# Sampling-Based Distributed Mining of Association Rules

Research Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Electrical Engineering

Dan Trock

Submitted to the senate of Technion - Israel Institute of Technology

HESHVAN, 5764 HAIFA NOVEMBER 2003

The research thesis was done under the supervision of Associate Professor Yoram Moses in the Faculty of Electrical Engineering.

THE GENEROUS FINANCIAL HELP OF TECHNION IS GRATEFULLY ACKNOWLEDGED.

I would like to express my gratitude to Prof. Yoram Moses and Prof. Assaf Schuster for their support during my studies. I thank Ran Wolff very much for his help, ideas and enthusiasm. Thanks to Prof. Sivan Toledo and the system group of the Tel-Aviv University for their cooperation. Thanks to Eran and Stas for the excellent technical support at the DSL lab. My special thanks to my family and Mila for loving and inspiring me throughout this work.

# Contents


**1 Introduction and Related Work** **3**

1.1 Introduction . . . 3

1.2 Problem Definition . . . 4

1.3 Characteristics of ARM Algorithms . . . 6

1.4 Distributed ARM Algorithms . . . 8

1.4.1 Apriori-based D-ARM Algorithms . . . 9

1.4.2 FP-Growth based Distributed Algorithm . . . 11

1.5 Sequential Sampling . . . 12

**2 D-Sampling** **13**

2.1 Algorithm . . . 15

2.2 MDDM – A Modified Distributed Decision Miner . . . 16

2.3 M-Max Algorithm . . . 16

**3 Statistical Analysis** **19**

3.1 A Bound on the Sampling Error . . . 19

3.2 Uniformly Sampling a Partitioned Database . . . 21

**4 Experiments** **22**

4.1 Speedup Results . . . 23

4.2 Dependency on $MinFreq$ . . . 23

4.3 Scale-up . . . 25

4.4 Number of Candidates . . . 26

**5 Discussion** **28**

**6 Conclusions** **30**

**A A High-Performance Distributed Algorithm for Mining Association Rules** **34**

**Abstract**

The economic value of data mining is today well established. Most large organizations regularly practice data mining techniques. One of the most popular techniques is association rule mining (ARM), which is the automatic discovery of pairs of element sets that tend to appear together in a common context. An example would be to discover that the purchase of certain items (say tomatoes and lettuce) in a supermarket transaction usually implies that another set of items (salad dressing) is also bought in that same transaction. Application domains for ARM include recommendation service, customer segmentation, catalog design, store layout and more.

ARM is a computationally and I/O intensive task. Given $m$ items there are $2^m$ subsets that might frequently occur together. An exhaustive search over this exponential space is infeasible, except for very small values of $m$. The number of records is also huge. Typical departmental stores stock thousands of items and collect millions of customer transactions every day. Processing all this data requires a lot of disk I/O. Given that data is increasing both in terms of the number of items (*dimensions*) and the number of transactions (*size*), one of the desirable characteristics of an ARM algorithm is *scalability*, i.e., the ability to handle massive data-stores. Since it is clear that sequential algorithms cannot provide the scalability in terms of the data dimensions, size, and the running performance for such large databases, it is a challenge of the high-performance parallel and distributed computing to fill this role.

Since the introduction of the problem, efficient sequential algorithms were presented for its solution. A variety of Distributed ARM (D-ARM) algorithms were also shown in the context of parallelization and IO-speedup. Unfortunately, all the distributed algorithms presented scan the database multiple times which is still the main show-stopper for this task. In this work, we present a new algorithm that is the first D-ARM algorithm to perform a single scan over the database. As such, its performance is unmatched by any previous algorithm. The algorithm is based on the sequential Sampling algorithm, and demonstrates superlinear speedup with the number of computing nodes. Scale-up experiments over standard benchmarks demonstrate stable run time regardless of the number of computers. Theoretical analysis reveals a tighter bound on error probability than the one shown in the corresponding sequential algorithm.

# List of Abbreviations

| | |
|---|---|
| $ARM$ | Association Rules Mining |
| $D-ARM$ | Distributed Association Rules Mining |
| $DB$ | transactional database to be mined |
| $D$ | number of transactions in $DB$ |
| $DB^i$ | relative portion of $DB$ at computing node $i$ |
| $D^i$ | number of transactions in $DB^i$ |
| $S$ | sample transactional database taken from $DB$ |
| $\|S\|$ | number of transactions in $S$ |
| $S^i$ | relative portion of $S$ at computing node $i$ |
| $Freq(X, DB^i)$ | local frequency of itemset $X$ in $DB^i$ |
| $Freq(X, DB)$ | global frequency of itemset $X$ in $DB$ |
| $MinFreq$ | minimum frequency of any itemset in $DB$ in order to consider associations with it |
| $MinConf$ | minimum confidence of any association between itemsets in order to be reported to the user |
| $F_{fr}(A)$ | group of all itemsets with frequency above or equal to $fr$ in $A$ |
| $NB$ | negative border of a group of itemsets |

# 1 Introduction and Related Work

## 1.1 Introduction

The availability of cheap storage and the progress in automated data gathering have led, during the last decade, to the development of data mining (a.k.a. knowledge discovery) algorithms and tools. The goal of these tools is to generate knowledge in the form of generalizations, summaries, rules of behavior etc. about a database. Such knowledge has many applications in the context of understanding, describing and acting upon the database.

