CONQUEST: A Distributed Tool for Constructing Summaries of High-Dimensional Discrete Attributed Datasets

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

Abstract

The problem of constructing bounded-error summaries of binary attributed data of very high dimensions is an important and difficult one. These summaries enable more expensive analysis techniques to be applied efficiently with little loss in accuracy. Recent work in this area has resulted in the use of discrete linear algebraic transforms to construct such summaries efficiently. This paper addresses the problem of constructing summaries of distributed datasets. Specifically, the problem can be stated as follows: given a set of n discrete attributed vectors distributed across p sites, construct a summary of $k << n$ vectors such that each of the input vectors is within given bounded distance from some output vector. In addition to being algorithmically efficient (i.e., must do no more work than corresponding serial algorithm), the distributed formulation must have low parallelization overheads. We present here, CONQUEST, a tool that achieves excellent performance and scalability for summarizing distributed datasets. In contrast to traditional parallel techniques that distribute the kernel operations, CONQUEST uses a less aggressive parallel formulation that relies on the principle of sampling to reduce communication overhead while maintaining high accuracy. Specifically, each individual site computes its local patterns independently. Various sites cooperate within dynamically orchestrated work-groups to construct consensus patters from these local patterns. Individual sites then decide to participate in the consensus or leave the group. Experimental results on a set of Intel Xeon servers demonstrate that this strategy is capable of excellent performance in terms of compression time, ratio, and accuracy with respect to post-processing tasks. The communication overhead associated with CONQUEST is also shown to be minimal, making it ideally suited to wide-area deployment.

keywords: distributed data mining, compressing binary attributed vectors, non-orthogonal matrix decompositions, correlations in high dimensions.

1 Introduction

The tremendous increase in recent years in organizations' ability to acquire and store data has resulted in extremely large, high dimensional datasets. For example, commonly used Wal-Mart sales data is in the order of terabytes, with each transaction typically defined over a space of several thousand dimensions (items). This paper focuses on efficient distributed techniques for analysis of such discrete attributed datasets. Analysis of discrete datasets presents significant challenges since it generally leads to NP-hard problems. Consequently, algorithms and heuristics for such problems rely heavily on principles of sub-sampling and compression for reducing the volume of data these algorithms must examine.

While serial techniques for sub-sampling and compression have been developed and applied with some success [15, 24, 32, 36], a variety of application characteristics necessitate the development of corresponding distributed formulations. These include:

- *Data volume:* Datasets often reside at geographically distributed locations. Collecting all of the data at a single location is infeasible because of network bandwidth and storage requirements.

- *Real-time response:* Certain applications in data analysis, such as network intrusion detection, require real time response from a number of different locations. Collecting data for analysis and disseminating results of analysis may be too time consuming for such applications.

- *Privacy considerations:* In other applications, privacy considerations might preclude collecting data at a single site. Depending on privacy requirements, only aggregate patterns may be communicated. CONQUEST allows precisely such privacy policies.

CONQUEST is based on the serial program PROX-

IMUS[1], which uses a variant of Semi-Discrete Matrix Decomposition (SDD) to efficiently compress binary datasets in an error bounded fashion. CONQUEST uses a parallel formulation that relies on the principle of sampling to reduce communication overhead while maintaining high accuracy. Specifically, each individual site computes its local patterns independently. Various sites cooperate within dynamically orchestrated work-groups to construct consensus patterns from these local patterns. Then, individual sites may decide to participate in the consensus or leave the group. We demonstrate that this strategy results in excellent parallel performance.

An important optimization criterion for the serial formulation, PROXIMUS, is the number of patterns extracted for a given error bound. A smaller number of patterns is desirable since it corresponds to better compression for a given distortion bound. Since the serial and parallel formulations use different algorithms, an important consideration is the effect of our parallelization strategy on the quality of the output, i.e., the number of patterns extracted. We show experimentally that in addition to excellent parallel performance, CONQUEST introduces little overhead in terms of number of patterns detected.

The rest of the paper is organized as follows: In Section 2, we discuss previous research related to CONQUEST's serial and parallel formulations. Section 3 introduces PROXIMUS briefly. In Section 4, we discuss the challenges associated with the parallel formulation and explain our design decisions. In Section 5, we evaluate the performance of CONQUEST on a set of 8 Intel Xeon servers on a range of inputs. We also discuss the application of CONQUEST in the context of association rule mining. Finally, in Section 6, we draw conclusions and outline avenues for future research.

2 Related Work

In this section, we first explore related work on analyzing binary datasets, followed by parallel formulations of these methods. Data reduction techniques typically take the form of probabilistic sub-sampling or data compression. Techniques based on probabilistic sub-sampling have been extensively explored [15, 32, 36]. Use of data compression techniques relies on extracting compact representations for data through discovery of dominant patterns. A natural way of compressing data relies on matrix transforms such as truncated Singular Value Decompositions (SVD), Semi-Discrete Decomposition (SDD), and Centroid Decomposition. These methods have been widely used in information retrieval [4, 9, 22, 23]. SVD decomposes a matrix into two orthogonal matrices, which contain the dominant patterns. Each pattern is represented by a pair of singular vectors and an associated singular value, which identifies the strength of the corresponding pattern in the matrix. Computation of a full SVD can be computationally expensive. SDD provides a convenient, and often faster approximation to SVD by limiting the entries of the singular vectors to the set $\{-1,0,1\}$. Centroid Decomposition represents the underlying matrix in terms of centroid factors that can be computed without knowledge of the entire matrix with the help of a fast heuristic named Centroid Method. The computation of a centroid decomposition only depends on the correlations between the rows of the matrix. The main difference between SVD and the centroid method is that centroid method tends to discover a single dominant pattern while the SVD tends to discover the overall trend of the data. This may be a collection of several independent patterns. Orthogonal matrix decompositions have been exploited by several distributed data mining algorithms as well [18, 30].

