Communication-Efficient Distributed Mining of Association Rules

Assaf Schuster and Ran Wolff
Computer Science Dept.
Technion - Israel Institute of Technology
{assaf,ranw}@cs.technion.ac.il

24th November 2002

Abstract

Mining for associations between items in large transactional databases is a central problem in the field of knowledge discovery. When the database is partitioned among several share-nothing machines, the problem can be addressed using distributed data mining algorithms. One such algorithm, called CD, was proposed by Agrawal and Shafer and was later enhanced by the FDM algorithm of Cheung, Han et al. The main problem with these algorithms is that they do not scale well with the number of partitions. They are thus impractical for use in modern distributed environments such as peer-to-peer systems, in which hundreds or thousands of computers may interact.

In this paper we present a set of new algorithms that solve the Distributed Association Rule Mining problem using far less communication. In addition to being very efficient, the new algorithms are also extremely robust. Unlike existing algorithms, they continue to be efficient even when the data is skewed or the partition sizes are imbalanced. We present both experimental and theoretical results concerning the behavior of these algorithms and explain how they can be implemented in different settings.

1 Introduction

1.1 Problem Description

Association Rules Mining (ARM) in large transactional databases is a central problem in the field of knowledge discovery. The input to ARM is a database in which objects are grouped by context. An example of such a grouping would be a list of items grouped by the customer who bought them. ARM then requires us to find sets of objects which tend to associate with one another. Given two distinct sets of objects, X and Y, we say Y is associated with X if the appearance of X in a certain context usually implies that Y will appear in that context as well. If X usually implies Y, we then say that the rule $X \Rightarrow Y$ is confident in the database. We would usually not be interested in an association rule unless it appears in more than a certain fraction of the contexts: if it does, we say that the rule is frequent. The thresholds of frequency (MinFreq) and confidence (MinConf) are parameters of the problem and are usually supplied by the user according to his needs and resources. The solution to the ARM problem is a list of all association rules which are both frequent and confident in that database. Such lists of rules have many applications in the context of understanding, describing and acting upon the database.

The ARM problem has been investigated intensively during the past few years ([2, 11, 8, 15, 4, 9, 14, 3] being a small sample). It was shown that the major computational task is the identification of all the frequent itemsets, those sets of items which appear in a fraction greater than MinFreq of the transactions. Association rules can then be produced from these frequent itemsets in a fairly straightforward manner. Once it is known that both $\{Pasta\,Sauce\}$ and $\{Pasta\,Sauce, Parmesan\}$ are frequent itemsets for

instance, the association rule $\{Pasta\ Sauce\} \Rightarrow \{Parmesan\}$ is obviously frequent, and all that remains is to check if the association is confident.

ARM algorithms are mainly concerned with reducing the number of database scans. This reduction is necessary because the database is usually very large and is stored in secondary memory (disk). The problem was restated in a distributed setting as the Distributed Association Rules Mining (D-ARM) problem. The main reason for restating the problem this way was to parallelize the disk I/O required to solve it. In D-ARM, the database is partitioned between several parties which can perform independent parallel computations as well as communicate with one another. Several algorithms were proposed to solve D-ARM, most of them for share-nothing machines [1, 7, 6], and some for shared-memory [16] or distributed shared-memory machines [10]. Since the parallelization of disk I/O is itself an easy task, the main show stopper for D-ARM algorithms is communication complexity. The most important factors in the communication complexity of D-ARM algorithms turn out to be the number of partitions, n, and $|C|$, the number of itemsets considered throughout the algorithm.

In this paper we present a set of new D-ARM algorithms for share-nothing machines. These algorithms improve the communication complexity of the best of the known D-ARM algorithms. We prove that our algorithms are the first to solve the problem with a communication complexity that is linear in n and $|C|$, and with a very small database-dependent multiplicative factor. When compared with current algorithms, our algorithms require a fraction of the communication bandwidth even for mid-range values of n. For higher n values, we prove that the advantage of our algorithms keeps growing at the same rate.

We see possible applications for our communication-efficient algorithms in three main areas. First, they may be used to mine peer-to-peer systems. For example, we may wish to find associations between the mp3 files of different Napster users (more than 1.5 million files in about 10,000 libraries at the time this paper was written). No previous algorithm can cope with $n = 10,000$ with the Internet communication speed available today. Second, we may use these algorithms for broad-scale parallelization of data mining, splitting the problem until each partition fits into the memory of a conventional PC. Third, we can use them in environments where communication bandwidth is an expensive resource, such as billing centers for large communication providers. Although these billing centers usually have fast and wide networks, data mining is performed there as an auxiliary task and the resources it consumes come at the expense of the main system activity.

1.2 Previous Work

The two major approaches to D-ARM as presented in [1] are data distribution (DD) and count distribution (CD). DD focuses on the optimal partitioning of the database in order to maximize parallelism. CD, on the other hand, considers a setting where the data is arbitrarily partitioned horizontally[1] among the parties to begin with, and focuses on parallelizing the computation.

The DD approach is not always applicable. At the time the data is generated, it is often already partitioned. In many cases it cannot be gathered and repartitioned for reasons of security and secrecy, cost of transmission, or sheer efficiency. DD is thus more applicable for systems which are dedicated to performing D-ARM. CD, on the other hand, may be a more appealing solution for systems which are naturally distributed over large expanses, such as stock exchange and credit card systems. This article focuses on the CD approach to D-ARM, although the main ideas can probably be adapted to DD as well.

All the algorithms discussed in this paper are based on the Apriori algorithm [2]. Apriori begins by assuming that any item is a candidate to be a frequent itemset of size 1. Apriori then performs several rounds of a two-phased computation. In the first phase of the k round, the database is scanned and the frequency of all k-sized candidate itemsets (itemsets containing k items) is calculated. Those candidate itemsets which have frequency above the user supplied MinFreq threshold are considered frequent itemsets. In the second phase, candidate $k+1$-sized itemsets are generated from the set of frequent k-sized itemsets if and only if all their k-sized subsets are frequent. The rounds terminate when

[1] Horizontal partitioning means that each partition includes whole transactions, in contrast with vertical partitioning where the same transaction would be split among several parties.

the set of frequent k-sized itemsets is empty.

The CD algorithm [1] is an obvious parallelization of Apriori. In the first phase, each of the parties performs the database scan independently on its own partition. Then a global sum reduction is performed on the support counts of each candidate itemset. Those itemsets whose global frequency is larger than MinFreq are considered frequent. The second phase of calculating the candidate $k+1$-sized itemsets can be carried out without any communication because the calculation depends only on the identity of the frequent k-sized itemsets, which is known to all parties by this time. CD fully parallelizes the disk I/O complexity of Apriori and performs roughly the same computations. CD also requires one synchronization point on each round and carries an $O(|C|\, n)$ communication complexity penalty, where C is the group of all candidate itemsets considered by Apriori and n is the number of parties. Since typical values for $|C|$ are hundreds of thousands, CD is obviously not scalable to large numbers of partitions.

FDM [6] offers a way by which the communication load of CD may be reduced. It takes advantage of the fact that ARM algorithms only look for rules which are globally frequent. FDM is based on the inference that in order for an itemset to appear in a certain portion of the transactions in the database, it must appear in at least that portion of at least one partition of the database. In FDM, the first stage of CD was divided into two rounds of communication. In the first round, every party names those candidate itemsets which are locally frequent in its partition (e.g., appear in the partition with a frequency greater than or equal to MinFreq). In the second round, counts are globally summed for those candidate itemsets which were named by at least one party. If the probability that an itemset will have the potential to be frequent is $Pr_{potential}$, then FDM only communicates $Pr_{potential}\,|C|$ of the itemsets and improves the communication complexity to $O(Pr_{potential}\,|C|\, n)$.

The main problem of FDM is that $Pr_{potential}$ is not scalable in n. In the discussion in section 2.2, we show that $Pr_{potential}$ quickly increases to 1 as n increases. The convergence to 1 is especially fast in nonhomogeneous databases. Cheung and Xiao presented this problem in [5] and showed that as the nonhomogeneity of the database (as measured by the skewness measure they presented) increases, FDM pruning techniques become ineffective. We extend their results by proving that even for moderate nonhomogeneity FDM becomes ineffective for a large enough n.

1.3 Our Solution

We first look at the distributive problem of deciding whether an itemset's global frequency is above or below a threshold. If we first solve this problem for the set of candidate k-sized itemsets, then support counts of those identified as frequent can be collected optimally, while support counts for the infrequent itemsets can remain uncollected. We show that solving this distributed agreement problem causes very little overhead.

We use this approach to propose a new family of Apriori-based D-ARM algorithms that improve the communication complexity of current solutions. These algorithms improve data skew robustness and scalability over large numbers of partitions. We show three examples for such algorithms, all of which retain the good time, space and disk I/O complexities of Apriori. Our algorithm for the decision problem has $O(Pr_{above}\,|C|\, n)$ communication complexity, where Pr_{above} is the probability that the frequency with which a candidate itemset will appear in a specific partition is above MinFreq. Pr_{above} is, by definition, smaller than or equal to $Pr_{potential}$. Unlike $Pr_{potential}$, Pr_{above} has no dependency on n and is only dependent on the nonhomogeneity of the database. As a result, our algorithms are the first to demonstrate low linear dependency of communication complexity on n and thus, scalability.

