Algorithmica (2006) OF1–OF25
DOI: 10.1007/s00453-006-1218-x

Algorithmica

© 2006 Springer Science+Business Media, Inc.

CONQUEST: A Coarse-Grained Algorithm for Constructing Summaries of Distributed Discrete Datasets[1]

Jie Chi,[2] Mehmet Koyutürk,[2] and Ananth Grama[2]

Abstract. In this paper we present a coarse-grained parallel algorithm, CONQUEST, for constructing bounded-error summaries of high-dimensional binary attributed data in a distributed environment. Such summaries enable more expensive analysis techniques to be applied efficiently under constraints on computation, communication, and privacy with little loss in accuracy. While the discrete and high-dimensional nature of the dataset makes the problem difficult in its serial formulation, the loose-coupling of distributed servers hosting the data and the heterogeneity in network bandwidth present additional challenges. CONQUEST is based on a novel linear algebraic tool, PROXIMUS, which is shown to be highly effective on a serial platform. In contrast to traditional fine-grained parallel techniques that distribute the kernel operations, CONQUEST adopts a coarse-grained 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 in dynamically orchestrated work groups to construct consensus patterns from these local patterns. Individual sites may then decide to continue their participation in the consensus or leave the group. Such parallel formulation implicitly resolves load-balancing and privacy issues while reducing communication volume significantly. Experimental results on an Intel Xeon cluster demonstrate that this strategy is capable of excellent performance in terms of compression time, ratio, and accuracy with respect to post-processing tasks.

Key Words. Coarse-grained data mining algorithms, 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). Such datasets are often binary in nature, or can be transformed into binary datasets easily. This paper focuses on efficient distributed techniques for analysis of such binary attributed datasets.

Analysis of binary 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 [1]–[4], a variety of application

[1] A preliminary version of this paper was presented at the SIAM Data Mining Conference, 2004.

[2] Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA. chij@purdue.edu, {koyuturk,ayg}@cs.purdue.edu.

Received February 27, 2004; revised July 21, 2005. Communicated by F. Dehne.
Online publication May 24, 2006.

characteristics necessitate the development of corresponding distributed formulations. These application characteristics include:

- *Loose-coupling*: As the sites that contain data are often loosely coupled, application of fine-grained parallel algorithms for the above-mentioned problems is likely to be very inefficient in terms of communication. In addition, the distribution of data and the network bandwidth may be heterogeneous among different sites, making the application of fine-grained algorithms considerably more difficult.
- *Data volume*: Large datasets often reside on geographically distributed locations. For purposes of data mining, clustering, classification, and learning, collecting all of the data at a single location is infeasible because of storage constraints.
- *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/or applying fine-grain parallel implementations of the underlying algorithms directly on the data 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 aggregated patterns may be communicated. Constructing summaries in parallel, rather than exchanging the original data implicitly addresses such constraints.

CONQUEST is based on the linear algebraic tool PROXIMUS, which uses a variant of Semi-Discrete Matrix Decomposition (SDD) [5] to compress binary datasets efficiently in an error-bounded fashion. In PROXIMUS the compression (or summarization) problem is formulated as follows. Given a set of binary vectors, find a compact set of binary vectors such that each input vector is within bounded distance from some output vector. Based on this formulation, each output vector may be interpreted as a pattern in the dataset. PROXIMUS is available over the public domain at http://www.cs.purdue.edu/homes/koyuturk/proximus/. To date, it has over 300 installations in over 10 countries.

While parallelizing PROXIMUS, CONQUEST uses a coarse-grained formulation that relies on the principle of sampling to reduce communication overhead while maintaining high accuracy. Specifically, each individual site computes its local patterns (vectors) independently. Various sites cooperate within dynamically orchestrated work groups to construct consensus patterns from these local patterns. Then individual sites decide to participate in the consensus or to leave the group based on the proximity of their local patterns to consensus patterns. We demonstrate that this strategy results in excellent parallel performance, particularly on loosely coupled platforms.

The optimization criteria for the problem are: minimizing the error in approximation provided by the output patterns, and the number of patterns in the approximation for given bound on error. Since the parallel formulation does not correspond directly to the serial formulation and relies on the principle of sampling, an important consideration is the effect of our parallelization strategy on the quality of the output. We show experimentally that CONQUEST demonstrates excellent parallel performance, in terms of efficiency, quality of approximation, and redundancy in the number of detected patterns.

The rest of the paper is organized as follows. In Section 2 we discuss prior research related to CONQUEST's serial and parallel formulations. Section 3 introduces PROXIMUS briefly. In Section 4 we discuss the challenges associated with a coarse-grained parallel formulation, show the drawbacks of a fine-grained parallelization approach and motivate

our design decisions. In Section 5 we evaluate the performance of CONQUEST on a cluster of 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. 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 [1], [3], [4]. 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 [5]–[8]. 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 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 called the Centroid Method. The computation of a centroid decomposition depends only on the correlations between the rows of the matrix. The main difference between SVD and the centroid method is that the centroid method tends to discover a single dominant pattern while the SVD tends to discover the overall trend in data. This may be a collection of several independent patterns. Orthogonal matrix decompositions have been used by several distributed data mining algorithms [9], [10].

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) [11], 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 a 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 [12]–[15]. Among these, CLARA [16] attempts to create multiple samples and applies PAM (Partitioning Around Medoids) 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 [10], [17]–[24]. Most of these approaches are based on the a priori algorithm [25] 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. Such formulations are well suited to massively parallel platforms. Another class of parallel formulations is based on computing frequent itemsets on each site individually and then working in parallel to join individual patterns into global association rules. This provides a more suitable framework for loosely coupled distributed systems. Work on parallel classification has resulted in systems such as SPRINT [26], ScalParC [27], and others [28]. These systems typically use decision tree-based classification.

In comparison with 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 on distributed platforms with large datasets tends to be expensive, it is desirable 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 [2], [29] 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.