Association Rules Mining (ARM) in large transactional databases is one of the central problems in the field of knowledge discovery. The input to the ARM problem 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 each other. 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 indeed $X$ usually implies $Y$ we say that the rule $X \Rightarrow Y$ is confident in the database.

Like other data mining techniques that must process enormous databases, ARM is inherently disk-I/O intensive. These I/O costs can be reduced in two ways: by reducing the number of times the database needs to be scanned, or through parallelization, by partitioning the database between several machines which then perform a distributed ARM (D-ARM) algorithm. In recent years much progress has been made in both directions.

The main task of every ARM algorithm is to discover the sets of items that frequently appear together – the frequent itemsets. The number of database scans required for the task has been reduced from a number equal to the size of the largest itemset in Apriori [2], to typically just a single scan in modern ARM algorithms such as Sampling and DIC [23, 4].

Much progress has also been made in parallelized algorithms. With these, the architecture of the parallel system plays a key role. For instance, many algorithms were proposed which take advantage of the fast interconnect, or shared memory of parallel computers. The latest development with these is [24], in which each processor makes just two passes over its portion of the database.

Parallel computers are, however, very costly. Hence, although these algorithms were shown to scale up to 128 processors, few organizations can afford to spend such resources on data mining. The alternative is distributed algorithms, which can be run on cheap clusters of standard, off-the-shelf PCs. Algorithms suitable for such systems include the CD and FDM algorithms [1, 6], both parallelized versions of Apriori,

which were published shortly after it was described. However, while clusters may easily and cheaply be scaled to hundreds of machines, these algorithms were shown not to scale well [5]. The DDM algorithm [16], which overcomes this scalability problem, was recently described. Unfortunately, all the D-ARM algorithms for share-nothing machines scan the database as many times as Apriori. Since many business databases contain large frequent itemsets (long patterns), these algorithms are not competitive with DIC and Sampling.

In this work we present a parallelized version of the Sampling algorithm, called D-Sampling. The algorithm is intended for clusters of share-nothing machines. The main obstacle of this parallelization, that of achieving a coherent view of the distributed sample at reasonable communication costs, was overcome using ideas taken from DDM. Our distributed algorithm scans the database once, just like Sampling algorithm, and is thus more efficient than any D-ARM algorithm known today. Not only does this algorithm divide the disk-I/O costs of the single scan by partitioning the database among several machines, but also uses the combined memory to linearly increase the size of the sample. This increase further improves the performance of the algorithm because the safety margin required in Sampling decreases when the (global) sample size increases.

Extensive experiments on standard synthetic benchmarks show that D-Sampling is superior to previous algorithms in every way. When compared to Sampling – one of the best sequential algorithm known today – it offers superlinear speedup. When compared to FDM and DDM, it improves runtime by orders of magnitude. Finally, on scalability tests, an increase in both the number of computing nodes and the size of the database does not degrade D-Sampling performance.

## 1.2 Problem Definition

The ARM problem has many variations differed by these four main features:

1. The data:

   The input to the ARM problem is a database of transactions. Each transaction may be composed of the set of items purchased by a client [9][2] or also contain the quantities bought [19]. A taxonomy, or partial order, may be supplied along with the items as in [10][18] or they can be simply enumerated as in all previous examples.

2. Rule importance criteria:

   All ARM algorithms report only those rules which are important in some sense. Many criteria were offered by which the importance of a rule can be judged. In [3] Bayardo and Agrawal give an

overview of such criteria and show that a simple support (frequency) and confidence partial order $\leq_{sc}$ on rules encompasses many of them.

3. Rule constraints:

   Many times the user of the algorithm is not interested in a rule unless it fulfills some constraints. For example there can be item constraints [20] or meta rules [7]. In [14] Pei and Han give a good categorization of such constraints.

4. The required mining process:

   The process of mining can have its own variations. It can be a one time action (as are all prior examples), an ongoing update process (see the FUP algorithm [21] and [22]), or even a cumulative process in the spirit of anytime algorithms, where more and more rules are discovered over time.