After the decision problem was solved, the obvious next step would have been to collect the support counts of the frequent itemsets from all parties with linear output complexity of $O(|L|\, n)$, where L is the group of frequent itemsets. However, we show that further improvement is possible: the confident rules can be identified with sublinear communication complexity if another decision problem is solved. This time the decision problem is concerned with the confidence of the rules generated from the frequent itemsets. By applying our algorithm to this problem too, we demonstrate that it is general and can be implemented for many functions of the database.

We complete this introduction with some notations and a description of our basic assumptions. The following section describes an algorithm called Distributed Decision Miner (DDM). This algorithm

demonstrates our basic approach. We then propose two additional derivatives of DDM, to be described in sections 3 and 4: Preemptive Distributed Decision Miner (PDDM) and Distributed Dual Decision Miner (DDDM). These two algorithms further improve the communication complexity of Distributed Decision Miner. Moreover, the improvements are complementary and we can combine them to get even better results. The combination will not be presented in this context. Section 5 describes in detail the experiments we carried out to verify the algorithm's superiority. We conclude the article in section 6.

1.4 Notations

Let $I = \{i_1, i_2, ..., i_m\}$ be the items in a certain domain. An itemset is some $X \subseteq I$. A transaction t is also a subset of I associated with a unique transaction identifier TID. A database DB is list of such transactions. Let $\overline{DB} = \left\{DB^1, DB^2, ..., DB^n\right\}$ be a partition of DB into n partitions with sizes $\overline{D} = \left\{D^1, D^2, ..., D^n\right\}$ respectively. For any itemset X and any group of transactions G let $Support\,(X, G)$ be the number of transactions in G which contain all the items of X and let $Freq\,(X, G) = \frac{Support(X,G)}{|G|}$. We call $Freq\left(X, DB^j\right)$ the local frequency of X in partition j and $Freq\,(X, DB)$ its global frequency.

For some frequency threshold $0 \leq MinFreq \leq 1$, we say that an itemset X is frequent in a database G if $Freq\,(X, G) \geq MinFreq$ and infrequent otherwise; if G is a partition we say that X is locally frequent and if G is the whole database then X is globally-frequent. The group of all itemsets with frequency above of equal to $MinFreq$ is called $L\,[G]$, where G may be the full database or a partition thereof. Finally, Let X, Y be two globally-frequent itemsets such that $X \subset Y$, and let $0 < MinConf \leq 1$ be some confidence threshold, we say the rule $X \Rightarrow Y \setminus X$ is confident iff $Freq\,(Y, DB) \geq MinConf \cdot Freq\,(X, DB)$.

Given a partitioned database $\overline{DB}$, $MinFreq$ and $MinConf$ the D-ARM problem is to find all the rules of the form $X \Rightarrow Y$ in DB.

The messages the parties send to one another contain pairs $\left\langle i, x_i^j \right\rangle$, where i is an itemset number and $x_i^j = Support\left(X_i, DB^j\right)$. We will assume that j, the origin of the message, can be inferred with negligible communication costs. For each party p and itemset X_i, let $G^p\,(X_i)$ be the group of all x_i^j such that $\left\langle i, x_i^j \right\rangle$ was received by p. We will assume $G^p\,(X_i)$ is equal for all p and refer to it as $G\,(X_i)$. $\overline{D}$ is either known to all parties in advance or can be exchanged in the first n messages.

1.5 Basic Assumptions

Distributed algorithms may vary significantly according to the nature of the distributed system under consideration. It cannot be expected that an algorithm which is optimal for a system containing just two computing nodes would also be optimal for a system containing 64 computing nodes. Thus, although it is possible to present such an algorithm with no reference to a specific architecture (as we did in [12]), it is also interesting to put forth some assumptions about that architecture and see how they affect the algorithm. We make three major assumptions in this paper: that communication bandwidth is limited, that the communication layer supports broadcast, and that the communication layer implicitly buffers messages to a fixed size. The latter assumption distinguishes this work from the one described in [12] because, as we show, it has significant performance implications.

The first assumption can be easily justified. It is very tempting to distribute an ARM algorithm because such distribution leads to a linear reduction in disk I/O and disk I/O is traditionally the bottleneck of ARM algorithms. Disk I/O could be reduced by increasing the number of computing nodes and dividing the database among the nodes. However, this will lead to a linear increase in communication costs which will result in diminishing, eventually negative, speedups. We will hence assume that communication bandwidth is the bottleneck of the algorithm.

The second assumption, that the network supports broadcast, can be justified by the fact that LAN and SAN networks usually do. The three most popular networks, Ethernet (IEEE-802.3), Fast Ethernet (such as Myrinet and ServerNet), and to a lesser degree Token Ring, support cost-effective broadcast.

The final assumption – implicit buffering – is introduced for clarity. Messages usually have a fixed size, which is typically 96 bytes for Fast Ethernet and 1500 bytes for Ethernet . If the application 'sends'

larger messages then they are broken down and recomposed; if the application 'sends' smaller messages they may be padded to reach the fixed size. Since the application level messages in our algorithms usually contain only several bytes, we assume that the communication layer stacks them until full message size is reached and then sends a full message. This feature can be written as part of the implementation of the algorithm if it is not supported by the communication layer.

2 The Distributed Decision Miner Algorithm

Algorithm 1 Distributed Decision Miner

For node j out of n

1. Initialize $C_1 = \{\{i\} : i \in I\}$, $k = 1$, $Passed = \emptyset$
2. While $C_k \neq \emptyset$
   (a) Do
   - Choose an itemset $X_i \in C_k$ which was not yet chosen and for which either $H(X_i) < MinFreq \cdot D \leq P(X_i, DB^j)$ or $P(X_i, DB^j) < MinFreq \cdot D \leq H(X_i)$, and broadcast $\langle i, Support(X_i, DB^j) \rangle$.
   - If no such itemset exists, broadcast $\langle pass \rangle$.

   (b) Until $|Passed| = n$.
   (c) $L_k = \{X_i \in C_k : H(X_i) \geq MinFreq \cdot D\}$.
   (d) Broadcast the support counts for every $X_i \in L_k$ that was never chosen.
   (e) $C_{k+1} = Apriori\_Gen(L_k)$.
   (f) $k = k + 1$.
3. $Gen\_Rules(L_1, L_2, ..., L_k)$

When node j receives a message M from node p:

1. If $M = \langle pass \rangle$, insert p into $Passed$
2. Else if $|Passed| = n$ then M is the support counts of itemsets p has not yet sent. Update accordingly.
3. Else $M = \langle i, Support(X_i, DB^p) \rangle$
   - If $p \in Passed$ then remove p from $Passed$
   - Recalculate $H(X_i)$ and $P(X_i, DB^j)$

The basic idea of the Distributed Decision Miner (DDM, Alg. 1) algorithm is to verify that an itemset is frequent before collecting its support counts from all parties. The algorithm differs from FDM in that, in our algorithm, the fact that an itemset is locally frequent in one partition is not considered sufficient evidence to trigger the collection of all the support counts for that itemset. Instead, the parties perform some kind of negotiation by the end of which they are able to decide which candidate itemsets are globally frequent and which are not. The rest is straightforward: the support counts of the frequent itemsets are collected optimally, with no communication wasted on globally infrequent, but locally frequent itemsets.

The parties negotiate by exchanging messages containing local support counts for various itemsets. At any given stage, a common hypothesis H is shared by all parties. This hypothesis concerns the global support of every candidate itemset. Given all the local support counts for an itemset, this hypothesis

must correctly predict whether it is frequent or infrequent. In addition, every party computes another private hypothesis P, based on both the support counts already expressed and the party's local support count for the candidate itemset. For at least one party which has not yet expressed its local support count, and given any subset of the support counts for an itemset, the local hypothesis must correctly predict whether the itemset is frequent or infrequent.

2.1 The Algorithm

The parties calculate the set of candidate itemsets for which they did not express their local support counts. For each such candidate, every party calculates the global hypothesis H and the local hypothesis P. If H and P disagree on whether a candidate itemset is frequent or infrequent, then the support count for that candidate should be expressed. The parties express support counts at a certain rate, limited by the bandwidth of the system. Instead of sending one message concerning all the candidates, the parties send smaller messages concerning one or several candidates. No synchronization is required for single messages. Every time a party receives a message, it updates H and P for the candidate itemsets referred to in that message.

If, for some party, H and P agree for every candidate itemset, that party has nothing to express and it passes on its turn. A party may resume sending messages if arriving messages cause disagreement between H and P for some yet unexpressed candidate itemset. If a full round of passes was received from all parties, then H and P of all parties agree on every candidate itemset. By the definition of P, there are two possibilities for each candidate itemset: either there is one party whose P correctly predicts the itemset size or all the local support counts have been collected. In the first case, the H and P of the party whose P correctly predicts the itemset size must agree; since all parties compute the same H, that H must be correct for all parties. In the second case, H must be correct by definition.

Formally, we define H and P as follows:

$$H\left(X_i\right) = \begin{cases} 0 & if \quad G\left(X_i\right) = \emptyset \\ \frac{\sum_{x_i^p \in G\left(X_i\right)} x_i^p}{\sum_{x_i^p \in G\left(X_i\right)} D^p} \cdot D & otherwise \end{cases}$$

$$P\left(X_i, DB^j\right) = \sum_{x_i^p \in G(X_i)} x_i^p + \frac{x_i^j}{D^j} \cdot \sum_{x_i^p \notin G(X_i)} D^p.$$

With H, the parties assume that the unexpressed support counts for each itemset are, on the average, the same as those already expressed. With P, on the other hand, a party assumes that those parties which have not yet expressed their local support counts for that itemset have the same relative support as it does.

