ayg}@cs.purdue.edu

ABSTRACT

This paper presents an efficient framework for error-bounded compression of high-dimensional discrete attributed datasets. Such datasets, 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 datasets. PROXIMUS provides a technique for reducing large datasets 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 datasets, and performance in terms of capability to represent data in a compact form. We also demonstrate applications of PROXIMUS in association rule mining. In doing so, we establish PROXIMUS as a tool for preprocessing data before applying computationally expensive algorithms or as a tool for directly extracting correlated patterns. Our experimental results show that use of the compressed data for association rule mining provides excellent precision and recall values (near 100%) across a range of support thresholds while reducing the time required for association rule mining drastically.

Keywords

compressing discrete-valued vectors, semi-discrete decomposition, non-orthogonal matrix decompositions.

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 extremely high-dimensional datasets. These datasets may comprise discrete attributes, such as those from business processes, information retrieval, and bio-informatics, 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 datasets. 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 reduction. This paper focuses on algorithms and heuristics for error-bounded compression of very large high-dimensional discrete-attributed datasets.

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, or more.

In order to overcome the computational requirements of the problem while providing efficient analysis of data we propose a new technique – binary($\{0,1\}$) non-orthogonal matrix transformations to extract dominant patterns. In this technique, elements of singular vectors of a discrete, positive valued matrix are constrained to binary entries with an associated singular value of 1. 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 datasets can be summarized.

PROXIMUS is a non-orthogonal matrix transform based on recursive partitioning of a dataset depending on the dis-

[*]The authors acknowledge the support of the National Science Foundation for this work.

tance of a relation from the dominant pattern. The dominant pattern is computed as a binary singular vector of the matrix of relations. PROXIMUS computes only the first singular 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 singular vector, we adopt an iterative alternating heuristic. Due to the discrete nature of the problem, initialization of singular 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 dataset.

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,
- isolating signal from noise in a multi-resolution framework.

We also demonstrate the use of PROXIMUS for preprocessing data for subsequent analysis using conventional techniques. Using the a-priori algorithm [2] for association rule mining we clearly show PROXIMUS' ability to accurately represent data in a highly compact form. Our experimental results show that use of the compressed data for association rule mining provides excellent precision and recall values (near 100%) across a range of support thresholds while reducing the time required for association rule mining drastically.

In the next section, we discuss the use of matrix transforms in the context of data analysis and compression and review existing approaches such as SVD, SDD, Centroid Decompositions and PDDP. 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, present our recursive algorithm for hierarchical discovery of patterns, and discuss implementation issues of these algorithms. In Section 4, we present an application of PROXIMUS in association rule mining. We demonstrate effectiveness of PROXIMUS on both synthetic and experimental data and explore the effect of various parameters on precision, recall and speedup in this application in Section 5. We also illustrate the scalability of PROXIMUS to extremely large datasets. 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 [13, 23, 24, 25]. 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 non-orthogonal matrix transformations such as truncated SVD, SDD, Centroid Decomposition and PDDP have been widely used in information retrieval [3, 4, 6, 15, 16]. In the rest of this section, we summarize commonly used orthogonal and non-orthogonal matrix transformations and their applications in data analysis and explore alternate approaches for binary datasets.

2.1 Rank Reduction and the Singular Value Decomposition (SVD)

SVD forms the basis for Latent Semantic Indexing (LSI) commonly used in information retrieval [3]. LSI is based on the idea of representing data by term-document matrices with entries corresponding to the frequency of occurrence of a term in a particular document, and selecting an appropriate number of singular vectors to represent the data to eliminate noise. SVD transforms a matrix into two orthogonal matrices and a diagonal matrix of the singular values. Specifically, an m by n rectangular matrix A can be decomposed into

$$A = U \Sigma V^T, \tag{1}$$

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. 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 multi-attributed 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 U, and k^{th} column in V represents a pattern in A characterized by σ_k. For larger singular values, the corresponding pattern is more dominant in the dataset. Taking advantage of this property of SVD, LSI summarizes the underlying data represented by matrix A by truncating the SVD of A to an appropriate number of singular values. In doing so, the insignificant patterns corresponding to small singular values are filtered.