3.1. *Discrete Rank-One Approximation of Binary Matrices*

DEFINITION 1 (Discrete Rank-One Approximation of Binary Matrices). 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:

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

As described above, discrete rank-one approximation can be considered as the discovery of the most dominant pattern in the matrix. This pattern is characterized by a pattern vector y and its presence in the rows of the matrix is signified by the presence vector x. The outer product of the presence and pattern vectors provides a rank-one approximation for A that is at the minimum Hamming distance from A over all binary matrices of rank one. The problem of finding a rank-one approximation is NP-hard. Therefore, PROXIMUS uses an alternating iterative heuristic as a fast and effective technique that is ideally suited to the discrete nature of the data.

It can be easily shown that minimizing the error in a rank-one approximation is equivalent to maximizing

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

We show that this optimization problem can be solved in linear time for x if y is fixed.

LEMMA 1. *For fixed y, the binary vector x that maximizes the objective function of* (2) *is given by*

$$x(i) = \begin{cases} 1, & \text{if } \ 2s(i) \geq \|y\|_2^2, \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

where $s = Ay$.

PROOF. Assume that vector x^* maximizes $C_d(x, y)$. Let $x^*(i) = 1$. Let $\hat{x}$ be the same as x^*, except that $\hat{x}(i) = 0$. Then $C_d(\hat{x}, y) = C_d(x^*, y) - 2s(i) + \|y\|^2 \leq C_d(x^*, y)$. Thus, $2s(i) \geq \|y\|_2^2$. A similar argument applies for the case $x^*(i) = 0$. □

Therefore, it is possible to find the optimal solution to x for fixed y in linear time. The same process can be applied 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. The fundamental operation in each iteration of this algorithm is a matrix-vector multiplication, which can be performed in time linear in the number of the non-zeros of matrix A. Note also that the number of iterations is bounded by the number of columns (rows) and generally a few iterations are sufficient for convergence in practice [30].

An elegant continuous approximation for the objective function of (2) based on SDD is $C_c(x, y) = (x^T Ay)^2 / \|x\|_2^2 \|y\|_2^2$ [5]. While not being equivalent to the original objective function, this function might provide better approximations especially for very sparse matrices. Both algorithms derived from these two objective functions are implemented in PROXIMUS and CONQUEST. Although we base our discussion on the original (discrete) objective function, the algorithms and analysis that follow from the continuous approximation are similar. The differences between the two objective functions are discussed in detail in [30].

3.2. *Recursive Decomposition of Binary Matrices.* PROXIMUS uses the rank-one approximation of a 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 with 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 resulting approximation adequately represents the entire matrix. Adequacy of representation is evaluated in terms of Hamming distance between the input vectors and discovered pattern vectors. The Hamming radius of a matrix is defined as the maximum of the Hamming distances of all rows in the matrix that are present in the approximation to the pattern vector.

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

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

If 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.

EXAMPLE 1. Figure 1 illustrates the recursive structure of PROXIMUS. Starting with matrix A, a rank-one approximation to A is computed. Matrix A is then partitioned into A_1 and A_0 based on the presence vector x_1. The rank-one approximation to A_1 returns a presence vector of all ones and the approximation is adequate so the recursion stops at that node and y_2 is recorded as a dominant pattern. On the other hand, matrix A_0 is

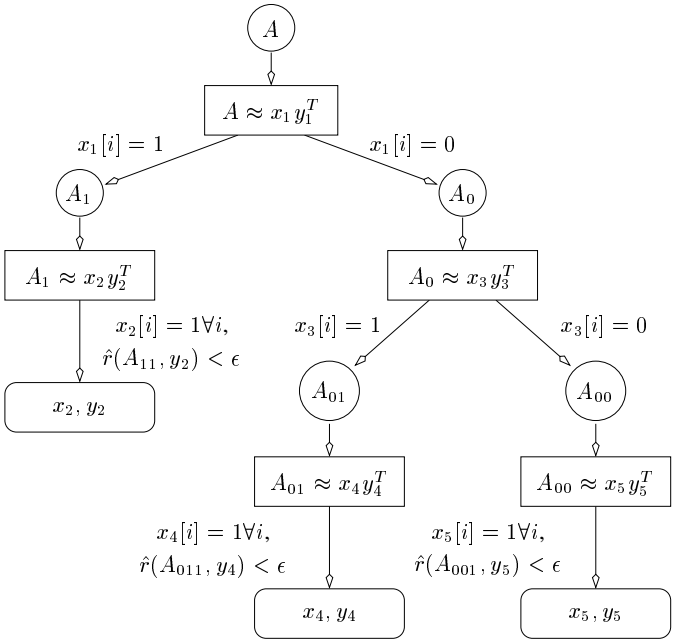


Fig. 1. Recursive structure of PROXIMUS. Leaves of the recursion tree correspond to final decomposition.

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 $Y = [y_2, y_4, y_5]$.

The running time of each rank-one approximation is linear in the number of non-zero entries in the matrix, assuming that the number of iterations is bounded. As the number of non-zeros in all the matrices that appear at a single level of the recursion tree is equal to the number of non-zeros in the original matrix ($nz(A)$), and the height of the recursion tree is at most equal to the number of patterns (k), the running time of the recursive decomposition is $O(k \times nz(A))$.

4. CONQUEST: A Coarse-Grained Tool for Constructing Summaries of Distributed Binary Datasets. In this section we present a coarse-grained parallel algorithm for constructing bounded-error summaries for binary datasets. The algorithm is based on the model and algorithms described in the previous section. However, it is not a fine-grained parallelization of PROXIMUS derived from a direct parallelization of the kernel operations. Instead, it adopts a coarse-grained approach that relies on the principle of sub-sampling to maintain quality of the solution while minimizing communication overhead. The distributed formulation is formally defined as follows:

DEFINITION 2 (Bounded-Error Approximation of Distributed Binary Vectors). Given m binary vectors of size n distributed across p sites, find a set of $k \ll m$ binary vectors of size n, so that for any input vector, there is an output vector such that the Hamming distance between the two vectors is at most ε.