As defined, the above assumptions are not required to hold for every party. It is enough that the assumption on H will hold eventually and that the assumption on P holds for one party which has not yet expressed its support count. This is easily proved: Out of all the parties which have not yet expressed support ($x_i^p \notin G\left(X_i\right)$), the one with the largest relative support $\frac{x_i^p}{D^p}$ computes a value for P which is an upper bound on the global support count of X_i, and the one with the lowest relative support computes a value for P which is a lower bound on the global support count of X_i. It follows that at least one of those two must always estimate the global support count correctly, thus satisfying the requirement for P. As for H, when all the support counts have been collected, H is equal to the global support count. Thus, the requirement from H is also satisfied.

Usually each party can choose which of several candidate itemsets will have its support count sent next. Many heuristics can be used to break ties, but we found one to be most effective. Our tie breaking heuristic is based on the following rationale: whenever two parties are able to express the local support counts of the same candidate itemset, it is best if the one which makes a greater change in P expresses its local support first. If there are opposing parties for a candidate itemset (some of whose P is larger and others whose P is smaller than $MinSup \cdot D$), then the one that makes the greater change has the better chance to "convince" opposing parties that they are wrong. If the opposing parties' P is changed

to the extent that it now agrees with that of the sending party, the opposing parties will refrain from expressing their own support and thus save the cost of messages. It is therefore a good strategy for a party to send those support counts which will cause the greatest change in the Ps of opposing parties, for those same itemsets.

When party k expresses support for itemset X_i, the influence on P of party l is equal to $\left| x_i^k - \frac{x_i^l}{D^l} \cdot D^k \right|$. However, since x_i^l has not yet been expressed , we estimate the change as $R\left(X_i, DB^k\right) = \left| x_i^k - \frac{H(X_i)}{D} \cdot D^k \right|$. We will thus break a tie by choosing those itemsets which have the maximal $R\left(X_i, DB^j\right)$ value.

Figure 1 describes a running example of one itemset and four computing nodes. At first the itemset is considered infrequent because the global hypothesis is zero. Nodes A and B disagree because their local hypothesis is that the itemset is frequent. After some time this disagreement causes node B to send its local count. This changes both the global hypothesis and the local hypothesis of the other nodes. Now node A is satisfied but nodes C and D disagree. Additional messages follow, and by now node C has sent its local count too. Now, for both nodes A and D, the global and local hypothesis agree. Since nodes B and C have already expressed their local counts, they accept the global hypothesis; so no more information will be sent with regards to this itemset even though the exact count is still unknown. An external viewer can indeed see that this itemset is infrequent.

2.2 Complexity Analysis

2.2.1 Communication Complexity

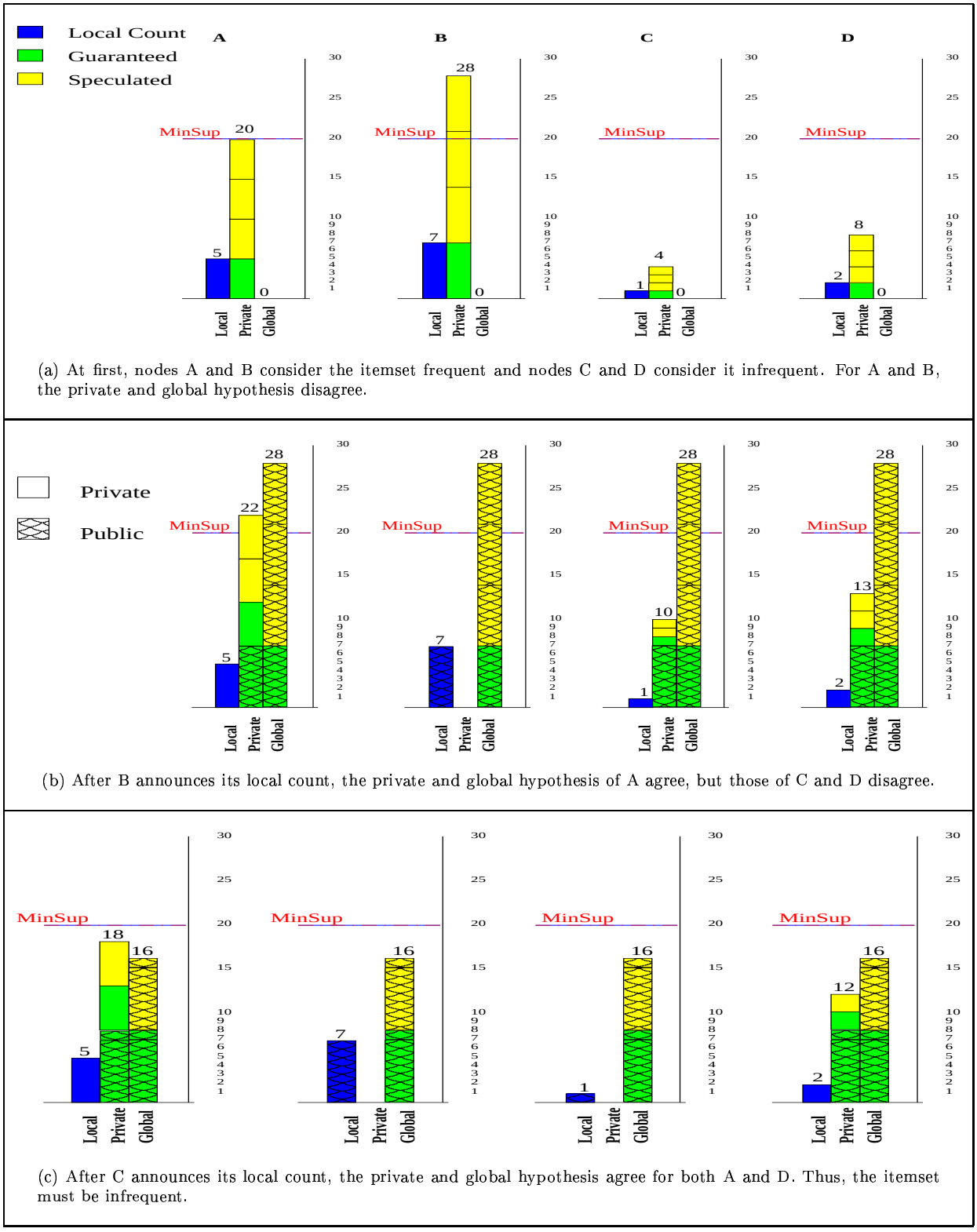
The messages sent by Distributed Decision Miner can be separated into two classes: messages relating to itemsets which eventually turn out to be frequent and messages relating to itemsets which eventually turn out to be infrequent. While the messages related to frequent itemsets are needed anyway for the calculation of confidence, those relating to infrequent itemsets are wasted. The wasted messages can also be separated into two classes: those which imply that the itemset being considered is frequent (strengthening evidence) and those which imply that the itemset is infrequent (weakening evidence).

By definition, every message containing strengthening evidence must be presented by a party with local frequency $Freq\left(X_i, DB^j\right) \geq MinFreq$. Thus, the expected number of such messages is $O\left(Pr_{above} \cdot |C| \cdot n\right)$, where n is the number of parties, C is the group of all itemsets considered by Distributed Decision Miner (which is equal to that considered by Apriori and FDM), and Pr_{above} is the probability that a specific itemset is locally frequent in a specific partition.

For a certain infrequent itemset, let the number of wasted messages containing strengthening evidence be s and the number of wasted messages containing weakening evidence be w. Let their average distances from $MinFreq$ be ϵ^s and ϵ^w accordingly. In the worst case, all the possible strengthening evidence for that infrequent itemset is collected. If we remove the last message containing weakening evidence, we can be sure that $H\left(X_i\right) \geq MinFreq \cdot D$; otherwise, the message containing it would never have been sent. We assume, for simplicity, that the partitions have the same size and that the final weakening message was of average distance from $MinFreq$. We then have $(s + w - 1) \cdot MinFreq < s \cdot (MinFreq + \epsilon^s) + (w-1) \cdot (MinFreq - \epsilon^w)$. It follows that $w < s \cdot \frac{\epsilon^s}{\epsilon^w}$. ϵ^s is only dependent on the distribution and not on C or n (the variance of ϵ^s decreases with n). ϵ^w can only increase with n because the algorithm has more possible messages with weakening evidence to choose from and it tends to choose those with more extreme values. Thus, the number of such messages has linear or lower dependency on the number of messages containing strengthening evidence, and the total communication complexity is $O\left(Pr_{above} \cdot |C| \cdot n\right)$.

By comparison, the communication complexity of FDM is $O\left(Pr_{potential} \cdot |C| \cdot n\right)$, where $Pr_{potential}$ is the probability that any of the partitions has relative support larger than $MinSup$. If we assume that the support counts of an infrequent itemset in different partitions are independent, then $Pr_{potential} = 1 - (1 - Pr_{above})^n$. This converges to 1 very quickly, even for a small Pr_{above}. It follows that for a large number of partitions, FDM performs as poorly as CD in terms of communication.

From the analysis of FDM and DDM communication complexity, it is clear that DDM can be as much as $\frac{Pr_{potential}}{Pr_{above}}$ times more efficient than FDM. This ratio is a property of the database, but it clearly



(a) At first, nodes A and B consider the itemset frequent and nodes C and D consider it infrequent. For A and B, the private and global hypothesis disagree.

(b) After B announces its local count, the private and global hypothesis of A agree, but those of C and D disagree.

(c) After C announces its local count, the private and global hypothesis agree for both A and D. Thus, the itemset must be infrequent.