2.2 Semi-Discrete Decomposition(SDD)

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 $\{-1, 0, 1\}$ [16]. 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. SDD applied to LSI has been shown to do as well as truncated SVD, while using less than one-tenth the storage [15]. 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 [26]. McConnell and Skillicorn 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 [20].

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 [16] 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 to SVD that is widely used in factor analysis. It has been shown empirically that CD provides a measurement of second order statistical information of the original data [6]. 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. The main difference between SVD and the centroid method is that SVD tends to discover a single dominant pattern while centroid method tends to discover the overall trend of some part of the data, which may be a collection of several independent patterns.

The centroid method runs in time linear in 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 provided that takes advantage of the discrete nature of data and is much sparser.

2.4 Principal Direction Divisive Partitioning (PDDP)

Principal Direction Divisive Partitioning (PDDP) is a hierarchical clustering strategy for high dimensional real-valued sparse datasets [4]. 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 this reason, PROXIMUS is significantly faster than PDDP.

2.5 Other Work on Summarizing Discrete-Attribute Datasets

Other work on summarizing discrete-attributed datasets is largely focused on clustering very large categorical datasets. A class of approaches is based on well-known techniques such as *vector-quantization* [9] and *k-means clustering* [19]. The *k-modes* algorithm [12] 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 [8, 10]. Hypergraph-based approaches observe that discrete-attribute datasets are naturally described by hypergraphs and directly define cost functions on the corresponding hypergraph [11, 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. NON-ORTHOGONAL DECOMPOSITION OF BINARY MATRICES

PROXIMUS is a collection of novel algorithms and data structures that rely on modified SDD to find error-bounded approximations to binary attributed datasets. While relying on the idea of orthogonal matrix transforms, PROXIMUS provides a framework that captures the properties of discrete datasets more accurately and takes advantage of their binary nature to improve both the quality and efficiency of the analysis. Our approach is based on recursively computing discrete rank-one approximations to the matrix to extract dominant patterns hierarchically [17].

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:*

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

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 non-trivial 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).*

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

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

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. Here, the first matrix is the original matrix itself and each subsequent matrix is a residual matrix, *i.e.*, the difference between the given matrix and the matrix produced by sum of previous rank-one approximations. However, the application of SVDs to binary matrices has two drawbacks. First, the resulting decomposition contains non-integral vector values, which is 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, and therefore, SDD also suffers from this problem: if the underlying data consists of non-overlapping (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. Furthermore, 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. A simple approach 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 non-orthogonal 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,
- 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 3.1. **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 problem is closely related to finding maximal cliques in graphs. This problem is known to be NP-hard and there exist no known approximation algorithms or effective heuristics in literature. As a matter of fact, if we view the problem as one of discovering significant patterns in the matrix, the optimal solution is not necessarily the desired rank-one approximation. We illustrate this point with a simple example:

Example 3. *Consider a simple binary matrix A as follows:*

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

Since A is dense enough, the optimal rank-one approximation to A is the outer product of two vectors both consisting of all 1's with error 4. However, there are two visible patterns in this matrix, namely [1 1 1 0] and [0 1 1 1], lying on first two and last two columns, respectively. The rank-one approximation corresponding to the first pattern is $[1\ 1\ 0\ 0] \times [1\ 1\ 1\ 0]^T$ with error 6, which is a local minimum. Therefore, if we were able to compute the optimal rank-one approximation to A, we would not be aware of the two dominant patterns lying in the matrix. This, in fact, is a fortunate property for an NP-hard problem as a heuristic can work well enough to discover a desired local optimum, which does not have to be the globally optimal solution. For this purpose we adapt an alternating iterative heuristic of SDD computation to binary matrices with suitable initialization heuristics.