It is widely accepted that, at least for academic purposes, the basic ARM problem is the one in which: A. there are no taxonomies or quantitative transactions, B. rules are generated which surpass frequency and confidence thresholds, C. no further rule constraints are applied and D. the process of mining is a one time action over static data. Hence the problem definition is the following:

> Let $I$ be a set of items. An itemset is some $A \subseteq I$ and a $k$-sized itemset is an itemset which includes exactly $k$ items. A transaction is some $T \subseteq I$ and a database $DB$ of size $D$ is a list of $D$ such transactions each one with its own unique transaction identifier ($tid$). For every itemset $A$ the support count of $A$ in $DB$, $Support(A, DB)$, is the number of transactions in $DB$ to which $A$ is a subset, and its frequency is $Freq(A, DB) = \frac{Support(A,DB)}{D}$.
>
> For a given frequency threshold $MinFreq$ we say that an itemset $A$ is frequent if and only if $Freq(A, DB) \geq MinFreq$. For a pair of frequent itemsets $X, Y$ such that $X \subset Y$ and a given $MinConf$ threshold we say that the association (rule) $X \Rightarrow Y \setminus X$ is confident if $Freq(Y, DB) \geq MinConf \cdot Freq(X, DB)$.
>
> Given such database and thresholds the problem of Association Rule Mining is to find the list of all the confident associations (rules) between frequent itemsets in the database.

All known ARM algorithms work by identifying all frequent itemsets. Once the frequent itemsets and their frequencies are known, significant rules can be generated in a straightforward procedure by comparing the frequency of a frequent itemset to the frequencies of all its generalizations.

## 1.3 Characteristics of ARM Algorithms

Several choices have to be made when a new ARM algorithm is developed. These choices have to do with fundamental characteristics of the algorithm and will determine, to a large extent, its points of strength and weakness. Following is a list of those characteristics:

**Direction of search for frequent itemset:** The group of itemsets can be viewed as a lattice (see figure 1) with the empty itemset at the bottom and the full itemset at the top. ARM algorithms search this itemset lattice for frequent itemsets. One of the main observations in ARM is that if an itemset $A$ is frequent then any itemset $B$ which generalizes it ($B \subset A$) must also be frequent. From this observation we can deduce two pruning rules. *Bottom-up*: given that a certain itemset is infrequent the algorithm can refrain from counting all its specifications, for they too are certainly infrequent. *Top-down*: given that a certain itemset is frequent the algorithm can refrain from counting all its generalizations for they too are certainly frequent. It is impractical to search the lattice in a top-down manner because the frequent itemsets are much more sparsely spread on the lattice at its top than at the bottom. All known algorithms use either bottom-up search (see Apriori [2], FP-tree [9]) or a hybrid bottom-up and top-down search (see Pincer-Search [12]).

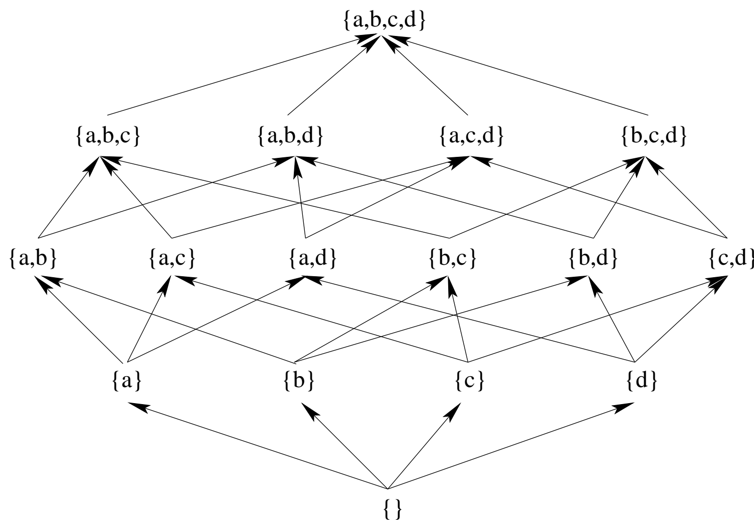
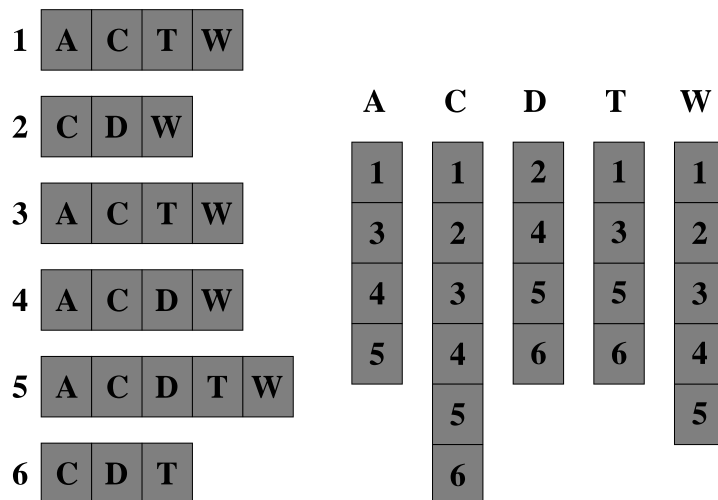



Figure 1: The itemsets form a lattice with the empty itemset at the bottom and the full itemset at the top.