Here, ε is a prescribed bound depending on the application. One can view the problem as decomposing p binary matrices $A_0, A_1, \ldots, A_{p-1}$ of size $m_i \times n$ for $0 \leq i \leq p-1$ where $\sum_{i=0}^{p-1} m_i = m$ to obtain global presence and pattern matrices X and Y of size $m \times k$ and $n \times k$ such that

$$A = \begin{bmatrix} A_0 \\ A_1 \\ \cdots \\ A_{p-1} \end{bmatrix} \approx XY^T. \tag{4}$$

Here X and Y approximate A in the sense that $\|A(i) - Y(j)\|_2^2 \leq \varepsilon$ for i, j such that $X(i, j) = 1$, where $A(i)$ denotes the ith row of matrix A. This approximation provides a global view of the underlying patterns in the overall data characterized by matrix Y. Matrix X on the other hand, signifies the presence of each pattern in the rows of a matrix at any site. As $k \ll m$, it is possible to replicate Y over all sites so that all sites will have a global view of the entire data, which can be used for further post-processing for the purpose of clustering, classification, pattern discovery, and so on.

EXAMPLE 2. A sample instance of the problem is shown in Figure 2. The instance consists of 12 binary vectors of size 5 distributed across four sites. Matrices X and Y shown in the figure provide an approximation to the input vectors such that each input

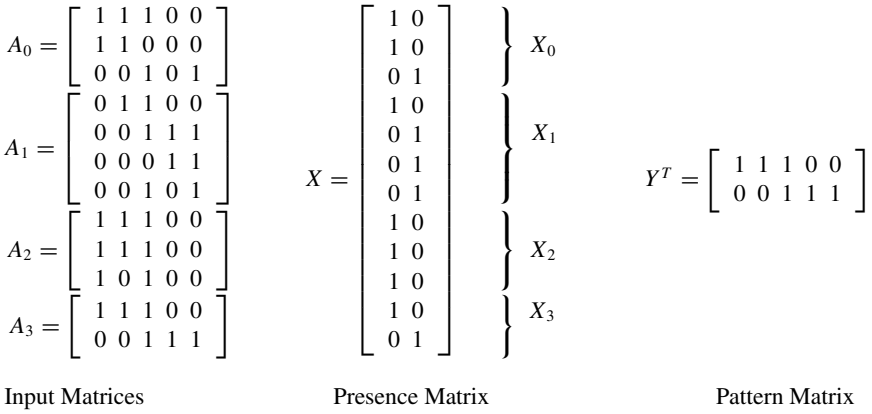


Fig. 2. A sample instance for bounded-error approximation of distributed binary vectors. The input is 12 vectors distributed across four sites. The two output pattern vectors provide an approximation for the input vectors within a Hamming distance of at most 1.

vector is within Hamming distance of 1 of at least one pattern vector. Note that the pattern matrix is common to all sites while the presence matrix might be distributed across sites correspondingly.

4.1. *Drawbacks of Fine-Grained Parallelization.* In the distributed formulation of the problem, no assumptions are made on the distribution of the input vectors among various sites. In other words, the number of vectors and the underlying pattern structure of the vectors residing at different sites are allowed to be arbitrary, making the model applicable to any distributed platform. This assumption simplifies the task of parallelization for fine-grained approaches and thus provides a more appropriate framework for the discussion of fine-grained parallelization, while preserving validity of the observations on drawbacks of such parallelization approaches.

In order to demonstrate the drawbacks of traditional fine-grained parallelization approaches, consider the following simple scenario: given a matrix A, find an error-bounded binary non-orthogonal decomposition for A in parallel on p processors. As discussed in the previous section, the underlying algorithm for binary non-orthogonal decomposition is a recursive application of rank-one approximation to the input matrix and its sub-matrices. Therefore, rank-one approximation is the core procedure in the algorithm. An outline of the alternating iterative heuristic for rank-one approximation is shown in Figure 3.

A fine-grained parallel algorithm would perform each rank-one approximation in parallel. As seen in Figure 3, the major computation that takes place in this algorithm is repeated matrix-vector multiplications (mat-vec), which appear on lines 2.1 and 2.3. Once the mat-vecs are computed, the optimization problems of lines 2.2 and 2.4 can be solved with relatively little effort. Therefore, the main computational bottleneck for the algorithm is comprised of the two mat-vecs that are performed repeatedly.

Algorithm RANKONEAPPROXIMATION(Binary Matrix A)
▷ **returns** Binary Vectors x and y that minimize $\|A - xy^T\|_2^2$
1 **initialize** y
2 **repeat**
2.1 $s \leftarrow Ay$
2.2 **solve** for x to **maximize** $2x^T s - \|x\|_2^2 \|y\|_2^2$
2.3 $s \leftarrow x^T A$
2.4 **solve** for y to **maximize** $2s^T y - \|x\|_2^2 \|y\|_2^2$
3 **until** no improvement on $C_d(x, y)$ is possible

Fig. 3. Outline of the alternating iterative heuristic for rank-one approximation.

Repeated mat-vec is an extensively studied problem in parallel computing [31]. For the case of sparse rectangular matrices as in our problem, three possible matrix partitioning schemes exist for mapping data and computation to processors, as shown in Figure 4. These are one-dimensional mapping based on rows, one-dimensional mapping based on columns, and two-dimensional mapping. Consider a one-dimensional mapping based on rows, in which each processor is responsible for a set of rows in the matrix. This is in agreement with the nature of our algorithm since we partition the matrix based on rows after each rank-one approximation. While computing each mat-vec Ay, a processor needs the entire y vector and it computes the part of vector x that corresponds to its set of rows. Therefore, while computing $x^T A$, each processor will need to communicate with other processors to obtain the entire y vector. A one-dimensional mapping based on columns also leads to a similar communication pattern, but it has additional complications for our problem as the recursive decomposition is carried out on the rows during the course of the algorithm. For partitioning schemes that only take computational load balancing into account, these schemes require a communication volume of $O((p-1)m)$ and $O((p-1)n)$, respectively. A two-dimensional mapping reduces the volume of communication to $O((\sqrt{p}-1)(m+n))$ [32]. However, this computation is repeated at every iteration of each rank-one approximation. Observing that all matrices that appear in the course of the computation have the same number of columns (n), the number of rows of the matrices at each level of the recursion tree add up to that of the original