3.1.1 *Alternating Iterative Heuristic*

Since the objective (error) function can be written as

$$||A - xy^T||_F^2 = ||A||_F^2 - 2x^T Ay + ||x||_2^2 ||y||_2^2, \qquad (3)$$

minimizing the error is equivalent to maximizing

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

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

$$x(i) = \begin{cases} 1, & if\ 2s(i) \geq ||y||_2^2 \\ 0, & otherwise \end{cases} \qquad (5)$$

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 $O(nz(A))$ ($nz(A)$ is the number of non-zeros in matrix A) time and Equation 5 can be evaluated in $O(m)$ time. 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 [16], 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. Indeed, this technique is distantly related to expectation maximization, which is a commonly used technique in statistical analysis [7].

Although the objective function of Equation 4 leads to this linear time algorithm and guarantees convergence to a local maximum, it has a significant drawback due to the discrete nature of the domain. Specifically, this algorithm does not have any global awareness, *i.e*, it always converges to the local maximum closest to initialization. This leaves the task of solving the problem to initialization of the pattern vector. A continuous objective function approximating

$C_d(x,y)$, alleviates this problem. This is more successful in forcing convergence to desired local maxima, especially for very sparse matrices.

3.1.2 Approximate Continuous Objective Function

In the case of decomposing continuous valued matrices, it has been shown [21] that the objective function of rank-one approximation is equivalent to maximizing

$$C_c(x,y) = \frac{(x^TAy)^2}{||x||_2^2||y||_2^2}. \qquad (6)$$

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 maximum at the same point, the behavior of these two functions is highly correlated. Thus, we can use $C_c(x,y)$ as a continuous approximation to $C_d(x,y)$. Fixing y and letting $s = Ay/||y||_2^2$ as above, the objective becomes one of maximizing $\frac{(x^Ts)^2}{||x||_2^2}$. This can be done in linear time by sorting elements of s via counting sort and visiting elements of x in the resulting order until no improvement in the objective function is possible.

This continuous function has the desirable property of having a broader range of convergence compared to the discrete objective function. Furthermore, since the rate of growth of this function declines less rapidly with increasing number of nonzeros in x, it favors discovery of sparser patterns. Although a local maximum of $C_c(x,y)$ does not necessarily correspond to a local maximum of the binary objective function, it may correspond to a point that is close to a local maximum and has a higher objective value than many undesirable local maxima. Note that although this metric provides more flexibility in initialization, selection of the initial pattern vector still has a significant impact on the quality of the solution due to the discrete nature of the domain.

3.2 Recursive Decomposition of Binary Matrices

We use the rank-one approximation of the given 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 3.2. **Partitioning based on rank-one approximation:**
Given rank-one approximation $A \approx xy^T$, 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 $1 \le i \le m$. 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 connected submatrix of A. Therefore, these rows have more similar non-zero structures among each other compared to the rest of the matrix. This partitioning can also be interpreted as creating two new matrices A_0 and A_1. 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.

This partitioning-and-approximation process continues until the matrix cannot be further partitioned or the resulting approximation adequately represents the entire matrix. We define a metric called normalized Hamming radius to measure the adequacy of the representation in terms of the Hamming distances of rows to the underlying pattern vector.

DEFINITION 3.3. **Normalized Hamming distance**
Given two n-dimensional binary vectors x and y, the normalized Hamming distance between x and y is defined as:

$$\hat{h}(x,y) = \frac{||x \; XOR \; y||}{n} = \frac{x^Tx + y^Ty - 2x^Ty}{n},$$

where $||x|| = ||x||_2^2 = ||x||_1$ is the number of nonzeros in a binary vector x.