A major problem associated with orthogonal decompositions for large scale binary data analysis is that the forced orthogonality of discovered patterns degrades the interpretability of the analysis (e.g., what is the physical interpretation of a negative number in binary data?). A variant of these methods, Principal Direction Divisive Partitioning (PDDP) [5], addresses this problem by recursively finding rank-one approximations of the input matrix and partitioning this matrix based on the approximation. All of these methods target the analysis of high-dimensional data of continuous nature. PROXIMUS adapts the idea of recursive matrix decomposition to the analysis of large-scale binary-valued datasets.

Prior work on parallel data mining algorithms has focused on tasks such as clustering, classification, and association rule mining. Several projects have addressed the parallelization of existing clustering algorithms [10, 12, 17, 29]. Among these, CLARA[20] attempts to create multiple samples and applies the serial algorithm PMD on each sample to achieve efficiency of analysis on relatively large datasets. However, this efficiency is achieved at the expense of a possibility (probabilistically small) of missing clusters in the data not sampled. Several researchers have developed parallel association rule mining algorithms for various platforms [2, 8, 14, 27, 30, 31, 34, 37, 38]. Most of these approaches are based on the *a-priori* algorithm [3] and its variants. One class of algorithms is based on aggressive parallel formulations that focus on partitioning the data elements (*e.g.*, candidate itemsets) so that each site performs an independent part of the task, which is well suited to massively parallel platforms. Another class is based on computing fre-

[1] PROXIMUS is available over the public domain at `http://www.cs.purdue.edu/homes/koyuturk/proximus/`

quent itemsets at each site individually and then working in parallel to join individual patterns into global association rules. This provides a more suitable framework for distributed systems. Work on parallel classification has resulted in systems such as SPRINT [33], ScalParC [16], and others [35]. These systems typically use decision tree based classification.

In comparison to the parallel techniques mentioned above, CONQUEST is based on a fundamentally different serial algorithm, PROXIMUS, which provides a more flexible formulation for discrete data analysis based on the principle of data reduction. Instead of analyzing a large dataset, PROXIMUS attempts to reduce the volume of data that any subsequent analysis task would have to deal with. Possible subsequent analyses include classification, clustering, pattern discovery and association rule mining. As the solution of such problems in distributed platforms on large scale data tends to be hard and expensive, it is possible to exploit the flexibility of PROXIMUS to simplify the problem for the underlying application. Based on this observation, CONQUEST adopts a parallel formulation that draws upon principles of sub-sampling to yield excellent parallel performance, while preserving the quality of the output.

3 PROXIMUS: An Algebraic Framework for Error Bounded Compression of Binary Datasets

PROXIMUS [24, 25] 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 non-orthogonal matrix transforms, PROXIMUS provides a framework for capturing the properties of binary datasets more accurately while taking advantage of their binary nature to improve both the quality and efficiency of analysis. PROXIMUS is based on recursively computing discrete rank-one approximations of a 0-1 matrix to extract dominant patterns hierarchically.

A rank-one approximation to a binary matrix A with m rows and n columns is an outer product of two binary vectors x (presence vector) and y (pattern vector) that is at minimum Hamming distance from A. Here x and y are of dimensions m and n, respectively. The problem of finding a rank-one approximation, *i.e.* finding binary vectors x and y that minimize $||A - xy^T||_F^2 = |\{a_{ij} \in (A - xy^T) : |a_{ij}| = 1\}|$ is NP-hard. PROXIMUS adapts the alternating iterative heuristic of SDD to the binary domain as a fast and effective technique that is ideally suited to the discrete nature of the data.

For decomposing continuous valued matrices, it was shown in [28] that the objective function of rank-one approximation is equivalent to one of maximizing:

$$C_c(x, y) = \frac{(x^T Ay)^2}{||x||_2^2 ||y||_2^2}. \qquad (3.1)$$

PROXIMUS approximates the objective function of discrete rank-one approximation with this function and applies an alternating iterative heuristic based on this function[2]. Fixing y and letting $s = Ay/||y||_2^2$, the objective becomes one of maximizing $\frac{(x^T s)^2}{||x||_2^2}$. 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. The same algorithm can be used to solve for y for a fixed x. Thus, we can iteratively apply this strategy by choosing an initial y, solving for x, fixing x, solving for y, and so on, until no improvement is possible. This heuristic finds a locally optimal solution to the rank-one approximation problem in time linear in the number of non-zeros in the matrix. This is because the fundamental operations performed in the algorithm are sparse matrix-vector multiplications and counting sorts. Note that the number of iterations is bounded by number of columns (rows) and generally a few iterations are sufficient for convergence in practice [26].

PROXIMUS uses the rank-one approximation of the given matrix to partition the rows into two sub-matrices A_1 and A_0 containing rows that correspond to the ones and zeros of the presence vector x, respectively. Therefore, the rows in A_1 have a greater degree of similarity with respect to their non-zero structure among themselves (characterized by the pattern vector y) compared to the rest of the matrix. Since the rank-one approximation of A yields no information about A_0, we further compute a rank-one approximation for A_0 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 to determine whether this representation is adequate as determined by 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

[2]Minimizing the error is equivalent to maximizing the (discrete) function $C_d(x, y) = 2x^T Ay - ||x||_2^2 ||y||_2^2$, which also allows a linear-time alternating iterative heuristic. The algorithm that is based on this discrete objective function is implemented in PROXIMUS as well. We base our discussion on the continuous approximation throughout this paper as the parallel implementation is based on this approximation, but the discussion on algorithms and possible parallelization schemes also applies to the discrete function. The differences between the two objective functions are discussed in [26].