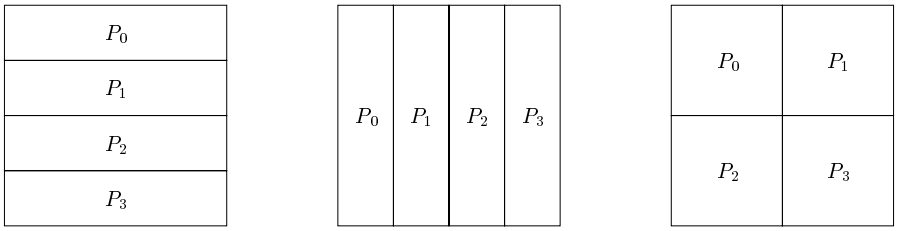


Fig. 4. Possible schemes for partitioning sparse rectangular matrices to perform repeated mat-vecs on four processors. Each processor stores the part of the matrix that is marked by its own ID.

matrix m, and the height of the recursion tree can be at most the number of discovered patterns (k), this means a total communication of $O(\sqrt{p}k(m+n))$, assuming that the number of iterations is bounded by a constant. This amount of communication poses significant bandwidth requirement, which may be unrealistic when we consider that the processors might be loosely coupled in a distributed scenario. Furthermore, the above communication overhead is likely to overwhelm the run time of the overall computation considering that the sequential runtime of the algorithm is just $O(k \times nz(A))$.

By means of some optimization techniques, it is possible to minimize the communication cost involved in the mat-vecs to take advantage of the sparse nature of the data. It is possible to formulate this problem as hypergraph partitioning or graph partitioning with vertex separator (GPVS) for the one- and two-dimensional mapping schemes, respectively. In such models the cutsize of the graph (hypergraph) corresponds to the total number of data elements to be communicated. There are many algorithms that solve these problems efficiently in order to minimize total communication volume while maintaining load balance [33]–[36]. However, the applicability of such methods to our problem is questionable 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 5. The matrix is initially distributed evenly among the two processors. If we simply assign the task of decomposing a child matrix to one 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 and minimum volume of communication, the communication-minimization 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 and communication mini-

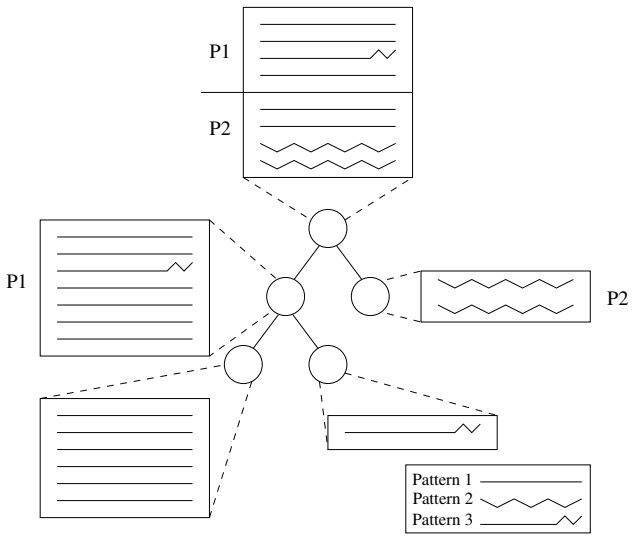


Fig. 5. A parallel recursion tree that results from a straightforward assignment of tasks to processors.

mization itself 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 easily overwhelm the cost of executing CONQUEST without any optimization (i.e., CONQUEST is much cheaper than existing graph partitioners!).

In addition to the communication cost incurred by parallel mat-vecs and transfer of data at each step of the recursive algorithm, the data is likely to be already distributed among different sites in a real-life scenario. Therefore, redistributing the data in order to minimize communication and provide load-balance might require transfer of the entire data through the network and/or solution of a graph-partitioning problem in parallel. Obviously, such attempts are likely to overwhelm the cost and purpose of the original problem. Moreover, privacy constraints might make it impossible to transfer raw data among sites. In this case, only general patterns that do not reveal the specific details in the dataset that each site owns might be exchanged between sites. All of these observations and constraints show that a fine-grained parallel formulation is likely to be inappropriate for our problem. For these reasons, CONQUEST adopts a coarse-grained parallel formulation based on the principle of sub-sampling.

4.2. *A Coarse-Grained Parallel Algorithm for Decomposing Binary Matrices*. 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 a rank-one approximation using its local data independent of the others. Processors then go through a consolidation process (described in detail in Section 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. Note that the work group is partitioned only for the sub-matrix (A_1) that corresponds to the rows that contain the pattern. While decomposing the rows that do not contain the discovered pattern (A_0), the work group of the parent matrix (A) is retained.

This process is illustrated in Figure 6. 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

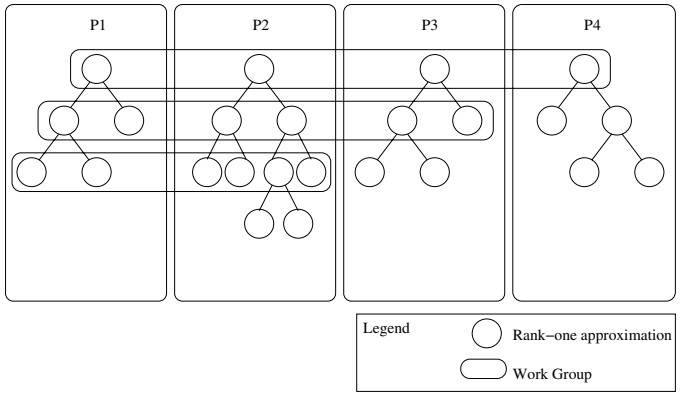


Fig. 6. CONQUEST parallel communication model.