Normalized Hamming distance measures the fraction of unmatched nonzeros between x and y among all entries of x and y. Note that $0 \le \hat{h}(x,y) \le 1$. The normalized Hamming distance between a row of the matrix and a pattern vector measures the fraction of the row that is not represented by the pattern as well as the fraction of the pattern that does not exist in the row. Thus, normalized Hamming distance provides a measure for detecting mismatched patterns as well as underrepresentation of a row by the underlying pattern.

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

$$\hat{r}(X,y) = \max_{1 \le i \le n} \hat{h}(x,y).$$

We use the normalized 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 one of the following two conditions holds for the rank-one approximation $A_i \approx x_iy_i^T$.

- $\hat{r}(A_{i1}, y_i) < \epsilon$, where ϵ is the prescribed bound on the normalized 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 one of the above conditions holds, the pattern vector y_i is identified as a dominant pattern in matrix A. The resulting approximation for A is represented as $\tilde{A} = UV^T$ where U and V 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.

3.3 Initialization of Iterative Process

While finding a rank-one approximation, initialization is crucial for not only 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 $\Theta(nz(A))$ 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 on 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 [14], this scheme starts with a randomly selected row in one part and grows that part by including the rows that share a nonzero 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.

All of the above initialization schemes require $O(nz(A))$ time. Our observations show that the *Neighbor* scheme tends to initialize the pattern vector close to a desired local minima, *i.e.*, the resulting rank-one 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, we select the dimension that yields the most balanced partition in order to increase the probability of partitioning along a significant dimension.

3.4 Computational Complexity

In the alternating iterative heuristic for computing rank-one approximations, each solution to the optimization problem of Equation 4 takes $O(nz(A))$ 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 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 $O(h \times nz(A))$, 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

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)

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

(b)

Figure 1: (a) A sample transaction set of 6 transactions on 5 items and (b) its corresponding transaction matrix.

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

(a)

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

(b)

Figure 2: (a) Decomposition of transaction matrix of the transaction set of Figure 1 and (b) the corresponding approximate transaction set.

can conclude that the time complexity of overall algorithm is $O(k \times nz(A))$. Note that k is a function of the underlying pattern structure of the input matrix and the prescribed bound on the normalized Hamming radius of the identified clusters.

4. APPLICATION TO ASSOCIATION RULE MINING

In this section, we demonstrate a simple application of PROXIMUS in accelerating association rule mining, a well-known and extensively studied problem in data mining. 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}$\Rightarrow${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 among all transactions in the set. Confidence of a rule is the fraction of the transactions that contain the right-hand-side of the rule among 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. Figure 1(a) and (b) illustrate a sample transaction set of 6 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 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 singular vectors, we can create a virtual transaction T'_1={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 singular vectors, we can create a second virtual transaction T'_2={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. Figure 2(a) and (b) show the decomposition of T into two pairs of singular 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 a-priori algorithm 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.

5. EXPERIMENTAL RESULTS

In this section, we illustrate the desirable properties of PROXIMUS in the context of association rule mining. In our implementation, we use the well-known *a-priori* algorithm [2] as the benchmark algorithm for association rule mining. While improved algorithms that reduce the number of passes over data have been developed, these improved algorithms can also be applied to the output of PROXIMUS. In order to illustrate the effectiveness of PROXIMUS in association rule mining and to explore the effects of parameters such as size of the transaction set, number of underlying patterns in the data, and bound on normalized Hamming radius, we test our framework on synthetically generated association data using the data generator made available by the IBM Quest Research Group [1]. We also test our method on the *Agaricus Lepiota* dataset in order to illustrate the effectiveness of PROXIMUS on real data. We use an efficient implementation of the *a-priori* algorithm which is available as open source [5] for our experiments. We create a second version of the software which is capable of mining

Dataset	# trans.	# items	# 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

Table 1: Description of generated data in terms number of transactions, items and patterns.