**Data Layout** The database can be organized in one of two ways (see figure 2): either **horizontal** as a list of tid's, each followed with the items which are included in that transaction, or **vertical** as a list of items, each followed by the tid's of the transactions in which that item appear. The latter view can, once the algorithm advances, be extended to a list of itemsets (initially 1-sized itemsets) followed again by the tid's of the transactions in which the prescribed itemset is contained. The main problem of vertical representation is that for usual distributions of items on transactions it is much larger, at least for 1-sized and 2-sized itemsets. A hybrid approach is also possible (see Eclat algorithm at [27]) where horizontal layout is used for mining the 1-sized and 2-sized itemsets and then vertical layout is used for mining larger sized itemsets.

Figure 2: Horizontal vs. Vertical Data Layout




(a) Horizontal Layout (b) Vertical Layout

**Deterministic vs. Heuristic Pruning:** As we already said, not all itemsets have to be considered by an ARM algorithm. Specifically, information gathered on some itemsets can be used to prune the search tree and refrain from counting many of the itemsets. Even if we restrict the discussion to algorithms which compute the complete set of frequent itemsets (without allowing approximation) there still remains the question of the order by which the itemsets are considered. For example, both Apriori [2] and FP-tree [9] consider itemsets in a breadth first search from the bottom-up. They do not start to count frequency of an itemset before it is known that all its generalizations are frequent. On the other hand,

algorithms such as Pincer-Search, Partition, AS-CPA, DIC and Sampling ([12][15][13][4][23] respectively) consider itemsets before it is certain that their generalizations are frequent. The drawback of the latter algorithms is the larger number of false positives, itemsets which are counted and prove to be infrequent. The advantage, on the other hand, is a reduction in the number of database scans.

**Parallel System Architecture:** The main distinction between parallel ARM algorithms is the architectures they are intended for. Typically these would include Shared Memory Machines, Distributed Shared Memory Machines or Shared Nothing Machines. Several of the methodologies used for parallel ARM are restricted to a certain architecture. Especially, any algorithm requiring comprehensive sharing of information will probably only be practical for Shared Memory Machines. Since our main interest is in share nothing machines we will focus on algorithms which are applicable to such systems.

**Data vs. Task Parallelism:** Considering parallel algorithms one of the main decisions is what to partition between the processors: the database or the candidate itemsets. Partitioning the database offers linear disk I/O speedup. Partitioning the candidates, on the other hand, offers better memory utilization but may require, in the case of Share Nothing Machines, replication of the database to all the processors (See Eclat parallelization [25]). Hybrid approaches have also been offered which partition both candidates and database (See Par-Eclat [28]).

## 1.4 Distributed ARM Algorithms

We are interested in the kind of Distributed ARM (D-ARM) problems which have the following two characteristics. First, the number of nodes participating $n$ is very large due to large sizes of todays databases. Second, at the beginning of the problem the databases may already be horizontally partitioned among those nodes. We would like not to limit in any way the other important characteristics of the problem: the local sizes of the database $D^i$, the number of items $I$ and the size of the itemsets.

The first characteristic dictates that our algorithms must use data parallelization, because any algorithm which requires moving data from one node to another will not scale with the size of the database or the number of nodes. The second characteristic dictates that the data layout must be horizontal.

For such distributed algorithms we give the following additional notation:

> We assume the algorithm is executed by $n$ computing nodes (parties) which communicate by exchanging messages. Let $DB$ be a transactional database of size $D$. Let $\overline{DB} = \{DB^1, DB^2, ..., DB^n\}$ be a horizontal partition of $DB$ into $n$ partitions of sizes $\overline{D} = \{D^1, D^2, ..., D^n\}$

respectively. Let $S$ be a list of transactions which were sampled uniformly from $DB$, and let $\overline{S} = \left\{S^1, S^2, ..., S^n\right\}$ be the partition of $S$ induced by $\overline{DB}$. For any itemset $X$ we call $Freq\left(X, DB^i\right)$ the *local frequency* of $X$ in partition $i$ and $Freq\left(X, DB\right)$ its *global frequency*; likewise, we call $Freq\left(X, S^i\right)$ the *estimated local frequency* of $X$ in partition $i$ and $Freq\left(X, S\right)$ its *estimated global frequency*. For some frequency threshold $0 \leq MinFreq \leq 1$, we say that an itemset $X$ is *frequent* in $A$ if $Freq\left(X, A\right) \geq MinFreq$ and infrequent otherwise. If $A$ is a sample, we say that $X$ is *estimated frequent* or *estimated infrequent*. If $A$ is a partition, we say that $X$ is *locally frequent*, and if $A$ is the whole database, then $X$ is *globally frequent*. Hence an itemset may be estimated locally frequent in the $k^{th}$ partition, globally infrequent, etc. The group of all itemsets with frequency above or equal to $fr$ in $A$ is called $\mathcal{F}_{fr}\left[A\right]$. The *negative border* of $\mathcal{F}_{fr}\left[A\right]$ is all those itemsets which are not themselves in $\mathcal{F}_{fr}\left[A\right]$ but have all their subsets in $\mathcal{F}_{fr}\left[A\right]$. Finally, for a pair of globally frequent itemsets $X$ and $Y$ such that $X \cap Y = \emptyset$, and some confidence threshold $0 < MinConf \leq 1$, we say the rule $X \Rightarrow Y$ is *confident* if and only if $Freq\left(X \cup Y, DB\right) \geq MinConf \cdot Freq\left(X, DB\right)$.