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 the algorithm to discover patterns within this matrix. Processors 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, by definition, work on datasets that have different underlying patterns. Additional communication would have 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 this manner. This consolidation process is implemented as shown in Figure 7.

We illustrate this process with a simple example that has four processors in a work group. After the broadcast in line 2 in Figure 7, each processor has a pattern matrix shown in Figure 8(a). Each row in the pattern matrix is a pattern vector discovered by a processor in the same work group, and 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, which results in the patterns shown in Figure 8(b).

The two vectors found during 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

Algorithm PATTERNCONSOLIDATION(Pattern Vector y, Current Workgroup G)
▷ **returns** Group Pattern $\hat{y}$ and New Workgroup $G_{\hat{y}}$
1 $P \leftarrow$ **all-to-all broadcast** y within G
2 decompose P to find dominant patterns and
store them as set $D = \{d_1, \ldots, d_k\}$.
3 **if** $|D| = 1$ **then**
3.1 $\hat{y} \leftarrow d_1$ and $G_{\hat{y}} \leftarrow G$
4 **else**
4.1 **for** $i \leftarrow 1$ **to** $|D|$ **do**
4.1.1 **if** y is similar to d_i **then**
4.1.1.1 $\hat{y} \leftarrow d_i$
4.1.1.2 create and join communication group $G_{\hat{y}}$

Fig. 7. Sketch of the parallel algorithm for pattern consolidation.

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 the initial balancing of 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 mat-vecs and sorting operations are performed independently at each processor.

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

(a)

$$\left[\begin{array}{cccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}\right]$$

(b)

Fig. 8. (a) Collection of pattern vectors gathered into global pattern matrix. The ID of the processor that contains each pattern is shown on the first column of the corresponding row. (b) Group patterns discovered resulting from the decomposition of global pattern matrix.

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 matrices, the size of the pattern vector tends to be 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 local 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 typical applications of sub-sampling.

To understand the likelihood of this event, consider the classical sub-sampling problem. Using Chernoff bounds, Toivonen [4] 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 1. *Let T be a set of transactions on set S of items. If $t \subset T$ is a sample of size*

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

then, for any subset $s \subset S$ of items, the probability that $e(s,t) > \varepsilon$ 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.

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 sub-sample 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 alleviate this problem further 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 evaluate the parallel performance of CONQUEST by examining its run-time properties and the various characteristics of the patterns discovered. We then show CONQUEST's application as a pre-processor 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 [37]. The measurements are taken on a cluster of Intel Xeon 800 MHz servers networked over a 100 Mbs LAN. The use of such an environment is only motivated by the convenience of allocating a group of identical processors for the purpose of investigating characteristics of parallel performance. As we discuss later in this section, a high bandwidth cluster is by no means a prerequisite for the efficient execution of CONQUEST.

Table 1. Description of datasets used in experiments.

Data	Number of rows	Number of columns	Number of patterns
M10K	7,513	472	100
M31K	23,228	714	100
L100K	76,025	178	20
LM100K	76,326	452	50
M100K	75,070	852	100
HM100K	74,696	3,185	500
H100K	74,733	7,005	2,500
M316K	237,243	905	100
M1M	751,357	922	100

The data matrices used in the experiments are generated using the synthetic data generator made available by the IBM Quest Research Group [38]. We use a synthetic data generator because it enables us to explore program performance with varying data parameters. We generate two sets of data, one with a varying number of rows and the other with a varying number of patterns. In the first set the number of patterns is fixed at 100 (medium) and five instances, named M10K, M31K, M100K, M316K, and M1M, containing $\approx$10K, $\approx$31K, $\approx$100K, $\approx$316K, and $\approx$1M rows, respectively, are generated. In the second set the number of rows is fixed at $\approx$ 100K (medium) and five instances, named L100K, LM100K, M100K, HM100K, and H100K, that contain 20, 50, 100, 500, and 2500 patterns, respectively, are generated. We set the average number of items per row and the average number of items per pattern both to 10. We also set the average correlation between every pair of patterns 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 inferred intuitively, the number of discovered patterns grows by increasing between-pattern correlation, for both serial and parallel algorithms. Table 1 shows the exact number of rows, number of items, and number of patterns in all datasets.

Unless otherwise specified, we set the bound on the Hamming radius of identified clusters to 3 and use the *Partition* initialization scheme for all experiments discussed in the remainder of this paper. For details on these parameters, please see [2].

5.2. *Parallel Performance*

5.2.1. *Run-Time Properties*. We demonstrate that CONQUEST is capable of excellent speedup, while maintaining accuracy of the patterns found by comparing the run times 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 a varying number of rows and number of patterns, respectively.

As the number of rows in the dataset grows from 10K to 1M, CONQUEST consistently demonstrates speedups ranging from 6 to 12 (Figure 9). A similar behavior is observed with respect to increasing number of patterns. The super-linear speedup, observed in some cases, is attributed to the effect of sub-sampling. CONQUEST and PROXIMUS perform

Table 2. Comparison of patterns discovered by parallel (eight processors) and serial formulations for a varying number of rows.

	Run time		Number of patterns	
Number of rows	Serial	Parallel	Serial	Parallel
10 K	0.40	0.07	296	526
31 K	0.96	0.16	406	700
100 K	7.90	0.99	745	964
316 K	27.62	2.35	1,343	1,624
1 M	76.78	7.25	189	322

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.

5.2.2. *Quantitative Evaluation of the Patterns Discovered by* CONQUEST. We first examine the number of patterns discovered by the parallel algorithm in comparison with that of the serial algorithm. Figure 10 shows that CONQUEST discovers as many as 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 a large communication overhead, we show that this redundancy in the patterns discovered does not contribute significantly to the overall error in approximation.