min. sup. %	time orig. sec.	time comp. sec.	rules orig. #	rules comp. #	rules match. #	prec. %	recall %
0.5	13.76	1.06	332395	333046	331087	99.6	99.8
1.0	8.01	0.39	138745	139519	137200	98.3	99.0
1.5	5.12	0.18	60500	59580	59580	100.0	98.5
2.0	3.14	0.11	36061	39117	35366	90.4	98.1
2.5	0.86	0.03	14750	14645	14645	100.0	99.3
3.0	0.68	0.02	11180	11070	11070	100.0	99.0
3.5	0.67	0.02	11075	10965	10965	100.0	99.0
4.0	0.51	0.02	7005	10965	7005	63.9	100.0
4.5	0.50	0.02	7005	6900	6900	100.0	98.5
5.0	0.48	0.01	6900	6270	6270	100.0	90.9
5.5	0.31	0.01	3960	3960	3960	100.0	100.0
6.0	0.30	0.01	3960	3960	3960	100.0	100.0

Table 2: Performance of PROXIMUS on M100K dataset.

weighted transaction sets by slightly modifying the original software. In each instance of the generated data, we mine the given 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 have made the source code of PROXIMUS available for free download at
`http://www.cs.purdue.edu/homes/koyuturk/proximus/`.

We generate two sets of data, one for varying number of transactions and the other for varying 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) transactions, respectively, are generated. In the second set, the number of transactions is fixed to $\approx 100K$ (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 items per transaction and the average number of items in a pattern are set to 10 while generating the data. The average correlation between each pair of patterns and the average confidence of a rule are set to 0.1 and 90% respectively. Table 1 shows the general description of the five instances in terms of number of transactions, number of items, and number of patterns.

Table 2 shows the comparison of rules obtained by running *a-priori* on original and approximate transaction sets for support values ranging from 0.5% to 6.0%, which is selected as a meaningful range of support values for this dataset. The approximate data is obtained by decomposing the M100K matrix with $\epsilon = 0.01$ resulting in 2479 patterns or equivalently, virtual transactions. The preprocessing time for this dataset is 5.89 seconds. The rules displayed in the table are the rules of cardinality 5, which is selected to be large enough for transactions that contain 10 items on the average. The performance of PROXIMUS for different rule cardinalities is further analyzed in Section 5.2. The table displays the run time of *a-priori* on original and approximate transaction sets, the number of rules discovered on original and approximate transaction sets, number of rules that are discovered on both transaction sets (matching rules), and the precision and recall rates for sample support values in the range. *Precision* is defined as the number of matching rules over all rules that are discovered on the approximate transaction set, measuring how precise are the results that are obtained on the approximate set. *Recall* is defined as the fraction of the rules discovered on the original transaction set that are also discovered on the approximate set, measuring how successful compression is in recalling the rules that are present in the original data.

As seen in Table 2, the precision and recall values are close to 100% for almost all support values while the runtime of *a-priori* is significantly smaller on the approximate set compared to that on the original transaction set; the speedup is observed to be as high as two orders of magnitude for some support values! Note that the same 3960 rules are discovered on both transaction sets for support values up to 9%, so the precision and recall values are all 100% in this range, which are not displayed in the table. Notice also the glitch in precision at support around 4%. This phenomenon is sometimes observed for all datasets for both precision and recall. It is caused by the fact that a group of rules has the same support value for all rules in the group in both original and approximate transaction sets. Although the approximate set represents the original set accurately enough, so that such groups are the same on both transaction sets as can be seen on the table, there is a small discrepancy between the support of each particular group of rules on original and approximate transaction sets. For example, as we derived from our more detailed results, a group of about 4000 rules have support about $(4 + \delta)\%$ on the original set while having support about $(4 - \delta)\%$ on the approximate set, δ being small. Thus, this group of rules are discovered on the approximate transaction set for a support threshold of 4%, but have not yet been discovered on the original transaction set, making the observed precision lower for a small interval.