Figure 1: The figures above show how the algorithm may treat a single itemset. In this example DDM discovers that the itemset is infrequent using just two messages (comparing to 6 in FDM).

grows with n. Hence, DDM is more scalable than FDM.

2.2.2 Computation Complexity

The computations FDM and DDM perform are actually quite similar. There are three important differences: In FDM a single machine (the polling station) performs most of the computation, whereas in DDM all the machines perform the same computation. In FDM the selection of the itemsets which should be reported is simpler than in DDM. Finally and most importantly, the selection criteria have to be computed just once for each itemset in FDM, while in DDM reevaluation of these criteria may be required each time new information arrives on that itemset.

We found that a simple data structure – a priority queue – can be used to address these problems most efficiently. We insert the itemsets to the queue according to their $R\left(X_i, DB^j\right)$ value; this causes a penalty of $O\left(\log\left(|C|\right)\right)$. Each time new data about an itemset arrives (n times at most, but typically only a few times) the priority is changed (causing the $O\left(\log\left(|C|\right)\right)$ penalty again). The total computation complexity is $O\left(n\log\left(|C|\right)\right)$ times that of FDM in the worst case. In practice, because only few itemsets require more than a few updates, the run time of the two algorithms is almost the same.

2.2.3 Space Complexity

The cause for most of the difference between the space requirements of FDM and those of DDM is that DDM stores the indexes of nodes which already sent their local count. This can be done with n bits for each itemset – inflicting an $\times n$ increase in space complexity over that of FDM. However, since for most candidate itemsets only a few nodes ever send their local count, a less trivial implementation may reduce space requirements considerably.

3 Preemptive Distributed Decision Miner

Often it is the case that partitions are not equally important. One partition may be exceptionally large or it may contain data which is more significant (e.g., a frequent itemset will be even more frequent in that partition). For example, if each partition contains the data from a different store, then partitions which belong to superstores are obviously more significant than those belonging to grocery stores.

We would like to allow parties which have more convincing evidence (extreme support counts) to send their support counts at an earlier stage of the negotiation in the hope that their evidence will shorten negotiation time and reduce communication. Similarly, we would like parties which do not have convincing evidence to refrain from sending messages so as not to use bandwidth which can be better employed. As we showed in the previous section, the rating function $R\left(X_i, DB^k\right) = \left|x_i^k - \frac{H(X_i)}{D}\cdot D^k\right|$ gives the k^{th} party an estimation for the effectiveness of each of its possible messages. We would like the series of messages to have a constantly decreasing value of R. In this section we show that parties can coordinate their messages and come nearer to this goal if each of them weighs the importance of its information against that of information contributed by other parties.

3.1 The Algorithm

In an R-optimal negotiation, the received messages have decreasing R values. Generating such a series requires, however, global knowledge, which is not available to the parties. We achieve a near monotonously decreasing series of R values by selecting as a leader the party which sent the message with the maximal R. Each node tracks the leader's identity and the R value of the last message sent by the leader. We do not allow any other party to send messages unless the R of its message is greater than that of the last message sent by the leader. If some other party sends a message with an R greater than that of the leader, this party then replaces the leader.

Preventing other parties from sending messages does not affect the correctness of the algorithm because the algorithm still terminates in the same state. We must make sure, however, that a leader

Algorithm 2 Preemptive Distributed Decision Miner

For node j out of n

1. Initialize $C_1 = \{\{i\} : i \in I\}$, $k = 1$, $Passed = \emptyset$, $leader = j$, $last\_R = 0$
2. While $C_k \neq \emptyset$
   (a) Do
   - Choose an itemset $X_i \in C_k$ which was not yet chosen and for which either $H(X_i) < MinFreq \cdot D \leq P(X_i, DB^j)$ or $P(X_i, DB^j) < MinFreq \cdot D \leq H(X_i)$ and which maximizes $R(X_i, DB^j)$.
   - If such an itemset exists and either $R(X_i, DB^j) > last\_R$ or $leader = j$, broadcast $\langle i, Support(X_i, DB^j) \rangle$
   - Else broadcast $\langle pass \rangle$.

   (b) Until $|Passed| = n$
   (c) $L_k = \{X_i \in C_k : H(X_i) \geq MinFreq \cdot D\}$
   (d) Broadcast the support counts for every $X_i \in L_k$ that was never chosen.
   (e) $C_{k+1} = Apriori\_Gen(L_k)$
   (f) $k = k + 1$
3. $Gen\_Rules(L_1, L_2, ..., L_k)$

When node j receives a message M from node p:

1. If $M = \langle pass \rangle$ insert p into $Passed$
2. Else if $|Passed| = n$ then M is the support counts of itemsets p has not yet sent. Update accordingly.
3. Else $M = \langle i, Support(X_i, DB^p) \rangle$
   - If $p \in Passed$ then remove p from $Passed$
   - Recalculate $H(X_i)$ and $P(X_i, DB^j)$
   - If $leader = p$ then update $last\_R = R(X_i, DB^p)$
   - Else if $last\_R < R(X_i, DB^p)$
     - Update $last\_R = R(X_i, DB^p)$
     - Update $leader = p$

passes the leadership to another party when it decides to pass on its turn; otherwise, the algorithm might not terminate. Hence, each time the leader passes on its turn, all parties set the value of the leader's last R to zero. When the leader's last R is zero, any party that has any message to send will send it and a party that has no message to send will pass on its turn. It is easy to calculate the leader's R for the R proposed in the prior section.

It is interesting to note that R can be extended to include other properties of the sent message. For example, R can be used to encode information about the cost of sending the message, whether that be in time (e.g., smaller bandwidth for that party) or in money (if this message is sent, for example, over a WAP channel). Preemptive Distributed Decision Miner (Alg. 2) will try to reach an R-optimal negotiation regardless of what R encodes.

3.2 Complexity Analysis

The complexity of Preemptive Distributed Decision Miner is dependent on the skewness of the database. In extreme cases where only one partition is significant, the communication can decrease to $O\left(Pr_{above} \cdot |C|\right)$ because only the party with that partition ever sends messages. On the other hand, when the partitions are very homogeneous, the parties may constantly compete over leadership. In this case, Preemptive Distributed Decision Miner reverts to Distributed Decision Miner, with communication complexity of $O\left(Pr_{above} \cdot |C| \cdot n\right)$.

If the time required for a party which receives a message to compute R of that message is $O\left(1\right)$, Preemptive Distributed Decision Miner will have the same time and space complexities as Distributed Decision Miner.

4 Distributed Dual Decision Miner

In their groundbreaking article [2], Agrawal and Srikant gave the following definition of ARM:

> "Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively."

Until now all the known algorithms actually gave, in addition to the list of rules, their respective support counts and confidence. As we will show here, in the distributed setting, we can detect whether rule support count and confidence are larger or smaller than the required minimum without ever fully calculating them.

The basic idea is that we can use a DDM-type algorithm to detect all frequent itemsets very efficiently. We need to collect the global support counts of the frequent itemsets if we wish to calculate the confidence of the rule. However, we must remember that our goal is only to decide if the confidence of a rule is above or below a given threshold, and not to find the exact confidence of the rule. This is exactly the same distinction we made when we presented Distributed Decision Miner. As we will show here, this second distributed decision problem can be solved in a similar manner.

To illustrate this idea, we present the following two examples:

1. Assume that $Parmesan$, $Pasta\,Sauce$ and $Parmesan \wedge Pasta\,Sauce$ are all globally frequent. The rule $Pasta\,Sauce \Rightarrow Parmesan$ should thus be considered. Assume also that this rule is locally frequent in every partition, but confident in none (e.g., $\frac{Support(Parmesan \wedge Pasta\,Sauce, DB^p)}{Support(Pasta\,Sauce, DB^p)} < MinConf$ for all p). Using Distributed Decision Miner, three messages are required to identify that both $Parmesan \wedge Pasta\,Sauce$ and $Pasta\,Sauce$ are significant (compared to $6n$ in FDM). With DDM and PDDM, we would need an additional $3\left(n-1\right)$ messages to collect the local support counts of the remaining parties for $Pasta \wedge Pasta\,Sauce$ and $Pasta\,Sauce$ before we could judge if $Pasta\,Sauce \Rightarrow Parmesan$ is significant. However, note that if for no party the local confidence is above $MinConf$, then the global confidence cannot be above $MinConf$. By implementing an algorithm similar to FDM we could have pruned this rule without sending a single message.

2. Assume that this same rule is both supported and confident in every partition. If one party suggests that the rule is globally confident and no other party objects, this is enough to determine that the rule is indeed globally significant.

Algorithm 3 Distributed Dual Decision Miner

For node j out of n

1. Initialize $C_1 = \{\{i\} : i \in I\}$, $k = 1$, $Passed = \emptyset$
2. While $C_k \neq \emptyset$
   (a) Do
   - Choose an itemset $X_i \in C_k$ which was not yet chosen and for which either $H(X_i) < MinFreq \cdot D \leq P\left(X_i, DB^j\right)$ or $P\left(X_i, DB^j\right) < MinFreq \cdot D \leq H(X_i)$ and broadcast $\left\langle i, Support\left(X_i, DB^j\right)\right\rangle$.
   - If no such itemset exists, broadcast $\langle pass \rangle$.