Non-orthogonal decomposition of an input matrix A provides an approximation to the matrix, namely $\tilde{A} = XY^T$, where X and Y are the presence and pattern matrices, respectively. A metric that immediately follows from the definition of the problem is the error in approximation, i.e., the number of non-zeros in the residual matrix, given by the following equation:

$$\text{error} = \|A - \tilde{A}\|_F^2 = |\{a_{ij} \in (A - \tilde{A}):\ |a_{ij}| = 1\}|. \tag{5}$$

As this metric is dependent on the size of the input matrix, we use two other normalized metrics, namely *precision* and *recall*, that provide more intuitive interpretation of the results. Precision measures the percentage of ones in the approximation matrix that also

Table 3. Comparison of patterns discovered by parallel (eight processors) and serial formulations for a varying number of underlying patterns.

	Run time		Number of patterns	
Number of of patterns	Serial	Parallel	Serial	Parallel
20	5.66	0.49	578	683
50	7.05	0.64	746	810
100	7.90	0.99	745	964
500	10.6	1.40	1,148	3,077
2,500	18.13	4.94	5,344	11,645

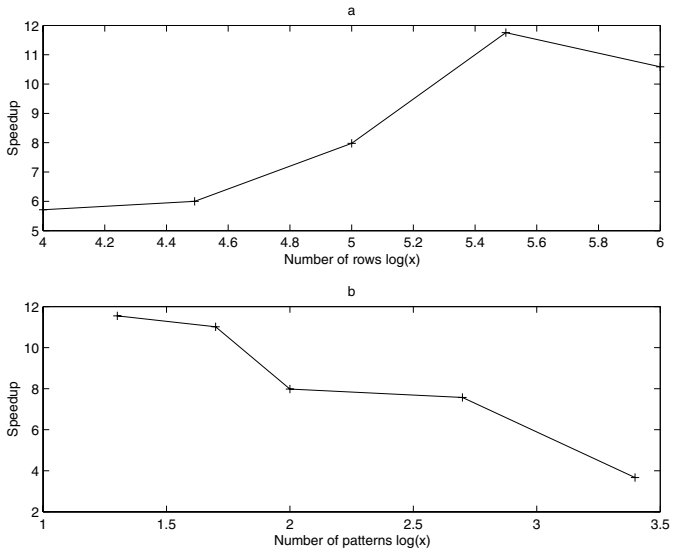


Fig. 9. Speedup obtained by CONQUEST on eight processors over a serial program with (a) an increasing number of rows and (b) an increasing number of underlying patterns.

exist in the original matrix. It is defined as follows:

$$\text{precision} = \frac{\|A\&\tilde{A}\|_F^2}{\|\tilde{A}\|_F^2} = \frac{|\{(i,j):\ A(i,j) = \tilde{A}(i,j) = 1\}|}{|\{(i,j):\ \tilde{A}(i,j) = 1\}|}. \tag{6}$$

Recall, on the other hand, measures the percentage of the ones in the original matrix that are also captured by the decomposition. It is defined as follows:

$$\text{recall} = \frac{\|A\&\tilde{A}\|_F^2}{\|A\|_F^2} = \frac{|\{(i,j):\ A(i,j) = \tilde{A}(i,j) = 1\}|}{|\{(i,j):\ A(i,j) = 1\}|}. \tag{7}$$

We use precision and recall to compare the approximations, A_s and A_p, which are provided by serial and parallel formulation, respectively. Figure 11(a) shows precision

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

Number of processors	Run time	Number of patterns
1	76.78	189
2	32.55	197
4	14.46	242
6	8.68	288
8	7.25	322
10	4.80	361
12	3.90	390

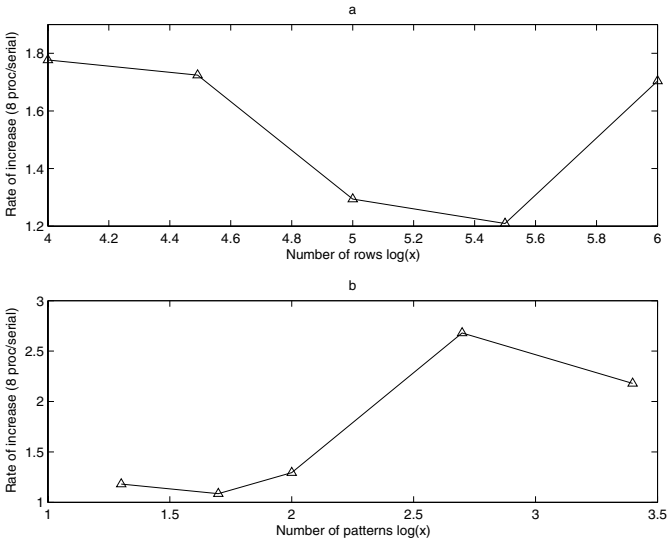


Fig. 10. The increase rate of the number of discovered patterns by the parallel algorithm with (a) an increasing number of rows and (b) an increasing number of underlying patterns.

and recall comparing A with A_s and A with A_p with an increasing number of rows in the original dataset. Figure 11(b) shows the same comparison for increasing number of underlying patterns in the original dataset. These comparisons show that in general precision and recall of the parallel approximation closely follow those of the serial approximation. Notable differences include lower precision for a dataset with a small number of rows (10K), and lower precision but higher recall for a dataset with a very large number of patterns (2500 patterns in 100K rows). These deviations are consistent with the expected variance as a result of sub-sampling. As the size of a dataset decreases or the number of underlying patterns in a dataset increases, each pattern is supported by fewer rows, which results in higher variations in the patterns discovered. In addition, we also evaluate precision and recall for serial and parallel approximations with a varying bound on the Hamming distance (1–5). As is evident in Figure 11(c), the precision and recall levels of serial and parallel approximations closely match and stabilize as the bound on the Hamming radius increases.