5.1 Effect of Number of Transactions and Patterns

Evaluation of PROXIMUS' performance on varying number of transactions is displayed in Table 3 for approximately 10K, 100K and 1M transactions. The table shows the average values of speedup, precision and recall over 100 uniform sample support values on the meaningful support range of each instance. Here, meaningful support range refers to an interval that has lower-bound that results in less than 500K rules (set to ignore the rules that are not interesting) and upper-bound above which *a-priori* is not able to discover any rules. Based on this principle, the meaningful range is set to [0.5%,9.0%] for all three datasets. The results of preprocessing the transaction sets via PROXIMUS, namely the preprocessing time and number of singular vectors are also displayed in the table. Based on these results, we can conclude that the benefit of compressing transaction sets increases with increasing number of transactions, if the number of underlying patterns does not change. Although the cost of preprocessing also grows with increasing number of transactions, the increase in the speedup makes it especially suitable for very large datasets since preprocessing is done only once while the set might be mined several times for association rules. Note also that the runtime of *a-priori* on

data set	prep. time sec.	sing. vecs. #	speedup	prec. %	recall %
M10K	0.51	870	6.1	96.1	98.9
M100K	5.89	2479	30.4	95.2	97.8
M1M	43.88	4460	209.8	95.9	95.8

Table 3: Effect of number of transactions on the performance of PROXIMUS.

data set	prep. time sec.	sing. vecs. #	speedup	prec. %	recall %
L100K	3.15	480	112.3	95.6	99.1
M100K	5.89	2479	30.4	95.2	97.8
H100K	28.8	7652	8.9	95.6	98.8

Table 4: Effect of number of patterns on the performance of PROXIMUS.

the original transaction set for lower support thresholds in the meaningful support range is significantly higher than the preprocessing time for all datasets. In addition, the precision and recall values are not significantly affected by the number of transactions.

Table 4 shows the performance of PROXIMUS on transaction sets containing 20, 100 and 500 patterns based on the same experimental setup discussed above. The meaningful support ranges are set to [1.0%,10.0%], [0.5%, 9.0%] and [0.3%, 1.5%] for datasets L100K, M100K, and H100K, respectively. As seen in the table, increasing the number of patterns causes PROXIMUS to approximate the matrix with a higher number of singular vectors increasing the preprocessing time, as expected. Thus, increasing number of patterns reduces the speedup as shown in the table. Also note that PROXIMUS attains over 95% average precision and recall rates for all datasets.

5.2 Effect of Other Parameters

We have also explored the effect of the bound on normalized Hamming radius (ϵ) on the performance of PROXIMUS. Our experimental results confirm the trade-off between obtaining higher speedups from decreasing the number of singular vectors by increasing ϵ and obtaining higher precision and recall values from improving the quality of compression by reducing ϵ. We observe that this trade-off is more apparent for harder problem instances (*e.g.*, datasets with larger number of patterns). We adjust ϵ properly based on the number of transaction sets and patterns to obtain consistent approximation for all datasets, in order to evaluate the effect of other parameters fairly.

Finally, we discuss the performance of PROXIMUS for varying levels of itemset cardinality. As the average cardinality of patterns is set to 10, rules of interest are expected to contain less than 10 items. Therefore, we compare the performance of PROXIMUS on M100K dataset for rules of cardinality between 2 and 10. Both precision and recall are higher for rules with larger number of items, generally. The benefit of compression becomes more apparent with increasing rule size since speedup also increases with increasing rule size in addition to precision and recall.