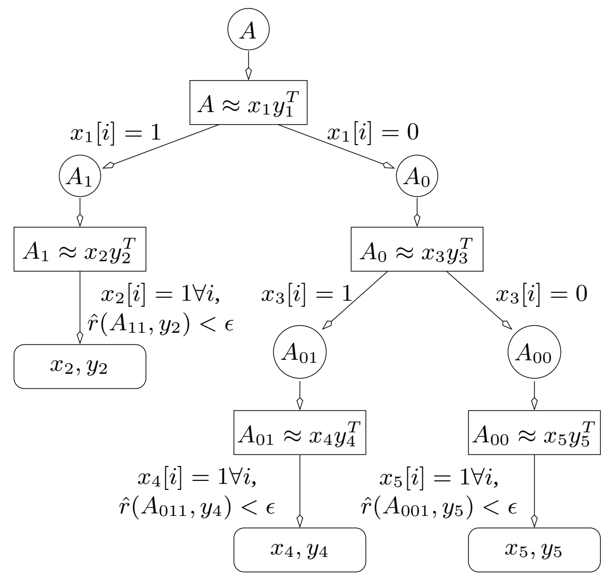

Figure 1: Recursive structure of PROXIMUS. Leaves of the recursion tree correspond to final decomposition.

resulting approximation adequately represents the entire matrix. PROXIMUS uses a metric called normalized Hamming radius to measure the adequacy of the representation, which is defined as the maximum normalized Hamming distance of all rows of the matrix that are present in the approximation to the pattern vector. The normalized Hamming distance between two binary vectors is the ratio of Hamming distance to the size of the vectors. Note that this metric only takes values in the range $[0, 1]$.

The recursive algorithm does not partition sub-matrix A_i further if both of the following conditions hold for the rank-one approximation $A_i \approx x_i y_i^T$.

- $\hat{r}(A_{i1}, y_i) < \epsilon$, 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 both of the above conditions hold, the pattern vector y_i is identified as a dominant pattern in matrix 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.

Figure 1 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_1. The rank-one approximation to A_1 returns a presence vector of all 1's and the approximation is adequate so the recursion

```
1. Loop until no improvement on $C_c(x, y)$ is possible
2.        $s \leftarrow Ay/||y||_2^2$
3.        Solve for $x$ to maximize $\frac{(x^T s)^2}{||x||_2^2}$
4.        $s \leftarrow A^T x/||x||_2^2$
5.        Solve for $y$ to maximize $\frac{(y^T s)^2}{||y||_2^2}$
6. End loop
```

Figure 2: Outline of the alternating iterative heuristic for rank-one approximation.

stops at that node and y_2 is recorded as a dominant pattern. On the other hand, matrix A_0 is further partitioned as the approximation $A_0 \approx x_3 y_3^T$ does not cover all rows of A_0. The overall decomposition is $A \approx XY^T$ where $X = [x_2, x_4, x_5]$ and $Y = [y_2, y_4, y_5]$.

4 CONQUEST: A Tool for Constructing Summaries of Distributed Binary Datasets

We first identify the kernel operations in the serial algorithm, PROXIMUS, and explore various parallel formulations.

4.1 Kernel Operations in PROXIMUS. At the core of PROXIMUS is the matrix decomposition that recursively partitions the original matrix into a series of matrix pairs A_1 and A_0. If we view the recursive process as a tree, the main task at each node of the recursion tree is an alternating iterative process that maximizes the objective function, $C_c(x, y) = \frac{(x^T Ay)^2}{||x||_2^2 ||y||_2^2}$. The steps involved in this process are shown in Figure 2. The operations on lines 2 and 4 are dominated by two sparse-matrix vector multiplications, Ay and $A^T x$. Parallel sparse-matrix vector products have been the subject of extensive research for many years. The challenge in optimizing this operation is to find a load balanced partitioning of the sparse matrix that minimizes the amount of communication among processors. Existing methods [7, 13, 19, 21] view the sparse matrix as a graph and reduce the load balancing problem to a problem of partitioning the graph into p roughly equal parts, where p is the number of processors. However, graph partitioning based load balancing techniques are unlikely to succeed in our context for the following reasons:

1. The datasets of interest often contain a few dominant patterns along with a number of weak patterns. As a result, the matrices that appear in intermediate steps often differ significantly in size. This situation is illustrated in Figure 4. The matrix is initially distributed evenly among the two processors. If

```
1. sort  s = [$s_{j_1}, s_{j_2}, \ldots, s_{j_m}$] in descending order
2. $x_i \leftarrow 0$ for $1 \le i \le m$
3. $C_c(x,y) \leftarrow 0$ , $sum \leftarrow 0$
4. for $i \leftarrow 1$ to $m$ do
5.     $sum \leftarrow sum + s_{j_i}$
6.     if $sum^2/(||x||_2^2 + 1) \ge C_c(x,y)$
7.       then $x_{j_i} \leftarrow 1$
8.            $C_c(x,y) \leftarrow sum^2/||x||_2^2$
9.       else break
10.    end if
11. end for
```

Figure 3: Algorithm for computing the presence vector for a fixed pattern vector.

we simply assign the task of decomposing a child matrix to one particular processor, we could have unbalanced distribution of tasks among processors as seen at the second level and the third level of the recursion tree. In order to maintain consistent load balance among processors, the balancing heuristic must be applied at every level of the recursion tree and large amount of data must be transferred as a result. The communication cost incurred in load balancing is likely to dominate the overall cost of the computation and therefore significantly reduce scalability.

2. Computing good partitions implies finding good clusters within the dataset. While this works for more expensive post-processing on the graph (such as repeated mat-vecs for solving linear systems), they are not suitable for inexpensive operations such as those involved in CONQUEST (no FLOPS at all!). The use of existing heuristics will overwhelm the cost of executing CONQUEST without any optimization.