   (b) Until $|Passed| = n$
   (c) $L_k = \{X_i \in C_k : H(X_i) \geq MinFreq \cdot D\}$
   (d) $C_{k+1} = Apriori\_Gen(L_k)$
   (e) $k = k + 1$
3. $Mine\_Rules(L_1, L_2, ..., L_k)$

When node j receives a message M from node p:

1. If $M = \langle pass \rangle$ insert p into $Passed$
2. Else if $|Passed| = n$ then M is the support counts of itemsets p has not yet sent. Update accordingly.
3. Else $M = \langle i, Support(X_i, DB^p)\rangle$
   - If $p \in Passed$ then remove p from $Passed$
   - Recalculate $H(X_i)$ and $P\left(X_i, DB^j\right)$

In Distributed Dual Decision Miner (Alg. 3) we will generalize these two examples: first, by defining H and P for rules as well as for itemsets, and then by performing a negotiation similar to the one we performed for the support count to decide whether or not potential rules are confident.

4.1 The Algorithm

Distributed Dual Decision Miner runs any Distributed Decision Miner variant[2] to identify frequent itemsets without performing stage 2(d), the collection of yet uncollected support counts. Then, instead of calling the original *Gen_Rules* procedure, it uses another variant of DDM called Distributed Decision Confidence Miner to mine the set of rules with confidence above a user-defined threshold λ.

We first describe a simple algorithm (Alg. 4) which mines rules with large confidence and then, in subsection 4.3, introduce a rule pruning method which does away with the need to consider many of the rules. Distributed Decision Confidence Miner makes one round of negotiations to decide which of

[2]We use here, for reasons of clarity, Distributed Decision Miner. But Preemptive Distributed Decision Miner can be used as well.

the candidate rules $X_p \Rightarrow X_a \setminus X_p$ have $Freq(X_a, DB) \geq MinConf \cdot Freq(X_p, DB)$. The algorithm sends messages of three types, $\langle rule\_id, x_p^i, x_a^i \rangle$, $\langle rule\_id, x_p^i \rangle$ or $\langle rule\_id, x_a^i \rangle$, depending on which of the two support counts was expressed, where $rule\_id$ is the number of the rule in some deterministic enumeration and $x_j^i = Support\left(X_j, DB^i\right)$. Again we define H and P with the same limitations as they had in the previous algorithms.

Algorithm 4 Distributed Decision Confidence Miner

For node i of n nodes

1. Initialize
   - R_l to be the set of all rules $X_p \Rightarrow X_a \setminus X_p$ such that $X_p, X_a \in L$ and $X_p \subset X_a$.
   - $Passed = \emptyset$.
2. Do
   - Choose r_k to be some $X_p \Rightarrow X_a \setminus X_p \in R_l$ such that $i \neq G(r_k)$ and either $H(r_k) < MinConf \leq P\left(r_k, DB^i\right)$ or $P\left(r_k, DB^i\right) < MinConf \leq H(r_k)$.
     - If both $Support\left(X_p, DB^i\right)$ and $Support\left(X_a, DB^i\right)$ were not sent, broadcast $\left\langle k, Support\left(X_p, DB^i\right), Support\left(X_a, DB^i\right)\right\rangle$.
     - If $Support\left(X_p, DB^i\right)$ was already sent broadcast $\left\langle k, Support\left(X_a, DB^i\right)\right\rangle$.
     - If $Support\left(X_a, DB^i\right)$ was already sent broadcast $\left\langle k, Support\left(X_p, DB^i\right)\right\rangle$.
   - If there is no such r_k broadcast $\langle pass \rangle$.
3. Until $|Passed| = n$
4. $R = \{r_k \in R_l : H(r_k) \geq MinConf\}$

When node i receives a message M from node j:

1. If $M = \langle pass \rangle$ insert p to $Passed$.
2. Else $M = \left\langle k, Support\left(X_p, DB^j\right), Support\left(X_a, DB^j\right)\right\rangle$
   - If $i \in Passed$ remove i from $Passed$.
   - Recalculate $G(r_l)$ for every r_l which includes X_a and/or X_p. If $G(r_l)$ changes, update $H(r_l)$ and $P(r_l)$ as well.

The functions we define in Distributed Decision Confidence Miner are[3]:

$$G(X_p \Rightarrow X_a \setminus X_p) = \left\{j : x_p^j \in G(X_p) \wedge x_a^j \in G(X_a)\right\}$$

$$P\left(X_p \Rightarrow X_a \setminus X_p, DB^i\right) = \frac{\sum_{l \in G(X_p \Rightarrow X_a \setminus X_p)} x_a^l + (n - |G(X_p \Rightarrow X_a \setminus X_p)|) \cdot x_a^i}{\sum_{l \in G(X_p \Rightarrow X_a \setminus X_p)} x_p^l + (n - |G(X_p \Rightarrow X_a \setminus X_p)|) \cdot x_p^i}$$

$$H(X_p \Rightarrow X_a \setminus X_p) = \begin{cases} \frac{\sum_{l \in G(X_p \Rightarrow X_a \setminus X_p)} x_a^l}{\sum_{l \in G(X_p \Rightarrow X_a \setminus X_p)} x_p^l} & if \quad |G(X_p \Rightarrow X_a \setminus X_p)| > 0 \\ 0 & otherwise \end{cases}$$

$$R\left(X_p \Rightarrow X_a \setminus X_p, DB^i\right) = \left| \frac{x_a^i - \frac{H(X_a)}{D} \cdot D^i}{x_p^i - \frac{H(X_p)}{D} \cdot D^i} \right|$$

[3]Note that we use here both G of a rule and G of an itemset.

To prove that Distributed Decision Confidence Miner works, it is enough to show that H and P have the properties defined in 2.1. For one party which has not yet expressed both parts of a rule, P is required to be an upper bound on the global confidence, and for another it is required to be a lower bound. The requirement for H is that H is correct if all the data was gathered. Clearly, when considering the parties not in $G(r_l)$, the one with the maximal $\frac{x_q^i}{x_p^i}$ computes a value for P which is an upper bound on the confidence, and the one with the minimal $\frac{x_q^i}{x_p^i}$ computes a value for P which is a lower bound. Thus, the requirement from P is satisfied. In addition, when all the support counts of a rule are collected, H must be correct because it is equal to the confidence. Thus, the requirement for H is met.

4.2 Complexity Analysis

Distributed Decision Miner and Preemptive Distributed Decision Miner reduce communication complexity by reducing the communication required for the identification of infrequent itemsets. Distributed Dual Decision Miner, in contrast, reduces the communication by refraining from collecting local counts of frequent itemsets.

When discussing the theoretical bounds of Distributed Dual Decision Miner, one must note that the distribution of frequent itemsets over partitions in a real database may have a very special form. Simplistic assumptions about that distribution, which were acceptable for infrequent itemsets, may mislead us with regard to the actual savings in communication. Hence, we will say only that in the algorithm's worst case, all the support counts of all itemsets are collected in the itemset identification stage. In this case, Distributed Decision Confidence Miner is trivial and requires no communication at all. Thus, the algorithm's performance is exactly the same as that of Distributed Decision Miner. Further analysis will be done at the experimental level, in the next section.

4.3 Rule Pruning

The number of rules which can be generated from a given set of frequent itemsets is enormous. If we check all the potential rules induced by a single k-sized frequent itemset X, we would have to check every rule $Y \Rightarrow Z \setminus Y : Y \subset Z \subseteq X$. This is a total of $\sum_{i=1}^{k} \left(\begin{array}{c} k \\ i \end{array} \right) \sum_{j=1}^{i-1} \left(\begin{array}{c} i \\ j \end{array} \right) = O\left(3^k\right)$ potential rules. We use the following observation to prune rules: If X, Y are two itemsets, such that $X \subset Y$ and the confidence of $X \Rightarrow Y \setminus X$ is below the $MinConf$ threshold, then for any $X' \subset X$, the confidence of $X' \Rightarrow Y \setminus X'$ is also below $MinConf$. Similarly, for any $Y' \supset Y$, the confidence of $X \Rightarrow Y' \setminus X$ is below $MinConf$. This observation is correct because $Freq(X, DB) \leq Freq(X', DB)$ and $Freq(Y', DB) \leq Freq(Y, DB)$. If, on the other hand, the rule $X \Rightarrow Y \setminus X$ is confident, then for every $X \subset X' \subset Y' \subseteq Y$, the rule $X' \Rightarrow Y' \setminus X'$ is confident as well.

This observation allows us to alter Distributed Decision Confidence Miner by splitting it into several rounds. At each round of the improved algorithm (Alg. 5), many of the possible rules can either be pruned or inferred with no communication. We initialize the candidate rule set R_k with a single rule $R_0 = \{\emptyset \Rightarrow \emptyset\}$, which must be both supported and confident. In each round we run an algorithm similar to Distributed Decision Confidence Miner to decide which of the rules in R_k are confident. We develop some new candidate rules according to the following two candidate generation methods: If a rule r_k is found to be confident, then every rule which specifies the antecedent or generalizes the consequent of r_k must also be confident and every rule which further specifies the consequent is considered a candidate. If, on the other hand, a rule was found not to be confident, then any rule which specifies its antecedent is still a candidate.

5 Experimental Results

We used synthetic databases generated with the gen tool [13], which is based on ideas published in [2]. We generated several large databases and then sampled them to generate the partitioned database. The

Algorithm 5 Pruning Distributed Decision Confidence Miner

Definitions: For some $X \in L_k$, $specifiers(X) = \{X' \in L_{k+1} : X \subset X'\}$

For node i of n nodes:

1. Initialize: $R_0 = \{\emptyset \Rightarrow \emptyset\}$, $k = 0$, $R = \emptyset$
2. While $R_k \neq \emptyset$
   (a) Initialize $Passed = \emptyset$
   (b) Do
   - Choose r_l to be some $X \Rightarrow Y \setminus X \in R_k$ such that $i \neq G(r_l)$ and either $H(r_l) < MinFreq \leq P(r_l, DB^i)$ or $P(r_l, DB^i) < MinFreq \leq H(r_l)$
     - If both $Support(X, DB^i)$ and $Support(Y, DB^i)$ were not sent, broadcast $\langle k, Support(X, DB^i), Support(Y, DB^i)\rangle$.
     - Else if $Support(X, DB^i)$ was already sent, broadcast $\langle k, Support(X, DB^i)\rangle$.
     - Else if $Support(Y, DB^i)$ was already sent, broadcast $\langle k, Support(Y, DB^i)\rangle$.
   - If there is no such r_l, broadcast $\langle pass\rangle$.