**D-ARM problem restatement**: Given a partitioned database $\overline{DB}$, and given $MinFreq$ and $MinConf$, the D-ARM problem is to find all the confident rules between itemsets in $\mathcal{F}_{MinFreq}\left[\overline{DB}\right]$.

Reviewing previously published distributed algorithms we can see that only a small group of shared-nothing Apriori [2] based D-ARM algorithms, and one more FP-Growth [9] based algorithm conform with this paradigm. The Apriori based include CD [1], FDM [6], FPM [5] and DDM [16] algorithms. The FP-Growth based is described in **[11]**.[1] Below, we overview these algorithms. For a good survey of parallel algorithms which includes algorithms fit only for Shared Memory or Tightly coupled Shared Nothing machines see [26].

### 1.4.1 Apriori-based D-ARM Algorithms

**Apriori** is a sequential algorithm which scans the itemset lattice in a DFS bottom-up manner. Starting from level 1 where each itemset contains exactly one item Apriori scans the database and computes the frequency of every itemset. Apriori considers, in the same manner, every $k+1$ level itemset if and only if all its $k$ level subsets proved to be frequent. The algorithm terminates when on some $k+1$ level no candidates should be considered. Hence, it performs a number of database scans

[1] This article was recently published, when most of the work described here was already carried out.

exactly equal to the maximal number of items in a frequent itemset.

**CD** is an obvious parallelization of Apriori in which the database is partitioned among $n$ parties. Each of these parties concurrently scans its partition and produces local frequencies for every candidate. Then a global sum reduction is performed in which every party receives the global frequency for every candidate. Given the list of $k$ level frequent itemsets, the list of $k+1$ level candidates can be independently calculated by all the parties with no communication or synchronization and the algorithm proceeds similarly to Apriori. The algorithm makes the same number of database scans as Apriori, but since the database is partitioned among the parties the I/O is expedited linearly with $n$. The algorithm carries $O(n \cdot |C|)$ communication complexity penalty where $C$ is the set of all itemsets considered by Apriori.

**FDM** introduces a new pruning technique called local pruning. Using the observation that in order for an itemset to be globally frequent it has to be locally frequent in at least on partition of the database. FDM replaces the sum reduction stage of CD with a three step procedure. First every party reports to a preassigned polling station which itemsets are relatively frequent (locally frequent) in its partition. Then the polling station broadcast the list of all itemsets which are locally frequent at some partition. Finally the global sum is computed only for those itemsets. As a result from this pruning technique many of the candidates may be ignored altogether and communication costs reduces. FDM has two main drawbacks. The first drawback is that it uses two synchronization points instead of the one used in CD. The second drawback is that the communication complexity of FDM is $O(Pr_{potential} \cdot n \cdot |C|)$ where $Pr_{potential}$ is the probability that an itemset is locally frequent on some partition. Unfortunately $Pr_{potential}$ grows very fast to 1 with the increase of $n$. This is because when the database is not homogeneous and the number of partitions is large there is a good chance that every candidate itemset might be locally frequent in at least one partition. If this happens then FDM communication costs deteriorates to be worse than those of CD.

**FPM** introduces two other pruning techniques the stronger of which is called global pruning. In global pruning each party independently computes a bound on the maximal local frequency of every other party for a $k+1$-sized itemset, those candidates for which the sum of the bounds is lower than $MinFreq$ can then be pruned. The bound each party computes is the minimum local frequency that other party reported for any $k$-sized itemset which is a generalization of the $k+1$-sized itemset considered. The efficiency of this pruning technique increases as the database becomes less homogeneous. However, for homogeneous databases the communication complexity of FPM is no

better than CD.

**DDM** In 2001, Schuster and Wolff presented another Apriori-based D-ARM algorithm called DDM [16]. As in Apriori, candidates in DDM are generated level-wise and are then counted by each node in its local database. The nodes then perform a distributed decision protocol in order to find out which of the candidates are frequent and which are not. DDM differs from FDM in that the DDM protocol allows that some of the nodes choose to publish the local frequency of a candidate while others choose not to. The protocol is directed by two hypotheses which are maintained about each candidate: in one, called the public hypothesis, each node assumes that the global frequency of the itemset is equal to the average of the local frequencies published for it thus far (or zero if none was published); in the other, called the private hypothesis, each node assumes that its local frequency is shared by all those which have not published their own local frequency for the candidate. If a node finds that the public and private hypotheses about an itemset disagree (i.e., one predicts that the itemset is frequent while the other predicts that it is infrequent), it will publish the local frequency. It is easy to show that when the protocol dictates that no node should publish the local frequency of a certain itemset, the public hypothesis for that itemset correctly predicts whether it is frequent or infrequent. DDM improves the communication complexity of previous solutions to $O\left(Pr_{above}\left|C\right|n\right)$, where $Pr_{above}$ is the chance of an itemset being locally frequent at a specific partition. $Pr_{above}$ is by definition smaller than $Pr_{potential}$ and is also independent of $n$. DDM is thus far more communication efficient, scalable, and resilient to data skewness.