The process of solving for the presence vector x (and similarly pattern vector y) can be implemented in the steps shown in Figure 3. These steps can be implemented in parallel by sorting entries of s followed by a prefix sum operation. As the entries of s are bounded, they can be sorted via counting sort in linear time, serially. For its parallel implementation, sample sort is the method of choice for this problem. The challenge here is the distribution of s-vector among processors once it is sorted. If the elements of s are distributed in blocks, *i.e.*, the first processor has the maximum n/p elements and so on, then the processors that have smaller entries of s might have no work, since the loop breaks as soon as no improvement on the

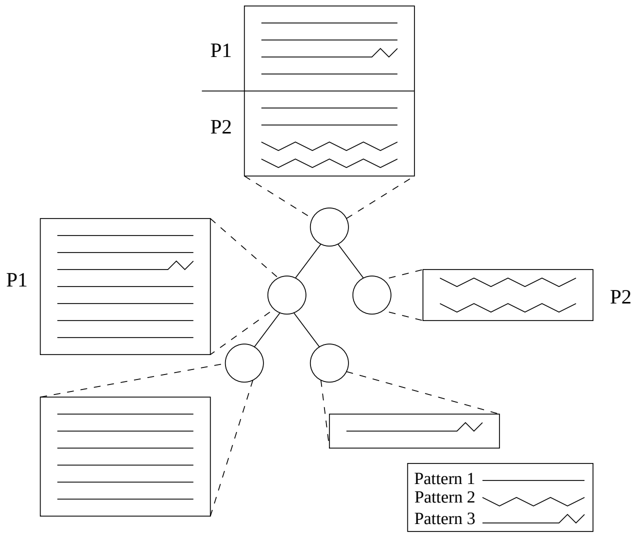

Figure 4: A parallel recursion tree that results from straightforward assignment of tasks to processors.

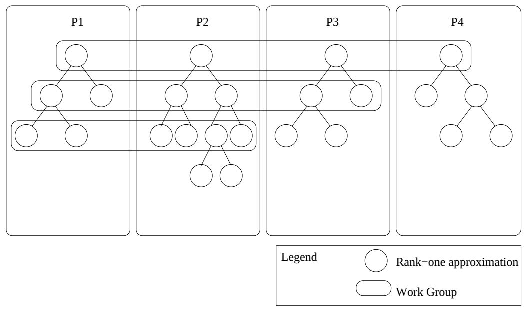

Figure 5: CONQUEST Parallel Communication Model.

objective function is possible, as seen in line 9 of the algorithm. Thus, cyclic distribution of the entries of the sorted s vector is necessary. However, in this case the vector must be redistributed after the computation of prefix sums. This necessitates an all-to-all personalized communication with message size n/p^2, which can be performed in $O(n/p)$ time on a network with bisection bandwidth $\Theta(p)$ [13]. This bandwidth requirement does not scale to large configurations. Furthermore, performing this step of computation at all iterations of rank-one approximation in parallel imposes considerable load on the network in addition to the overhead introduced by the parallelization of the sorting algorithm. For these reasons, we adopt an alternate parallelization strategy for CONQUEST.

4.2 CONQUEST **Parallel Formulation.** CONQUEST addresses performance bottlenecks of conventional parallelization techniques by relying on the principle of sub-sampling with limited communication to maintain accuracy. CONQUEST uses the concept of *work* *groups* to aggregate processors working on data with similar patterns. Initially all processors are associated with a single work group. Each processor proceeds to compute the rank-one approximation using its local data independent of others. Processors then go through a consolidation process, (described in detail in 4.2.1), to refine work groups to include only those processors that find similar patterns at the most recent step. After regrouping, processors repeat the same steps within their own work groups until the stopping criterion is satisfied.

This process is illustrated in Figure 5. In this example, there are four processors, each initially assigned to a global work group. After the first round of computation, the first three processors decide the patterns they found are similar and form a new work group. They repeat the same process within the new group. Processor P4 discovers a pattern that is sufficiently supported only by its own data (but is different from other processors' patterns) and thereafter continues on its own. After consolidation, the recursive procedure continues and processors consolidate within their work group after each step of computation in the recursion tree until each of their stopping criteria is satisfied. At this point, processors that terminate notify the remaining members of the work group of their departure. When all processors terminate, they exchange their patterns and each processor keeps a copy of all unique patterns.

The idea of constructing local work groups among processors is motivated by the observation that geographically distributed datasets often exhibit patterns that are somewhat unique in relation to their locations. For example, a Gap store in Minnesota in the winter is likely to have sales patterns very different from those observed at a store in California. This implies that the global data exchange in conventional parallelization schemes is unnecessary and the additional gains in terms of accuracy of patterns discovered from conventional strategies are likely to be limited.

4.2.1 Pattern Consolidation.

After each rank-one approximation, processors in the same work group exchange most recently discovered pattern vectors. Each processor stores all the pattern vectors as a matrix and executes a serial version of PROXIMUS to discover patterns within this matrix. Processor then replace their original patterns with a consolidated pattern that is closest to the original, and use the new pattern for continuing the process. By doing so, processors learn from each other by exchanging the summary of the data in their local partitions and discovering the global trend in patterns. It is noteworthy that communication happens only among processors that are in the same work group. Communication across work groups is not necessary since processors in different work groups are, by definition, working on datasets that have different underlying patterns. Additional communication would have very little effect in terms of improving the solution. Once new patterns are computed, processors form new work groups with others sharing the same pattern and continue computation in the same manner. This consolidation process is implemented as shown in Figure 6.

```
1. $P \leftarrow$ **all-to-all broadcast** $y$ within current work group
2. run serial algorithm on $P$ to find dominant patterns and
   store them as set $D = \{d_1, \ldots, d_k\}$.
3. **if** $|D| = 1$ **then**
4.   **continue**
5. **else**
6.   **for** $i \leftarrow 0$ **to** $|D|$ **do**
7.       **if** $y$ is similar to $d_i$
8.         $y \leftarrow d_i$
9.         create and join communication group $i$
10.      **end if**
11.  **end for**
12. **end if**
```

Figure 6: Sketch of the parallel algorithm for pattern consolidation.

We illustrate this process with a simple example that has four processors in a work group. After the broadcast, each processor has a pattern matrix that looks as follows:

$$\left[\begin{array}{c|cccccccc} P1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ P2 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ P3 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ P4 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}\right]$$

Each row in the above pattern matrix is a pattern vector discovered by a processor in the same work group, and it is tagged with the corresponding processor id. After obtaining the pattern matrix, each of the four processors tries to find patterns in this matrix using the serial version of the algorithm. This procedure results in the following patterns:

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

The two vectors found in local analysis can be thought of as the representatives of all the patterns in the pattern matrix. These representative patterns provide the basis for regrouping the processors. Processors 1, 2, and 3 have pattern vectors similar to the first representative pattern, and form a new work group. They use the first representative pattern to partition their local matrices. Processor 4 is in a group of its own and uses the second representative pattern, which in this case, is the same as its original pattern, to partition its local matrix.