   (c) Until $|Passed| = n$
   (d) For each $r_l = X \Rightarrow Y \setminus X \in R_k$ such that $H(r_l) < MinConf$, $R_{k+1} = R_{k+1} \cup \{X' \Rightarrow Y \setminus X' : X' \in large\_extensions(X_p)\}$
   (e) For each $r_l = X \Rightarrow Y \setminus X \in R_k$ such that $H(r_l) \geq MinConf$
   - $R_{k+1} = R_{k+1} \cup \{X \Rightarrow Y' \setminus X : Y' \in large\_extensions(Y)\}$
   - $R = R \cup \{X' \Rightarrow Y' \setminus X' : X \subseteq X' \subset Y' \subseteq Y\}$

   (f) $k = k + 1$

When node i receives a message M from node j:

1. If $M = \langle pass\rangle$ insert p to $Passed$.
2. Else $M = \langle k, Support(X, DB^j), Support(Y, DB^j)\rangle$, $\langle k, Support(X, DB^j)\rangle$, *or* $\langle k, Support(Y, DB^j)\rangle$
   - If $i \in Passed$ remove i from $Passed$.
   - Recalculate $G(r_l)$ for every r_l which includes X and/or Y. If $G(r_l)$ changes, update $H(r_l)$ and $P(r_l)$ as well.

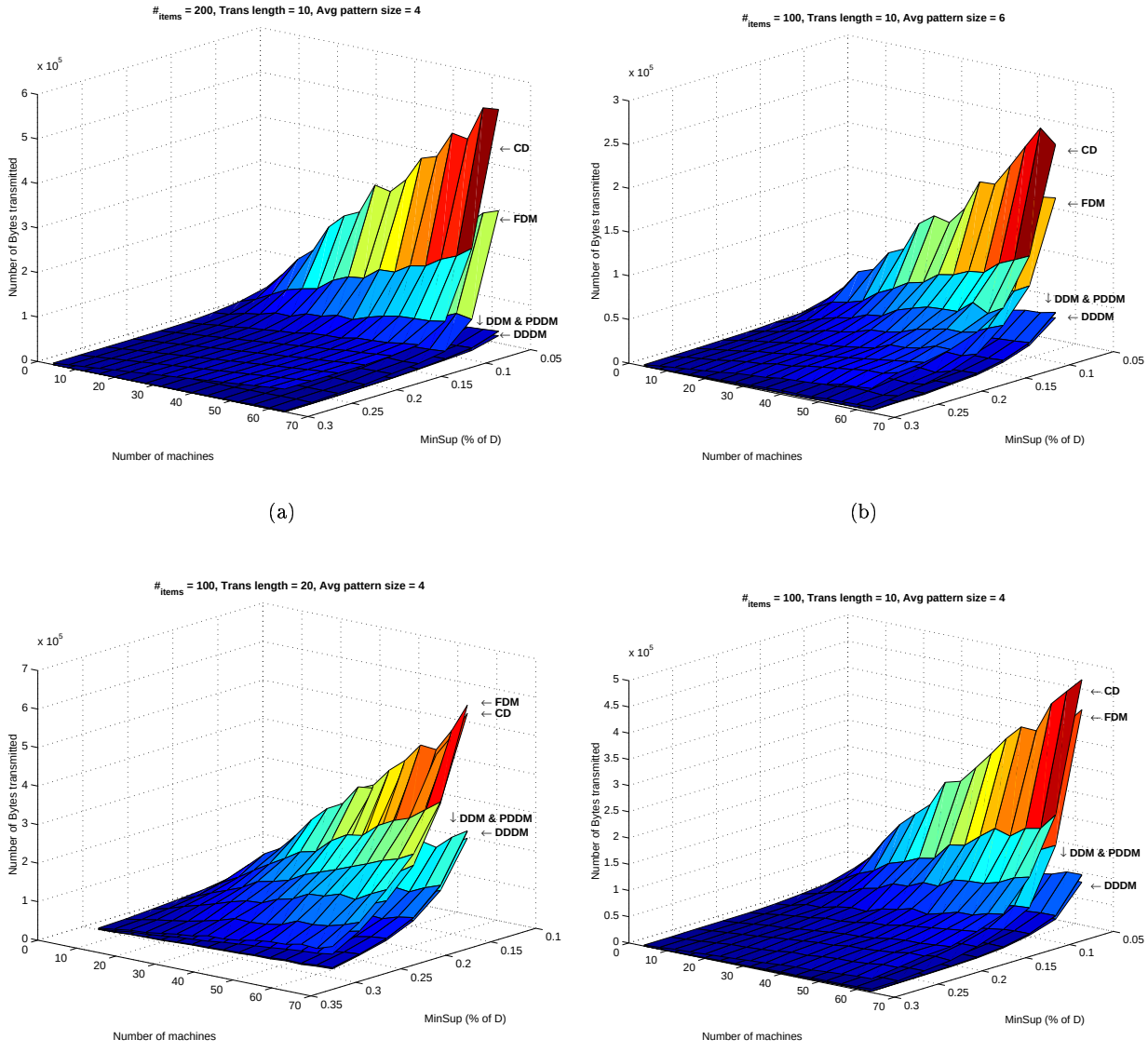
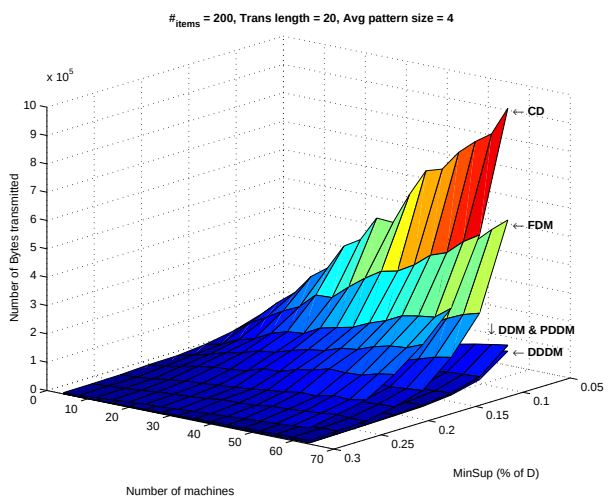
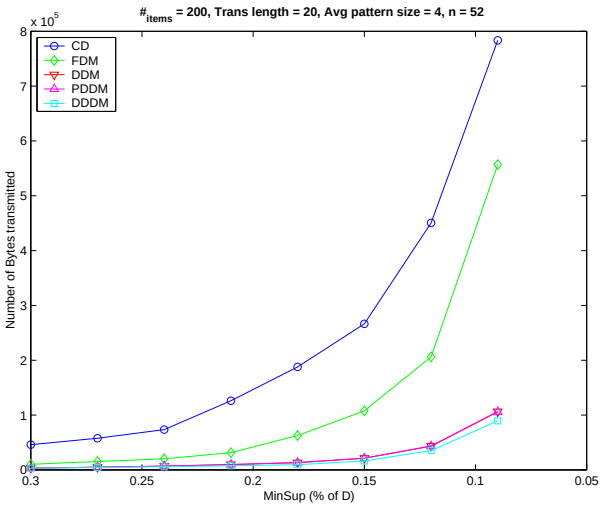


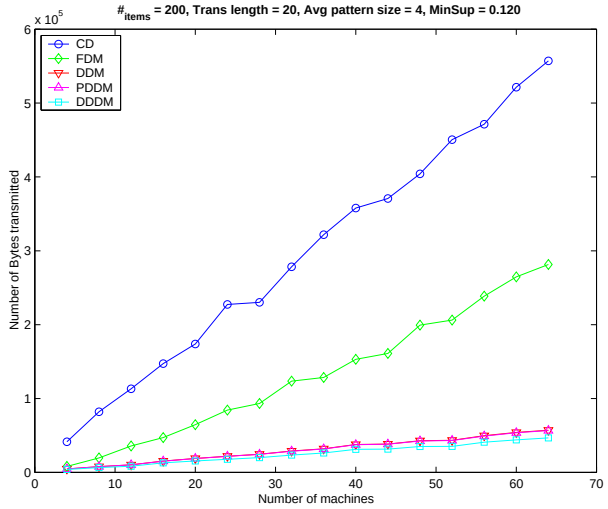
Figure 2: Figures 2(a), 2(b), 2(c) and 2(d) show the typical performance of CD, FDM, DDM, PDDM and DDDM over several unskewed databases, measured by the number of transmitted bytes vs. n and MinSup. The MinSup parameter is related to both the number of candidates and Pr_{above}. It can be seen that CD and FDM are not scalable with either n or MinSup. Note that when the partition is unskewed, DDM and PDDM are effectively the same algorithm because there are no obvious leaders.