5.3 Performance of PROXIMUS on Real Data

In order to illustrate the effectiveness of PROXIMUS on real data, we have also conducted experiments on the *Agaricus Lepiota* dataset that includes descriptions of samples corresponding to 8124 species of gilled mushrooms in the

min. sup. %	time orig. sec.	time comp. sec.	rules orig. #	rules comp. #	rules match. #	prec. %	recall %
10.0	46.49	7.49	497813	464546	464185	99.9	93.2
12.4	21.81	3.88	66695	66363	66191	99.7	99.2
14.8	14.98	2.19	63413	63299	63210	99.9	99.7
17.2	9.77	1.63	32691	32647	32647	100.0	99.9
19.6	7.43	1.30	32593	32589	32589	100.0	99.9
21.2	6.83	1.23	32593	32589	32589	100.0	99.9
24.4	1.61	0.28	147	156	147	94.2	100.0
26.8	1.01	0.17	108	108	108	100.0	100.0
29.2	0.70	0.11	26	26	26	100.0	100.0
31.6	0.56	0.09	2	2	2	100.0	100.0

Table 5: Performance of PROXIMUS on *Agaricus Lepiota* dataset.

Agaricus and Lepiota family [18]. 23 categorical attributes are associated with each species. We mine this dataset for rules of kind "mushrooms with bell-shaped and fibrous caps have brown gills" by defining 118 items corresponding to categories of the 23 attributes. This gives us a binary dataset with 8124 transactions on 118 items, with all transaction sizes being at most 23 (one category has missing values). We represent this transaction set by a 8124×118 binary matrix with 184372 non-zero entries.

We decompose the *Agaricus Lepiota* matrix with $\epsilon = 0.05$ getting an approximation consisting of 1142 patterns, which means compressing the transaction set to nearly one seventh of its original size. The preprocessing time for this dataset is 15.73 seconds. Notice that this figure is about 50 times the running time of PROXIMUS on M10K data. This unexpected discrepancy between the running times is due to the fact that the complexity of PROXIMUS is proportional to the height of the recursion tree; number of discovered patterns is only an upper-bound on this number. Interestingly, the *Agaricus Lepiota* dataset almost attains this upper-bound as the recursion tree turns out to be totally imbalanced for this particular dataset. In addition, high variance in pattern structure degrades the rate of convergence of the iterative algorithm, increasing the execution time. Therefore, we can conclude that the running time of PROXIMUS is mostly determined by the structure of the data rather than its size.

Table 5 shows the comparison of rules obtained by running *a-priori* on original and approximate data for support values ranging from 10.0% to 31.6%, which is selected as a meaningful range of support values for this dataset. In the experiments reported here, we are looking for rules of cardinality 10, which is reasonable since each mushroom is characterized by 23 categorical attributes. Note that the results are similar for different rule sizes. As seen in the table, the speed-up achieved by compressing the transaction set is about 7 for all support values, which is roughly equal to the compression factor. Both precision and recall rates are above 90% for all of the support values (being above 99% for most), which is outstanding for this value of speedup. Note that the phenomenon of glitches in precision and recall values is not observed on the *Agaricus Lepiota* dataset, which is as expected in real-life applications.

5.4 Runtime Scalability

The results displayed in Figure 3 demonstrate the scalability of PROXIMUS in terms of number of rows, number of nonzeros and number of patterns. The settings for these three experiments are as follows:

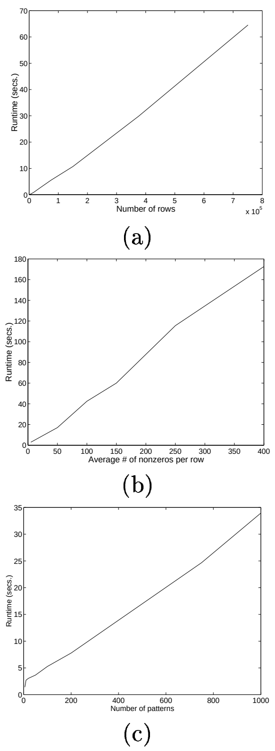

Figure 3: Runtime of PROXIMUS (secs.) with respect to (a) number of rows (b) average number of nonzeros per row (c)number of patterns.

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 approximately 1K to 1M. Note that number of nonzeros grows linearly with number of rows while number of columns remains constant.