4.2.2 Performance Aspects of CONQUEST**.** In essence, the CONQUEST parallel formulation replaces the global rank-one approximation in the serial algorithm with local approximation operations at each individual processor. There are two major advantages of this formulation:

1. Load balancing is no longer a major issue. The only effort required is to initially balance the load among processors. We no longer need to be concerned with the communication patterns among the partitions of the matrix at different processors. This is because kernel operations of matrix-vector multiplications and sorting operations are performed independently at each processor.

2. Communication overhead of the scheme is minimal. For each processor, there is at-most one all-to-all broadcast of its pattern vector required at each recursion step. The size of the data being exchanged is the number of non-zeros in the pattern vector, which is of the same dimension as the data matrix. For sparse data, the size of the pattern vector tends to be very small.

One potential drawback of this approach is that the processors may work on local partitions of the data most of the time. The computation of the patterns is carried out by processors largely independently of each other and therefore is at the risk of converging to undesirable local optima. The problem in this case is similar to that faced in sub-sampling.

To understand the likelihood of this event, consider the classical sub-sampling problem. Using Chernoff bounds, Toivonen [36] shows that the probability δ of error ϵ in frequency of a subset in the original dataset and the sample is bounded by a function of the sample size, $|s|$, and the error bound ϵ.

THEOREM 4.1. *Let T be a set of transactions on set S of items. If $t \subset T$ is a sample of size*

$$|t| \geq \frac{1}{2\epsilon^2} ln \frac{2}{\delta},$$

then, for any subset $s \subset S$ of items, the probability that $e(s,t) > \epsilon$ is at most δ, where $e(s,t) = |fr(s,T) - fr(s,t)|$ is the difference between the frequencies of s in T and t, respectively.

Data	# transactions.	# items	# patterns (approximately)
M10K	7513	472	100
M31K	23228	714	100
L100K	76025	178	20
LM100K	76326	452	50
M100K	75070	852	100
HM100K	74696	3185	500
H100K	74733	7005	2500
M316K	237243	905	100
M1M	751357	922	100

Table 1: Description of datasets used in experiments.

Note that in the context of our problem, T is the matrix of interest and the items in S are the columns of T. In this respect, the part of the matrix at each processor can be considered as a subsample t of the original matrix. Thus, the theorem applies directly with frequency of item sets in the theorem being viewed as the frequency of patterns in CONQUEST. Since datasets in practical applications are likely to be large, the error bound and the probability of error are both quite small. In addition, we are able to further alleviate this problem to a satisfactory degree by periodic reconciliation among processors to improve the accuracy of patterns that they converge to.

5 Experimental Results

In this section, we first evaluate the parallel performance of CONQUEST by examining its run time properties and its accuracy in terms of discovered patterns with respect to the serial algorithm. We then show CONQUEST's application as a preprocessor in association rule mining and compare the results with those of the *a-priori* algorithm applied on raw data. We examine the results in terms of precision and recall of discovered rules.

5.1 Execution Environment. CONQUEST is implemented for message-passing platforms using MPI [11]. The measurements presented here are taken on a set of eight Intel Xeon servers.

The data matrices used in the experiments are generated using the synthetic data generator made available by the IBM Quest Research Group [1]. We generate two sets of data, one with varying number of transactions and the other with varying number of patterns. In the first set, the number of patterns is fixed to 100 (medium) and five instances, named M10K, M31K, M100K, M316K and M1M, containing $\approx 10K$, $\approx 31K$, $\approx 100K$, $\approx 316K$ and $\approx 1M$ transactions are generated, respectively. In the second set, the number of transactions is fixed to $\approx 100K$ (medium) and five instances, named L100K, LM100K, M100K, HM100K and H100K, that contain 20 , 50, 100 , 500, and 2500 patterns are generated, respectively. We

	Run Time		# patterns		Average
# of trans.	serial	parallel	serial	parallel	ham. dist.
10K	0.40	0.07	303	409	4.85
31K	0.96	0.16	275	501	4.8
100K	7.90	0.99	246	544	4.51
316K	27.62	2.35	207	463	2.59
1M	76.78	7.25	180	443	2.52

Table 2: Comparison of patterns discovered by parallel and serial formulations for varying number of transactions.

	Run Time		# patterns		Average
# of pat.	serial	parallel	serial	parallel	ham. dist.
20	5.66	0.49	192	228	1.82
50	7.05	0.64	224	395	2.41
100	7.90	0.99	246	544	4.51
500	10.6	1.40	950	2407	4.68
2500	18.13	4.94	5370	12650	6.13

Table 3: Comparison of patterns discovered by parallel and serial formulations for varying number of underlying patterns.

set the average number of items per transaction and the average number of items per pattern both to 10. We also set the average correlation between every pair of pattern to 0.1 and the average confidence of a rule to 90%. Note that although other choices of these parameters are explored, we are restricting our discussion to a single setting for simplicity, which is chosen to be reasonable and observed to be representative for general performance results. As might be guessed intuitionally, the number of discovered patterns grows by increasing between-pattern correlation, for both serial and parallel algorithms. Table 1 shows the exact number of transactions, number of items, and number of patterns in all datasets.

For all experiments reported in this section, we set the bound on the normalized Hamming radius of identified clusters to 0.01 and use the *Partition* initialization scheme. For details on these parameters, please see [24].

5.2 Parallel Performance. We demonstrate that CONQUEST is capable of excellent speedup, while maintaining accuracy of the patterns found by comparing the runtimes of CONQUEST on eight machines and the serial algorithm on an identical machine with the same parameters. Tables 2 and 3 summarize the parallel performance for varying number of rows and number of patterns, respectively.