(a)



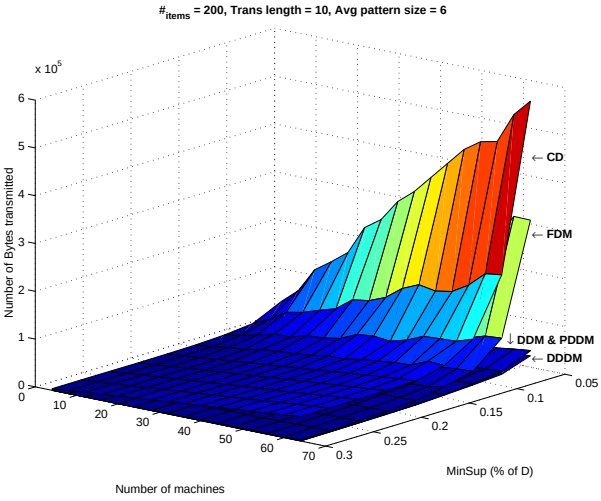
(b)



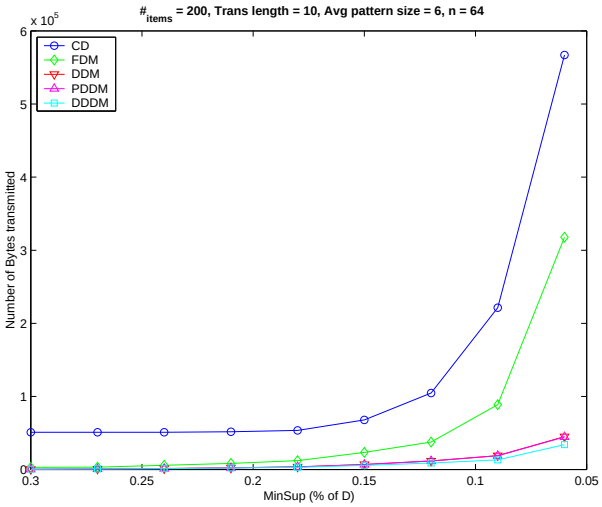
(c)

	16		52	
0.12	CD	147200	CD	450528
	FDM	47190	FDM	206160
	DDM	15360	DDM	43480
	$PDDM$	15244	$PDDM$	43184
	$DDDM$	12892	$DDDM$	35270
0.30	CD	14144	CD	45968
	FDM	2576	FDM	10296
	DDM	1070	DDM	3386
	$PDDM$	1056	$PDDM$	3292
	$DDDM$	1062	$DDDM$	3186

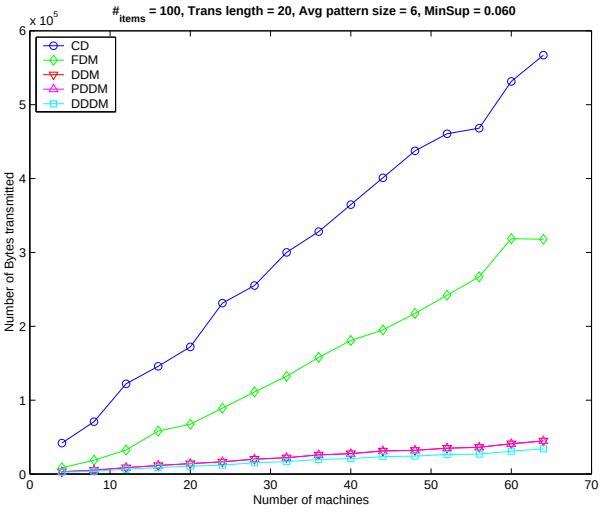
Figure 3: Figure 3(a) shows the performance of CD, FDM, DDM, PDDM and DDDM on an unskewed database like those of 2(a) through 2(d). Here we added views of the performance for fixed n (3(b)) and fixed MinSup (3(c)) and a sample of values for all algorithms for $n = 16$, 52 and $Minsup = 0.12 \cdot D$, $0.3 \cdot D$.



(a)



(b)



(c)

	8		56	
0.06	CD	70880	CD	468160
	FDM	18756	FDM	266916
	DDM	5242	DDM	36340
	$PDDM$	5286	$PDDM$	36058
	$DDDM$	3816	$DDDM$	26918
0.30	CD	6368	CD	44576
	FDM	218	FDM	1650
	DDM	46	DDM	312
	$PDDM$	42	$PDDM$	252
	$DDDM$	36	$DDDM$	270

Figure 4: Figure 4(a) shows the performance of CD, FDM, DDM, PDDM and DDDM on yet another unskewed database. The performance of DDM is far better than FDM, and DDDM outperforms DDM.

sample sizes were kept to less than 10% of the original database in order to reduce the dependency between different experiments. We systematically scanned MinSup values of 3% to 30% of the size of the database in order to sidestep the risks of parameter tuning. We used the reasonable $MinConf = 0.5$. Communication load was measured assuming 4 bytes for support count encoding and 2 bytes for itemset number encoding.

Figures 2, 3 and 4 show the typical communication loads of CD, FDM, Distributed Decision Miner, Preemptive Distributed Decision Miner and Distributed Dual Decision Miner on various databases. It can be seen that the latter three algorithms use far less communication than the former two. It can also be seen that both CD and FDM are nonscalable with respect to either n or MinSup. For unskewed databases, PDDM and DDM behavior is effectively the same, and DDDM is the best of the three.

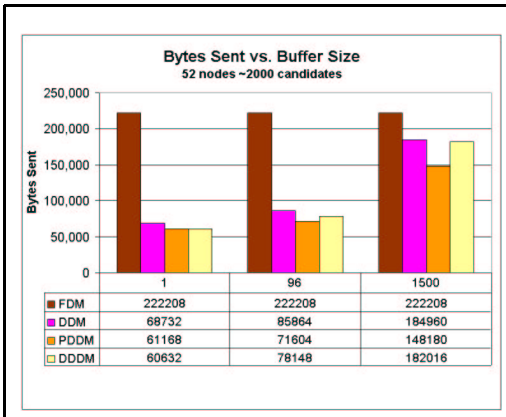
Next we investigate the dependency of our algorithms on message size. We assume implicit buffering of messages, i.e., each time the algorithm 'sends' an itemset index or local support count, that information is stored in a buffer. The buffer is sent only when it fills up, or when the algorithm explicitly flushes it (as would happen in PDDM when the leader has no more data to send). This change has no effect on algorithm correctness but it may affect performance in two ways: Larger buffers mean smaller message overhead but they also increase the probability that several nodes will simultaneously (i.e., in concurrent buffers) send information about the same candidate. This probability decreases, however, as the number of candidates increases.

We first checked how the number of bytes sent and the number of messages sent respond to changes in the buffer size (figure 5). We checked two buffer sizes: buffers of 96 bytes, which are typical for fast Ethernet networks such as Myrinet and ServerNet, and buffers of 1500 bytes, which are typical for Ethernet (IEEE-802.3). To show that these results are not much different from those obtained in an ideal network, we included results for a network that has no buffering at all.

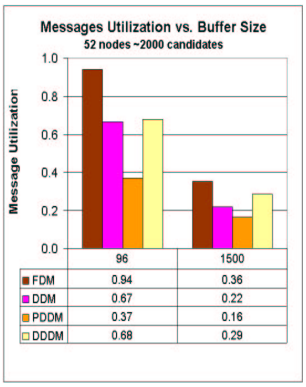
The results show that our algorithms are always better with respect to the number of bytes sent. The number of messages sent is strongly related to another metric – the message utilization. Message utilization is the average fraction of the message buffer which is actually used. For large buffers and a small number of candidates, message utilization is low and the number of messages is high with respect to FDM. For small buffers and a high number of candidates, the message utilization is high and our algorithms are far better than FDM. PDDM uses consistently more messages than DDM and DDDM because in these balanced datasets the leader passes on its turn many times. Every such pass necessitates sending a message whether it is full or empty. Hence, these passes lead to much lower message utilization.

A further set of experiments (figure 6) checked if the behavior displayed in figure 5 remains the same when the number of nodes increases. In other words, the question is whether our algorithms remain scalable when messages are buffered. The results show that message utilization remains very stable regardless of the number of nodes. This means that the superior scalability of our algorithms holds not only with respect to the number of bytes, but with respect to the number of messages sent as well. Hence, as the number of nodes increases, so does the message efficiency of our algorithms, as compared to FDM. For the reasons mentioned above, PDDM uses more messages than FDM when the buffers are large. Although PDDM shows the same trend as DDM and DDDM, it is still less message-efficient even for 64 nodes.

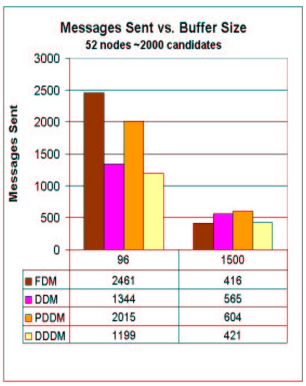
The final set of experiments (figure 7) is concerned with heavily skewed datasets. For this set of experiments we created 65 database partitions: 64 partitions had the same number of transactions and the last one had the same number of transactions as the former 64 combined. This emulates real-life scenarios in which there is one dominating database (a server, a central department, etc.) and many lesser ones (clients, regional departments, etc.). As expected, PDDM gives significant improvement over DDM in these scenarios. Furthermore, it should be noted that the improvements made by PDDM and DDDM are completely orthogonal. This is because PDDM reduces the communication needed for the identification of the frequent itemsets, while DDDM only deviates from DDM after the frequent itemsets have been identified. Hence a combined algorithm (which is not presented here) will necessarily be better than both DDDM and PDDM.