### 1.4.2 FP-Growth based Distributed Algorithm

The FP-Growth algorithm can be divided into two phases: the construction of FP-Tree, and mining frequent patterns from the FP-Tree [9]. The construction of FP-Tree requires two scans of the database. The first scan accumulates the support of each item and then selects items that satisfy minimum support, i.e. frequent 1-itemsets. Those items are stored in frequency descending order to form F-list. The second scan constructs FP-Tree as follows: first, the transactions are reordered according to F-list, while non-frequent items are stripped off. Then, reordered transactions are inserted into FP-Tree. The order of items is important since in FP-Tree itemsets with same prefix share same nodes. If the node corresponding to itemset in transaction exists the count of the node is increased, otherwise a new node is generated and the count is set to 1. FP-Tree also has a frequent-item header table that holds heads of node-links, that connect nodes of same item in FP-Tree. The node-links facilitate item traversal during frequent pattern

mining.

The second stage of the algorithm is to mine frequent patterns from this FP-Tree. FP-Growth then traverses nodes in the FP-Tree starting from the least frequent item in F-list. The node-link originating from each item in the frequent-item header table connects the same item in FP-Tree. While visiting each node, FP-Growth collects the prefix-path of the node, that is the set of items on the path from the node to the root of the tree. FP-Growth stores the count on the node as the count of the prefix-path. The prefix-paths form the so called *conditional pattern base* of that item.

The conditional pattern base is a relatively small database of patterns which co-occur with the item. Then FP-Growth creates small FP-Tree from the conditional pattern base called $conditional\, FP-Tree$. The process is recuresively iterated until no conditional pattern base can be generated and all frequent patterns that contain the item are discovered. The same iterative process is repeated for other frequent items in the F-list.

Parallelization of the FP-Growth algorithm on a PC cluster presented in [**11**] exploits the fact that the processing of a conditional pattern base is independent of the processing of other conditional pattern bases. After the first scan of the database the support counts of all items are exchanged to determine globally frequent items and identical F-lists are built at each node. At the second database scan, each node builds local FP-Tree from local transaction database with respect to the global F-list. To find all locally frequent patterns local conditional pattern bases are generated from the FP-Tree. Then the counts are communicated and globally frequent patterns are determined.

The paper demonstrates speed-up ratio of 8 with 32 nodes which is not very promising. More extensive experiments should be conducted, most importantly, on larger than 100K transactions databases in order for this work to be adequately considered. FP-Growth approach also has other limitations; the algorithm requires two scans of the database, and there is a potential problem that FP-Tree cannot fit into memory.

## 1.5 Sequential Sampling

In 1996 Toivonen presented a single scan algorithm called Sampling [23]. The idea behind Sampling is simple. A random sample of the database is used to predict all the frequent itemsets, which are then validated in a single database scan. Because this approach is probabilistic, and therefore fallible, not only the frequent itemsets are counted in the scan but also their negative border. If the scan reveals that itemsets which were predicted to belong to the negative border are frequent, then all their possible frequent supersets are generated. A second scan is then performed to discover whether they are also frequent. To further reduce the chance of failure, Toivonen suggested that mining be performed using

some $low_fr < MinFreq$, and the results reported only if they pass the original $MinFreq$ threshold. He also gives a heuristic which can be used to determine $low_fr$. The cost of using $low_fr$ is an increase in the number of candidates. The Sampling algorithm and the DIC algorithm (Brin 1997 [4]) are the only single-scan sequential ARM algorithms known today. The performance of the two is thus unrivaled by any other sequential ARM algorithm.

## 2 D-Sampling

The distributed algorithms described in the previous section are based on Apriori. Indeed, all parallel algorithms that have been presented until today are level-wise and require multiple database scans. The reason why no distributed form of Sampling was suggested in the six years since its presentation may lie in the communication complexity of the problem. As we have seen, the communication complexity of D-ARM algorithms is highly dependent on the number of candidates and on the noise level in the partitioned database. When Sampling reduces the database through sampling and lowers the $MinFreq$ threshold, it greatly increases both the number of candidates and the noise level. This may render a distributed algorithm useless.

This is the reason that the reduced communication complexity of DDM seems to offer an opportunity. The main idea of D-Sampling is to utilize DDM to mine a distributed sample using $low_fr$ instead of $MinFreq$. After $\mathcal{F}_{low_fr}\left[\overline{S}\right]$ has been identified, the partitioned database is scanned once in parallel, to find the actual frequencies of $\mathcal{F}_{low_fr}\left[\overline{S}\right]$ and its negative border. Those frequencies can then be collected and rules can be generated from itemsets more frequent than $MinFreq$.