The average Hamming distance between the patterns discovered by CONQUEST and the serial program is calculated as follows:

$$\bar{H}(Y, Y') = \frac{\sum_{i=1}^{k} H(y_i, Y')}{k}$$

where $H(y_i, Y') = \min_{0 \le j < k'} ||y_i \texttt{ XOR } y'_j||$, $y_i \in Y$ are the patterns discovered by CONQUEST, $y'_j \in Y'$ are the patterns discovered by PROXIMUS, k and k' are the number of rows (patterns) in Y and Y', respectively.

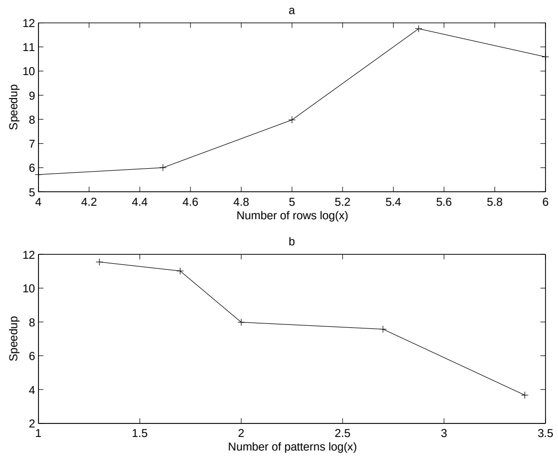

Figure 7: Speedup of parallel formulation on eight machines over serial program with (a) increasing number of transactions (b) increasing number of underlying patterns.

Here, $H(y_i, Y')$ measures the difference between pattern y and the pattern in Y' that represents y best. Thus, $\bar{H}(Y, Y')$ provides an average measurement on how well serial patterns are represented by parallel patterns.

As the number of rows in the dataset grows from 10K to 1M, CONQUEST consistently demonstrates speedups ranging from 6 to 12 (Figure 7). A similar behavior is observed with respect to increasing number of patterns. The super-linear speed-up, observed in some cases, is attributed to the effect of sub-sampling. CONQUEST and PROXIMUS perform different amounts of computation – and due to sub-sampling, CONQUEST often performs less computation than its serial counterpart. The tradeoff for this lower computational cost is that CONQUEST returns a larger number of pattern vectors.

Table 4 shows the parallel performance on dataset of 1M rows and 100 patterns with increasing number of processors. Figure 8 shows that CONQUEST discovers as many as 1.2 to 2.5 times the number of patterns discovered by the serial program. While this is an undesirable consequence resulting from the choice made in the design of the parallel formulation in order to avoid large communication overhead, Figure 9 shows that the average number of bit differences (Hamming distance) between the parallel patterns and the serial patterns is in fact quite small. This indicates that while there are more patterns found, most of these “new” patterns are quite similar to those patterns identified by the serial algorithm. This means that some patterns discovered by the parallel algorithm are redundant, which is natural and would be expected in a real-life setting as there might be similar patterns in different sites which are partially different. Thus, significant intersection may not mean redundancy in some contexts.

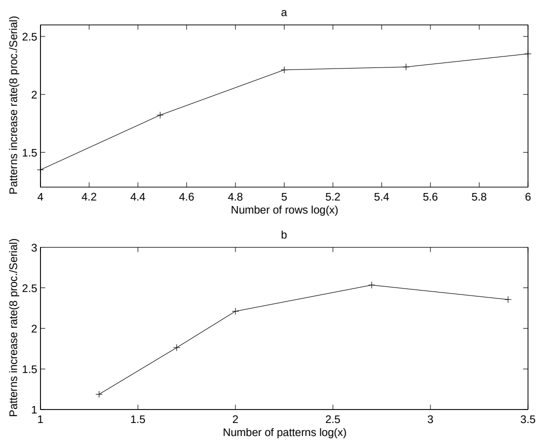

Figure 8: Increase in number of discovered patterns with (a) increasing number of transactions (b) increasing number of underlying patterns.

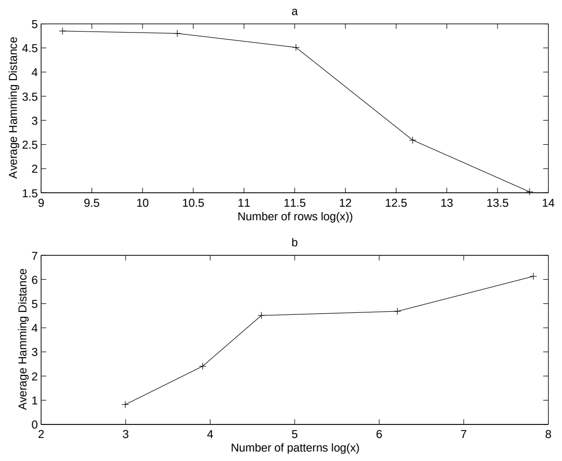

Figure 9: Average Hamming distance between patterns discovered by parallel formulation on eight machines and serial program with (a) increasing number of transactions (b) increasing number of underlying patterns.

# proc.	Runtime	# patterns	Avg. ham. dist.
1	76.78	180	0
2	32.55	194	0.15
4	14.46	248	0.48
6	8.68	288	0.73
8	7.25	443	2.52
10	4.80	361	1.97
12	3.90	390	2.43

Table 4: Performance of CONQUEST on M1M data with increasing number of processors.

Although CONQUEST addresses redundancy to a certain extent taking advantage of the formulation of the problem, *i.e.*, each transaction is represented by exactly one pattern and the distance between a transaction and its representative pattern is bounded, it is necessary to provide a quantitative definition of redundancy in order to explore the opportunities to further reduce this overhead.

We now examine the bandwidth requirement of CONQUEST. As indicated in section 4.2.2 each processor is required to perform at-most one all-to-all broadcast of its pattern vector within its own work group at each recursion step. We can measure the bandwidth required in this process by recording the number of bytes each processor sends and receives during the all-to-all communication. Since processors may join different work groups, and thus generate different amount of network traffic, we choose the processor that generates the most traffic as a meaningful measure of the bandwidth required by CONQUEST. Figures 10 and 11 show the maximum network traffic generated during the execution of CONQUEST on various datasets. These results show that the heaviest traffic, generated during the execution on a dataset of 100K transactions with 2500 patterns, is about 80K bytes, which is well within the capability of typical wide area networks (it takes milliseconds to transfer 80KB of data over a 1Mbps network link). Furthermore, comparing this communication time to the total parallel run time, we can safely conclude that the time it takes to transfer the small amount of data generated by CONQUEST is indeed a very small fraction of the overall run time.