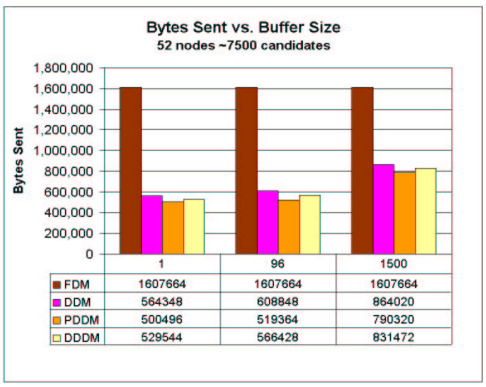
(a) The number of bytes increases with buffer size because nodes refer to the same itemsets in concurrent buffers.



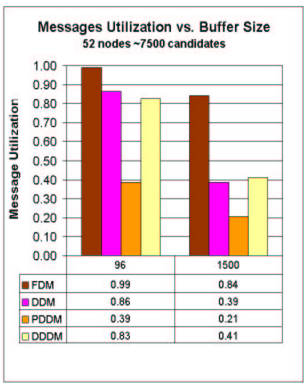
(b) Message utilization is the average fraction of the buffer that is actually used.



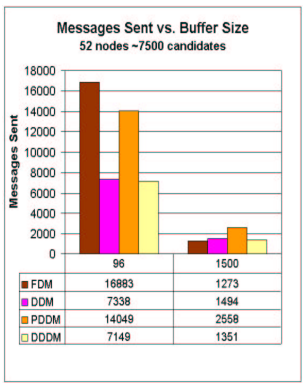
(c) As a result of the lower utilization, the number of messages inceases over that of FDM.



(d) When the number of candidates increases, the effects of large buffers subside.



(e) When the buffers are small, the messages are nearly full; when the buffers are large (top), they are half empty.



(f) Our algorithms give even greater improvement for small buffers and become equivalent for large buffers.

Figure 5: The effect of buffer size on the number of bytes and messages sent. If the number of candidates is large with respect to the buffer size (bottom row), our algorithms send far fewer bytes and messages than FDM. If the number of candidates is small (top row), messages are sent half-empty and FDM becomes competitive. PDDM uses many more messages because the leader sends a message each time it passes on its turn, regardless of whether that message is full or empty.

(a) Both the numbers of bytes sent and messages sent increase linearly with the number of nodes. However, the steepest increase is for FDM.

(b) Message utilization does not depend on the number of nodes

(c) For larger buffers, PDDM sends many more messages than all other algorithms. Yet DDM and DDDM are still better than FDM.

(d) Again, Message utilization is independent of the number of nodes. Not surprisingly, it is also lower.

Figure 6: Message utilization (5 and 5) remains stable regardless of the number of nodes. Hence, since our algorithms are scalable with respect to the number of bytes sent, they are also scalable with respect to the number of messages sent. However, this scalability does not suffice in the case of PDDM and large buffers because the initial number of messages is so much larger than that required by FDM that it remains the largest even for 64 nodes. DDM and DDDM, on the other hand, are better in both respects than FDM.

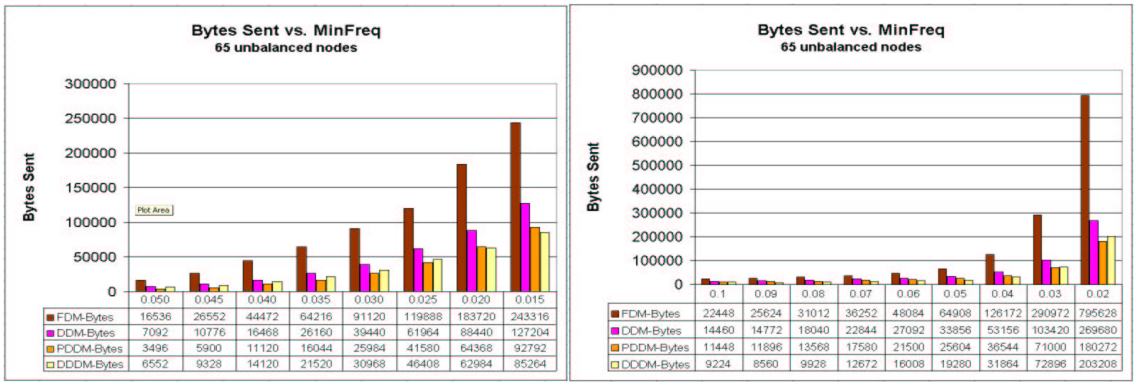


Figure 7: The number of bytes sent vs. MinFreq value for two different skewed databases. Both databases have 64 partitions which contain the same number of transactions and a 65^{th} partition which has the same number of transaction as the former 64 combined. Both PDDM and DDDM give significantly better results than DDM. Since those two algorithms improve different aspects of DDM, those improvements can be combined.

6 Conclusions

During the past few years, distributed systems have become a mainstream computing paradigm. Whether it be a company's Virtual Private Network, a multi-server billing center, a network of independent stockbrokers or a peer-to-peer mp3 library like Napster, the wealth of information available on-line is constantly expanding. This information is by and large distributed, and there is a growing need for tools that will assist in understanding and describing it.

These new databases differ from distributed databases of the past. The partitioning of the data is usually skewed. The connections between partitions are sparse and often unreliable, and various throughputs and latencies may apply. Distributed knowledge discovery algorithms will, in our view, become a major tool in the making, maintaining and analysis of distributed systems. This will require us to change our approach to distributed knowledge discovery and accept the skewed, sparsely connected, sometimes unreliable environment of these distributed systems. We will have to restate well-known problems and define new ones.

This paper can be viewed as an example of a new approach to one such well-known problem. The D-ARM problem was restated here as a decision problem, negotiated among different parties. The resulting algorithms are both more efficient, more resilient to data skewness, and better able to overcome certain communication difficulties such as unordered messages, variable or uneven throughputs, and the like. We intend to further extend the concept of mining through distributed decision-making and apply it to other areas of knowledge discovery.

Several open research questions remain. The hypothesis functions H and P play a central role in all our algorithms. Their optimality is thus an important open question. Given a certain target function, we would like to find those H and P which will result in the shortest discussion, i.e., those functions that yield the lowest communication costs. Our algorithms also assume that the network supports broadcast. It would be interesting to review the problem in networks that do not support broadcast. Finally, the DDDM algorithm is unique in that it reduces communication by not collecting all counts for the large itemsets. It would be interesting to see how this affects performance for real applications.

References

[1] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962 – 969, 1996.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l. Conference on Very Large Databases (VLDB'94)*, pages 487 – 499, Santiago, Chile, September 1994.

[3] R. Bayardo and R. Agrawal. Mining the most interesting rules. In *Proc. of the ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 145 – 154, San Diego, California, 1999.

[4] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD Record*, 6(2):255–264, June 1997.

[5] D. Cheung and Y. Xiao. Effect of data skewness in parallel mining of association rules. In *12th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 48 – 60, April 1998.

[6] D.W. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. of 1996 Int'l. Conf. on Parallel and Distributed Information Systems*, pages 31 – 44, Miami Beach, Florida, December 1996.

[7] Eui-Hong (Sam) Han, George Karypis, and Vipin Kumar. Scalable parallel data mining for association rules. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):352 – 377, 2000.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. Technical Report 99-12, Simon Fraser University, October 1999.

[9] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *Proc. of the 21st Int'l. Conference on Very Large Data Bases (VLDB'95)*, pages 420 – 431, Zurich, Switzerland, September 1995.

[10] Zoltan Jarai, Aashu Virmani, and Liviu Iftode. Towards a cost-effective parallel data mining approach. Workshop on High Performance Data Mining (held in conjunction with IPPS'98), March 1998.

[11] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. of ACM SIGMOD Int'l. Conference on Management of Data*, pages 175 – 186, San Jose, California, May 1995.

[12] A. Schuster and R. Wolff. Communication-efficient distributed mining of association rules. In *Proc. of the 2001 ACM SIGMOD Int'l. Conference on Management of Data*, pages 473 – 484, Santa Barbara, California, May 2001.

[13] R. Srikant. Synthetic data generation code for association and sequential patterns. Available from the I.B.M. Quest Web site at http://www.almaden.ibm.com/cs/quest/.

[14] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. of the 20th Int'l. Conference on Very Large Databases (VLDB'94)*, pages 407 – 419, Santiago, Chile, September 1994.

[15] Hannu Toivonen. Sampling large databases for association rules. In *The VLDB Journal*, pages 134–145, 1996.

[16] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. In *Proc. of Supercomputing'96*, pages 17–22, Pittsburg, PA, November 1996.

Authors' Biography:

Assaf Schuster (http://www.cs.technion.ac.il/~assaf) is an Associate Professor with the Computer Science Department at the Technion - Israel Institute of Technology. He received his B.A., M.A., and Ph.D. degrees in Mathematics and Computer Science from the Hebrew University of Jerusalem, the latter one in 1991. His research interests and fields of publication include memory hierarchies and consistency models,

Java memory model, distributed shared memory, parallel and distributed computing, scalable symbolic model checking, global memory systems, database replication and scalability, and distributed data mining. He is the head of the Distributed Systems Laboratory at the Technion (http://dsl.cs.technion.ac.il), serves as an associate editor of the Journal of Parallel and Distributed Computing, and is leading several large projects in his area.

Ran Wolff (http://www.cs.technion.ac.il/~assaf) is a Ph.D. student with the Computer Science Department at the Technion - Israel Institute of Technology, where he received his B.A. in Computer Sience as well. His research deals with distributed data mining algorithms and data mining of streams.