We added three modifications to this scheme. First, although the given DDM is level-wise, here it is executed on a memory resident sample. Thus we could modify DDM to develop new itemsets on-the-fly and calculate their estimated frequency with no disk-I/O. Second, a new method for the reduction of $MinFreq$ to $low_fr$ yielded two additional benefits: it is not heuristic, i.e., our error bound is rigorous, and it produces many less candidates than the method suggested previously. Third, after scanning the database, it would not be wise to just collect the frequencies of all candidates. Since these candidates were calculated according to the lowered threshold, few of them are expected to have frequencies above the original $MinFreq$. Instead, we run DDM once more to decide which candidates are frequent and which are not. We call the algorithm D-Sampling (Algorithm 1).

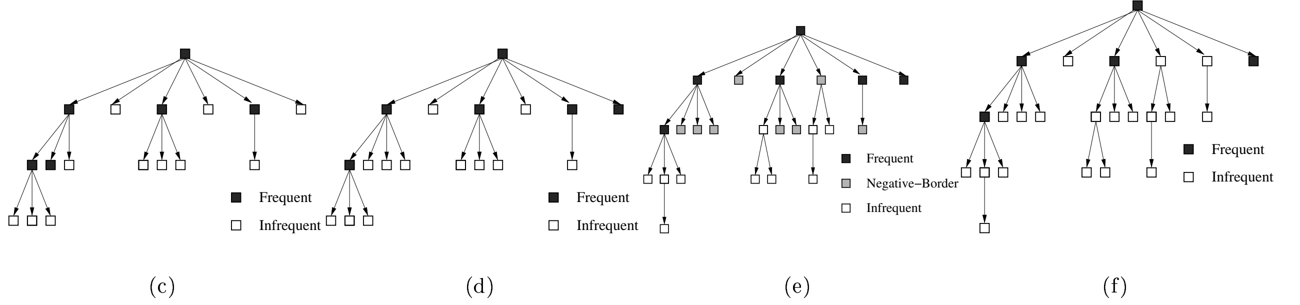



Figure 3: The development of the trie throughout D-Sampling: First (c) the trie is developed according to the local frequencies of the itemsets. Then (d) MDDM is performed once and the estimated globally frequent itemsets are identified. The error reduction phase (e) follows, by the end of which low_fr is set and the itemsets which are frequent according to this threshold are identified. At this stage the negative border is calculated, the database is scanned and actual frequencies are counted for the combined candidate set. Finally, in (f) MDDM is run once more with these frequencies and the original MinFreq. The frequent itemsets are identified. If one of them belongs to the negative border, failure is reported; otherwise, rules are calculated.

---

**Algorithm 1** D-Sampling

---

**For node $i$ out of $n$**
**Input:**
$MinFreq$, $MinConf$, $DB^i$, $s$, $M$, $\delta$
**Output:**
The set of confident associations between globally frequent itemsets
**Main:**
Set $p_error \leftarrow 1$,$low_fr \leftarrow MinFreq$
Load a sample $S^i$ of size $s$ from $DB^i$ into memory
Initialize the trie with all the size-1 itemsets and calculate their $TID$ lists
$\mathcal{F}_{low_fr}\left[\overline{S}\right] \leftarrow MDDM\left(MinFreq\right)$
While $p_error > \delta$

1. $\mathcal{F}_{low_fr}\left[\overline{S}\right] \leftarrow \mathcal{F}_{low_fr}\left[\overline{S}\right] \cup M_Max\left(M\right)$
2. Set $low_fr$ to the frequency of the least frequent itemset in $\mathcal{F}_{low_fr}\left[\overline{S}\right]$
3. Set $p_error$ to the new error bound according to $MinFreq$, $low_fr$ and $\mathcal{F}_{low_fr}\left[\overline{S}\right]$

Let $C$ be $\mathcal{F}_{low_fr}\left[\overline{S}\right] \cup Negative_Border\left(\mathcal{F}_{low_fr}\left[\overline{S}\right]\right)$
Scan the database and compute $Freq\left(c, DB^i\right)$ for each $c \in C$. Update the frequencies in the trie to the computed ones
Compute $\mathcal{F}_{MinFreq}\left[\overline{DB}\right]$ by running $MDDM\left(MinFreq\right)$, this time with the actual frequencies
If exists $c \in \mathcal{F}_{MinFreq}\left[\overline{DB}\right]$ such that $c \notin \mathcal{F}_{low_fr}\left[\overline{S}\right]$ (i.e., from negative border) report failure
$Gen_Rules\left(\mathcal{F}_{MinFreq}\left[\overline{DB}\right], MinConf\right)$

---

## 2.1 Algorithm

D-Sampling begins by loading a sample into memory. The sample is stored in a trie – a lexicographic tree. This trie is the main data structure of D-Sampling and is accessed by all its subroutines. Each node of the trie stores, in addition to structural information (parents, descendants etc.), the list of $TID$'s of those transactions that include the itemset associated with this node. These lists are initialized from the sample for the first level of the trie; when a new trie node – and itemset – are developed, the $TID$ lists of two of the parent nodes are intersected to create the $TID$ list of the new node.