5.3 Application to Association Rule Mining. In this section, we illustrate an application of CONQUEST as a tool for gathering large amounts of data that are distributed among various sites for applying expensive data mining algorithms on the collection. The idea is to construct a summary of distributed data and mining this summary for association rules using a conventional algorithm sequentially. This is generally more efficient than collecting all of the data at one site or running parallelized version of the conventional algorithm since the underlying algorithm has considerable parallelization overhead, especially for geographically distributed platforms.

Association rule mining is a well-known and extensively studied problem in data mining[3]. Given a set of transactions over a set of items, association rule mining aims to discover rules between itemsets that satisfy the minimum support and confidence constraints prescribed by the user. An association rule is an assertion of the kind "$\{$bread, milk$\} \Rightarrow \{$butter$\}$", meaning that if a transaction contains bread and milk, it is also likely to contain butter. The support of a rule in a set of transactions is defined as the percentage of transactions that contain all items of the rule over all transactions in the set. The confidence of a rule is the conditional probability of observing the right

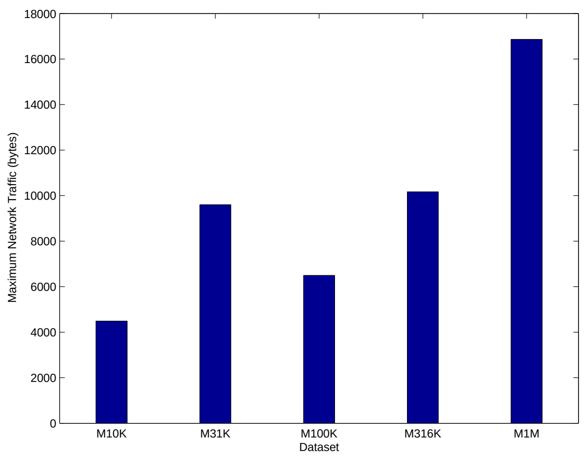

Figure 10: The maximum network traffic generated by CONQUEST on datasets with different number of transactions, ranging from 10K to 1M.

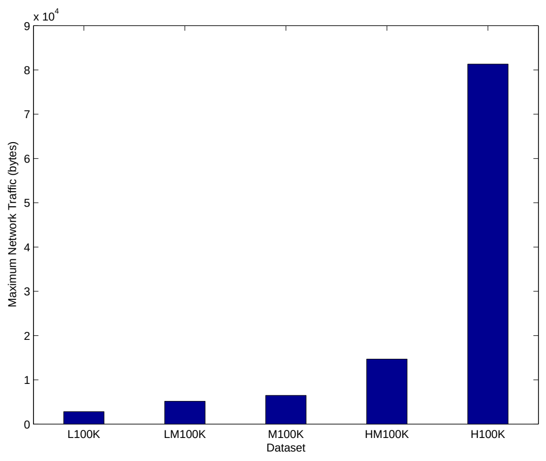

Figure 11: The maximum network traffic generated by CONQUEST on datasets with different number of patterns, ranging from 20 to 2500.

hand side, given the left hand side.

We illustrate the use of CONQUEST for the purpose of association rule mining with an example. Given a sample set of 6 transactions as shown in Figure 12(a), we can construct a binary transaction matrix by mapping transactions to rows and items to columns and setting entry t_{ij} of the transaction matrix T to 1 if item j is in transaction T_i. The resulting matrix is shown in Figure 12(b). As shown in Figure 13(a) two rank-one approximations decompose T into a set of orthogonal presence vectors(x_i) and a set of pattern vectors(y_i), with $x_1 = [0\ 0\ 1\ 1\ 1\ 1]^T$ and $y_1 = [0\ 0\ 1\ 1\ 1]$ in one pair and $x_2 = [1\ 1\ 0\ 0\ 0\ 0]^T$ and $y_2 = [1\ 1\ 1\ 0\ 0]$ in another. We can construct a set of virtual transactions, using the pattern vectors as transactions and the number of non-zeros in presence vectors as their weights (Figure 13(b)). We can now analyze this smaller approximate transaction set using any existing association rule mining technique. Note that this summary can be obtained using CONQUEST if the transactions are distributed among various sites.

We show the validity of this technique by using existing association rule mining software to mine the original dataset M1M (above) and the approximate set generated by CONQUEST. The software we use is an open source implementation [6] of the well-known *a-priori* algorithm. We also create a slightly modified version of this software which is capable of mining weighted approximate transaction sets.

Table 5 shows the comparison of results obtained by running the *a-priori* software on the original 1M transaction matrix and on the approximate transaction matrix generated by running CONQUEST using 8 processors. The *a-priori* software was run with 90% confidence in all instances. The figures in the table include minimum support for the rules, total time spent mining the original matrix, total time spent mining approximate matrix, rules discovered from the original matrix, rules discovered from the approximation matrix, rules matched in the two cases, precision and recall. *Precision* is defined as the number of matching rules over all rules that are discovered on the approximate transaction set, measuring how precise the results obtained on the approximate set are. *Recall* is defined as the fraction of the rules discovered in the original transaction set that are also discovered in the approximate set, measuring how successful the compression is on recalling the rules that are present in the original data.

min. sup. %	time orig. sec.	time apprx. sec.	rules orig. #	rules apprx. #	rules match #	prec. %	recall %
2.0	28.45	0.38	63503	53280	53280	100.0	91.0
2.5	12.69	0.14	27090	20299	20299	100.0	75.0
3.0	10.99	0.14	20353	19527	19527	100.0	98.1
3.5	10.56	0.13	19617	19527	19527	100.0	99.5
4.0	9.23	0.08	12793	12793	12793	100.0	100.0
4.5	9.27	0.08	12787	12789	12787	100.0	99.98
5.0	8.98	0.06	12398	10955	10955	100.0	90.9
5.5	7.20	0.05	6740	6732	6732	100.0	99.88
6.0	7.12	0.05	6732	6732	6732	100.0	100.0

Table 5: Association rule mining performance of CONQUEST on M1M dataset.