5.2.3. *Bandwidth Requirement*. We now examine the bandwidth requirement of CONQUEST with a view to justifying its coarse-grained nature. The coarse-grained nature of CONQUEST's parallel formulation alleviate much of the bandwidth requirement imposed by traditional fine-grained formulation. Figure 12(a) presents a lower bound on the communication volume of a fine-grained parallel formulation derived from optimal parallelizations of underlying kernels and the coarse-grained formulation adopted in CONQUEST. Reported values for the total volume of communication for the fine-grain parallel algorithm are conservative estimates that take into account only the repeated mat-vecs. We assume a one-dimensional mapping of the matrix based on rows, which is the most suitable for our application. We compute a lower-bound on the total volume of

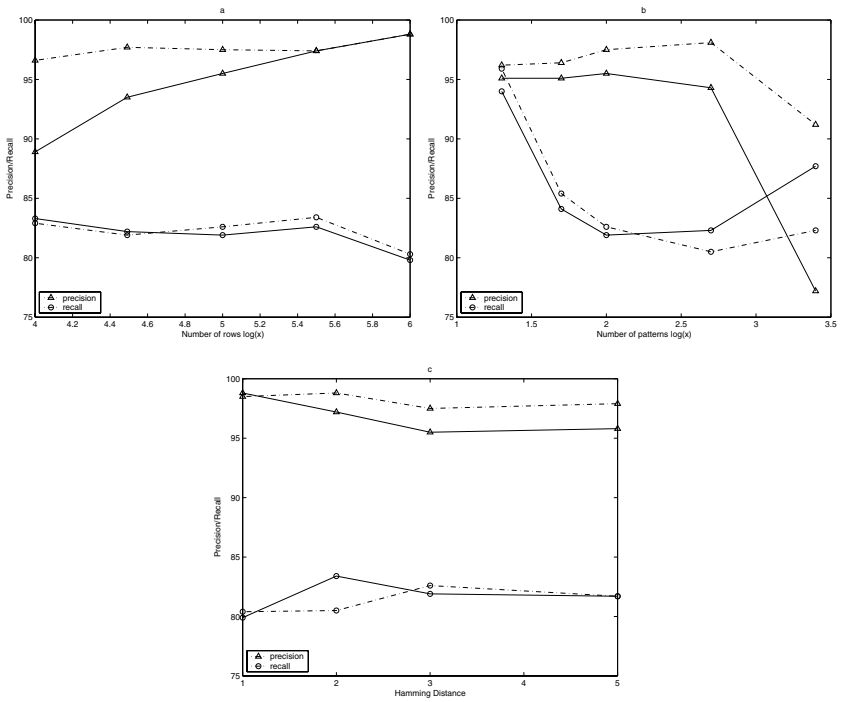


Fig. 11. Comparison of serial (dotted line) and parallel (solid line) approximations in terms of precision and recall with respect to the original matrix with (a) an increasing number of rows, (b) an increasing number of underlying patterns, and (c) an increasing bound on the Hamming radius.

communication optimized by a min-cut graph model using the following formula:

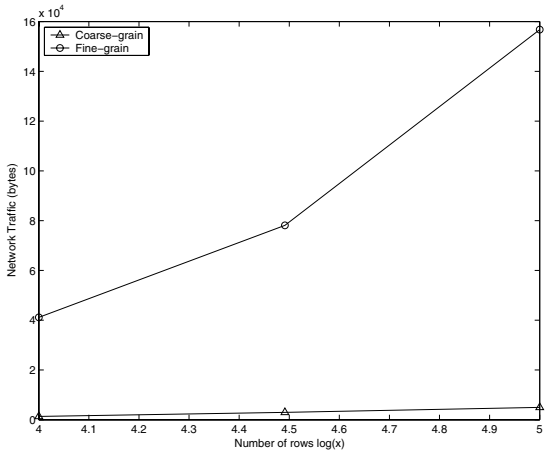
$$V_f = 4 \sum_{R} \#\text{ of iterations } \times \frac{\#\text{ rows in submatrix} \times \text{cutsize}}{\#\text{ rows in original matrix}}. \tag{8}$$

Here R denotes the set of all rank-one approximation computations. Using a hypergraph model, we compute a good cutsize using the hypergraph partitioning tool hMeTiS [39], which is equal to the number of vector elements to be communicated during each mat-vec (note again that the process of deriving a min-cut partition is more expensive than CONQUEST itself!). We compute a conservative approximation for each sub-matrix by averaging over the number of rows, which is an optimistic approximation to the cutsize of the sub-matrix being approximated since the matrices that appear deeper in the recursion tree are denser than the original matrix. As the mat-vec is repeated through iterations, we multiply this number by the number of iterations in that computation. The factor of 4 in the equation is for conversion to bytes since integer vectors need to be transferred in a fine-grain formulation. Note that, in contrast, only bit-vectors are transferred in CONQUEST.

We present only the matrices with 10K, 31K, and 100K matrices since the hypergraph partitioning the problem's complexity exceeds that of sequential application of PROXIMUS, making the minimum cut-based optimization infeasible.

# of rows	size×iter.	cutsize	Total Communication fine-grained	coarse-grained
10K	27	381	41148	1360
31K	31	630	78120	2981
100K	46	852	156768	4988
316K	-	-	-	12935
1M	-	-	-	38417

(a)



(b)

Fig. 12. (a) Lower bound on communication volume of a hypothetical fine-grained algorithm compared with the traffic volume generated by the execution of CONQUEST with the number of rows ranging from 10K to 1M. (b) Comparison of the amount of network traffic generated by CONQUEST and fine-grained algorithms for the first three datasets.

These results provide clear contrast between the fine-grained approach and the coarsed-grained approach adopted by CONQUEST. Even with a conservative estimation outlined above, a fine-grained parallel formulation would generate at least 150 KB network traffic for the analysis of a dataset of 100K rows. In contrast, the execution of CONQUEST generates only 5 KB of network traffic for the same analysis. Even the heaviest traffic, generated during the execution on a dataset of 1 million rows, is about 38 KB, which is well within the capability of any wide area network (e.g., it takes milliseconds to transfer 38 KB of data on a 1 Mbps network link). Thus, we can safely conclude that the time it takes to transfer the small amount of traffic 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 summarizing large amounts of data distributed across various sites. The objective is use this summary for mining association rules using a conventional algorithm, sequentially. This is generally more efficient than collecting all

of the data at one site or running a 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 [25]. Given a set of rows over a set of items, association rule mining aims to discover rules in 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-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 six transactions as shown in Figure 13(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 13(b). As shown in Figure 14(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 14(b)). We can now analyze this smaller approximate transaction set using any existing association rule-mining technique. Note that this summary can be constructed using CONQUEST if the transactions are distributed among various sites.