2. Number of rows and number of patterns are kept constant at approximately 100K and 100 respectively. Average number of nonzeros per row ranges from 5 to 400. Note that number of nonzeros and number of columns grow linearly with average row density.

3. Number of rows and average row density are kept constant at approximately 100K and 10 respectively. Number of patterns range from 5 to 1000. Note that number of columns grows linearly with number of patterns while 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 server with 512 MB RAM. In the first case, the number of nonzeros grows linearly with number of rows while the number of patterns is constant. Therefore, we expect the runtime to grow linearly with number of rows as discussed in Section 3.4. As seen in Figure 3(a), the runtime of PROXIMUS grows linearly with 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 number of nonzeros grows linearly. We see this expected behavior of run time in Figure 3(b). 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 number of vectors to be linear in number of patterns, we expect a linear behavior of runtime with growing number of patterns since the number of nonzeros remains constant. Figure 3(c) shows that the behavior of runtime with respect to 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 algebraic techniques and data structures, this technique achieves excellent performance and scalability. The application of the method to association rule mining showed 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. The results reported for this particular application 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 datasets to general discrete positive valued attribute sets.

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

7. REFERENCES

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

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, 1994.

[3] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, 1995.

[4] D. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

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

[6] M. T. Chu and R. E. Funderlic. The centroid decomposition: Relationships between discrete variational decompositions and SVDs. *SIAM J. Matrix Anal. Appl.*, 23(4):1025–1044, 2002.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39 (1):1–38, 1977.

[8] D. Gibson, J. Kleingberg, and P. Raghavan. Clustering categorical data: An approach based on dynamical systems. In *Proc. 24th VLDB Conf.*, 1998.

[9] R. M. Gray. Vector quantization. *IEEE ASSP*, 1(2):4–29, 1984.

[10] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[11] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Hypergraph-based clustering in high-dimensional datasets: A summary of results. *Bulletin of the Technical Committee on Data Engineering*, 21(1), 1998.

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

[13] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In E. Simoudis, J. Han, and U. M. Fayyad, editors, *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining, KDD*, pages 367–370. AAAI Press, 2–4 1996.

[14] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[15] T. G. Kolda and D. P. O'Leary. A semidiscrete matrix decomposition for latent semantic indexing information retrieval. *ACM Transactions on Information Systems*, 16(4):322–346, 1998.

[16] T. G. Kolda and D. P. O'Leary. Computation and uses of the semidiscrete matrix decomposition. *ACM Transactions on Information Processing*, 1999.

[17] M. Koyutürk, A. Grama, and N. Ramakrishnan. Algebraic techniques for analysis of large discrete-valued datasets. In *Proc. 6th European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD 2002)*, pages 311–324, 2002.

[18] G. H. Lincoff. *Mushroom Records Drawn From The Audubon Society Field Guide to North American Mushrooms.* Alfred A. Knopf, New York, 1981.

[19] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium*, volume 1, pages 281–297, 1967.

[20] S. McConnell and D. B. Skillicorn. Outlier detection using semi-discrete decomposition. Technical Report 2001-452, Dept. of Computing and Information Science, Queen's University, 2001.

[21] D. P. O'Leary and S. Peleg. Digital image compression by outer product expansion. *IEEE Trans. on Communications*, 31:441–444, 1983.

[22] M. Özdal and C. Aykanat. Clustering based on data patterns using hypergraph models. *to be published in Data Mining and Knowledge Discovery*, 2003.

[23] F. J. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3(2):131–169, 1999.

[24] H. Toivonen. Sampling large databases for association rules. In 22^{th} *Intl. Conf. on Very Large Databases (VLDB'96)*, pages 134–145, 1996.

[25] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. Technical Report TR617, 1996.

[26] S. Zyto, A. Grama, and W. Szpankowski. Semi-discrete matrix transforms (SDD) for image and video compression. In D. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, pages 249–259. Kluwer, 2002.