As we observe from our results, CONQUEST demonstrates excellent overall accuracy. Precision values in all cases are 100%. Recall values are almost all close to 100%; dropping below 90% in one case (75%)(Figure 14), and immediately rising up to 98%. This phenomenon is sometimes observed and is due to the sudden change in the support value of a large group of data (an artifact of

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

the Quest data generator). While maintaining high accuracy, CONQUEST provides a speed-up of several orders of magnitude over *a-priori* operating on un-preprocessed data. Note that the time for constructing the summary for M1M dataset is 7.25 seconds, which is well below the time spent on mining the original transaction set for almost all meaningful support values.

This demonstrates that, in addition to being a useful tool for the purpose of correlating large high-dimensioned datasets, CONQUEST can be used as a powerful preprocessor for creating summaries of distributed data for conventional data mining techniques without incurring the expensive overhead of transferring and centrally storing all of the original datasets.

6 Conclusions

In this paper, we have presented CONQUEST, a novel parallel formulation of a powerful new technique for analysis of large high-dimensional binary attributed sets. This parallel formulation is designed to conserve network bandwidth and alleviate the problem of load balancing faced in other conventional parallelization techniques. These properties make it ideal for mining extremely large datasets over geographically distributed locations. We also show that CONQUEST successfully relies on the principle of sub-sampling and periodic consolidation among processors to achieve excellent speed-ups while maintaining high accuracy. Finally, we demonstrate the application of CONQUEST in association rule mining as a powerful pre-processing tool to significantly accelerate existing

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

(a)

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

(b)

Figure 13: (a) Decomposition of transaction matrix of the transaction set of Figure 12 and (b) the corresponding approximate transaction set.

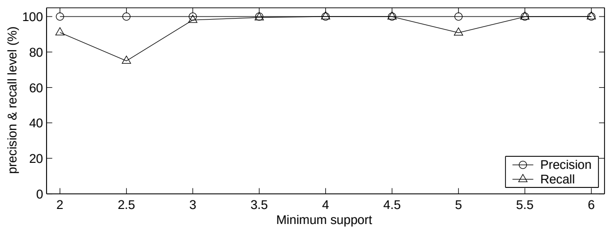

Figure 14: Precision and recall rates with varying minimum support.

data mining software.

References

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

[2] R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation and experience. Technical report, IBM Almaden Research Center, IBM Corp, 1996.

[3] 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.

[4] 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.

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

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

[7] U. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. on Parallel and Distributed Systems*, 10(7):673–693, 1999.

[8] D. Cheung, V. Ng, A. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. *IEEE Trans. on Knowledge and Data Engineering*, 8(6):911–922, 1996.

[9] 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.

[10] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, pages 245–260, 2000.

[11] M. P. I. Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.

[12] D. Foti, D. Lipari, C. Pizzuti, and D. Talia. Scalable parallel clustering for data mining on multicomputers. *Lecture Notes in Computer Science*, 1800, 2000.

[13] A. Grama, A. Gupta, V. Kumar, and G. Karypis. *Introduction to Parallel Computing*. Addison Wesley, 2003.

[14] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. pages 277–288, 1997.

[15] 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.

[16] M. Joshi, G. Karypis, and V. Kumar. ScalparC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *IPPS: 11th International Parallel Processing Symposium*. IEEE Computer Society Press, 1998.

[17] D. Judd, P. K. McKinley, and A. K. Jain. Large-scale parallel data clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):871–876, 1998.

[18] H. Kargupta, W. Huang, S. Krishnamurthy, B. Park, and S. Wang. Collective principal component analysis from distributed, heterogeneous data. In *Proc. 4th European Conf. Principles of Data Mining and Knowledge Discovery (PKDD 2000)*, pages 452–457, 2000.

[19] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, 1995.

[20] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York, 1990.

[21] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J*, 49(2):291–307, 1970.

[22] 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.

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

[24] M. Koyutürk and A. Grama. *Proximus*: A framework for analyzing very high-dimensional discrete attributed datasets. In *Proc. Ninth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD 2003)*, pages 147–156, 2003.

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

[26] M. Koyutürk, A. Grama, and N. Ramakrishnan. Non-orthogonal decomposition of binary matrices for bounded-error data compression and analysis. *submitted to ACM Trans. Mathematical Software*, 2004.

[27] A. Mueller. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 1995.

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

[29] C. F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21(8):1313–1325, 1995.

[30] B. Park and H. Kargupta. Distributed data mining: Algorithms, systems, and applications. In N. Ye, editor, *The Handbook of Data Mining*. Lawrence Erlbaum Assoc., 2003.

[31] J. S. Park, M.-S. Chen, and P. S. Yu. Efficient parallel data mining for association rules. In *Proc. Fourth Intl. Conf. Information and Knowledge Management*, pages 31–36, 1995.

[32] F. J. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Knowledge Discovery and Data Mining*, pages 23–32, 1999.

[33] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 544–555. Morgan Kaufmann, 3–6 1996.

[34] Shintani and Kitsuregawa. Hash based parallel algorithms for mining association rules. In *PDIS: International Conference on Parallel and Distributed Information Systems*. IEEE Computer Society Technical Committee on Data Engineering, and ACM SIGMOD, 1996.

[35] A. Srivastava, E.-H. Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery*, 3(3):237–261, 1999.

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

[37] O. R. Zaiane, M. El-Hajj, and P. Lu. Fast parallel association rule mining without candidacy generation. In *ICDM*, pages 665–668, 2001.

[38] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. In *Proc. ACM/IEEE Conf. Supercomputing*, 1996.