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)

$T =$

	beer	snacks	bread	milk	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

(b)

Fig. 13. (a) A sample transaction set of six transactions on five 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 transactions	Weight
T_1' : {bread, milk, butter}	4
T_2' : {beer, snacks, bread}	2

(b)

Fig. 14. (a) Decomposition of the transaction matrix of the transaction set of Figure 13 and (b) the corresponding approximate transaction set.

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 [40] 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 eight 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 the 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

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

Min. support (%)	Time orig. (sec.)	Time approx. (sec.)	Rules orig. (#)	Rules approx. (#)	Rules match (#)	Rules match	
						Precision (%)	Recall (%)
2.0	28.45	0.38	63,503	53,280	53,280	100.0	91.0
2.5	12.69	0.14	27,090	20,299	20,299	100.0	75.0
3.0	10.99	0.14	20,353	19,527	19,527	100.0	98.1
3.5	10.56	0.13	19,617	19,527	19,527	100.0	99.5
4.0	9.23	0.08	12,793	12,793	12,793	100.0	100.0
4.5	9.27	0.08	12,787	12,789	12,787	100.0	99.98
5.0	8.98	0.06	12,398	10,955	10,955	100.0	90.9
5.5	7.20	0.05	6,740	6,732	6,732	100.0	99.88
6.0	7.12	0.05	6,732	6,732	6,732	100.0	100.0

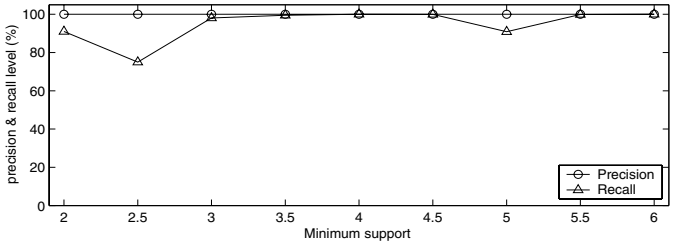


Fig. 15. Precision and recall level with varying minimum support.

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 in recalling the rules that are present in the original data.

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 15), 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 the Quest data generator). While maintaining high accuracy, CONQUEST provides a speedup 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 (eight processors), 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 pre-processor 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. CONQUEST adopts a coarsed-grained parallel formulation designed to conserve network bandwidth and alleviate the problem of load balancing faced in fine-grained 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 speedups while maintaining high accuracy. Finally, we demonstrate the application of CONQUEST in association rule mining as a powerful pre-processing tool to accelerate significantly existing data-mining software.

References

[1] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In E. Simoudis, J. Han, and U. M. Fayyad, editors, *Proc*. 2*nd Internat. Conf. on Knowledge Discovery and Data Mining* (*KDD*), pages 367–370. AAAI Press, Menlo Park, CA, 1996.

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

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

[4] H. Toivonen. Sampling large databases for association rules. In *Proc.* 22*nd Internat. Conf. on Very Large Databases* (*VLDB* '96), pages 134–145, 1996.

[5] T. G. Kolda and D. P. O'Leary. Computation and uses of the semidiscrete matrix decomposition. *ACM Transactions on Mathematical Software*, 26(3):416–437, 2000.

[6] 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.

[7] M. T. Chu and R. E. Funderlic. The centroid decomposition: relationships between discrete variational decompositions and SVDs. *SIAM Journal on Matrix Analysis and Applications*, 23(4):1025–1044, 2002.

[8] 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.

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

[10] B. Park and H. Kargupta. Distributed data mining: algorithms, systems, and applications. In Nong Ye, editor, *The Handbook of Data Mining*, pages 341–358. Erlbaum, Hillsdale, NJ, 2003.

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

[12] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining*, pages 245–260. Lecture Notes in Artificial Intelligence, Vol. 1759. Springer, Berlin, 2000.

[13] D. Foti, D. Lipari, C. Pizzuti, and D. Talia. *Scalable Parallel Clustering for Data Mining on Multicomputers*. Lecture Notes in Computer Science, vol. 1800. Springer, Berlin, 2000.

[14] 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.

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

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

[17] R. Agrawal and J. C. Shafer. Parallel mining of association rules: design, implementation and experience. Technical Report, IBM Almaden Research Center, IBM Corp., San Jose, CA, 1996.

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

[19] E. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):337–352, 2000.

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

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

[22] T. Shintani and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proc. Fourth International Conference on Parallel and Distributed Information Systems*, pages 19–30, Miami Beach, FL, 1996.

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

[24] 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*, Article No. 43, 1996.

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

[26] 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.* 22*nd Internat. Conf. on Very Large Databases* (*VLDB*), pages 544–555. Morgan Kaufmann, San Mateo, CA, 1996.

[27] M. Joshi, G. Karypis, and V. Kumar. ScalparC: a new scalable and efficient parallel classification algorithm for mining large datasets. In *Proc*. 11*th International Parallel Processing Symposium*, page 573. IEEE Computer Society Press, Los Alamitos, CA, 1998.

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

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

[30] M. Koyutürk, A. Grama, and N. Ramakrishnan. Non-orthogonal decomposition of binary matrices for bounded-error data compression and analysis. *ACM Transactions on Mathematical Software*, 32(1), 2006.

[31] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*: *Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, CA, 1994.

[32] B. Hendrickson, R. Leland, and S. Plimpton. An efficient parallel algorithm for matrix-vector multiplication. *International Journal of High Speed Computing*, 7(1):73–88, 1995.

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

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

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

[36] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[37] Message Passing Interface Forum. MPI: a message-passing interface standard. Technical Report UT-CS-94-230, 1994.

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

[39] 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.

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