Figure 3 describes the development of the trie throughout D-Sampling. The first step of D-Sampling is to run a modification of DDM on the distributed sample. Then, in order to set $low_fr$, the algorithm enters a loop; in each cycle through the loop it calls another DDM derivative called M-Max to mine the next $M$ estimated-frequent itemsets. $M$ is a tunable parameter we set to about 100. After it finds those additional itemsets, D-Sampling reduces $low_fr$ to the estimated frequency of the least frequent one and re-estimates the error probability using a formula described in section 3. When this probability drops below the required error probability, the loop ends. Then, D-Sampling creates the final candidate set $C$ by adding to $\mathcal{F}_{low_fr}\left[\overline{S}\right]$ its negative border.

Once the candidate set is established, each partition of the database is scanned exactly once and in parallel, and the actual frequencies of each candidate are calculated. With these frequencies D-Sampling performs yet another round of the modified DDM. In this round the original $MinFreq$ is used; thus, unless there is a failure, no candidates outside the negative border need be used. If indeed no failure occurs, then all frequent itemsets will be evaluated according to the actual frequencies which were found in the database scan. Hence, after this round it is known which of the candidates in $C$ are globally frequent and which are not. In this case, rules are generated from $\mathcal{F}_{MinFreq}\left[\overline{DB}\right]$ using the known global frequencies.

If an itemset belonging to the negative border of $\mathcal{F}_{low_fr}\left[\overline{S}\right]$ does turn out to be frequent, this means that D-Sampling has failed: a superset of that candidate, which was not counted, might also turn out to be frequent. In this case we suggest the same solution offered by Toivonen: to create a group of additional candidates that includes all combinations of anticipated and unanticipated frequent itemsets, and then perform an additional scan. The size of this group is limited by the number of anticipated frequent itemsets times the number of possible combinations of unanticipated frequent itemsets.[2] Since failures are very rare events, and the probability of multiple failure is exponentially small, the additional

[2]The number of existing frequent patterns $|L|$ is typically only a few thousands (the rest hundreds of thousands is the negative border). If there are $l$ independent misses, the number of candidates to check in the second pass would be $2^l \cdot |L|$ at most. The probability for $l$ independent misses is $\delta^l$.

scan will incur costs that are of the same scale as the first scan.

## 2.2 MDDM – A Modified Distributed Decision Miner

The original DDM algorithm, as described in section 1.4, is level-wise. When the database is small enough to fit into memory, the level-wise structure of the algorithm becomes superfluous. Modified Distributed Decision Miner, or MDDM (Algorithm 2), therefore starts by developing all the locally frequent candidates, regardless of their size. It then continues to develop candidates whenever they are required, i.e., when all their subsets are assumed frequent (according to the local hypothesis - $P$) or when another node refers to the associated itemset.

The remaining steps of MDDM are the same as DDM. Each party looks for itemsets for which the global hypothesis and local hypothesis disagree and communicate their local counts to the rest of the parties. When no such itemset exists, the party passes (it can return to activity if new information arrives). If all of the parties pass, the algorithm terminates and the itemsets which are predicted to be frequent according to the public hypothesis $H$ are the estimated globally frequent ones.

Figure 4 exemplifies the development of the trie as messages are sent and received. First, the locally frequent itemsets are developed, their TID lists calculated, and their public hypothesis and private hypothesis evaluated ($H$ and $P$ respectively); the starting value of $H$ is zero and that of $P$ is the local frequency. As messages are received, those values change. Itemsets are sent when their $H$ and $P$ are on opposite sides of $MinFreq$. Therefore, in this toy example, where $MinFreq$ is 0.75, itemset $\{1\}$ is sent. When a message is received about an itemset which has already been developed (as is the case for $\{2\}$, $\{3\}$ and $\{4\}$), it causes the reevaluation of $H$ and $P$. If a message is received for an itemset which has not yet been developed (as is the case for $\{3,4\}$), it is developed on-the-fly and its local frequency is calculated.

## 2.3 M-Max Algorithm

The modified DDM algorithm identifies all itemsets with frequency above $MinFreq$. D-Sampling, however, requires a further decrease in the frequency of itemsets which are included in the database scan. The reason for this, as we shall see in section 3, is that three parameters affect the chances for failure. These are the size of the sample $N$, the size of the negative border, and the estimated frequency of the least frequent candidate. The first parameter is given, the second is a rather arbitrary value which we can calculate or bound, and the last parameter is the one we can control.

**Algorithm 2** Modified Distributed Decision Miner

**For node $i$ out of $n$**

**Input:**

$fr$ – the target frequency

**Output:**

$\mathcal{F}_{fr}\left[\overline{S}\right]$

**Definitions:**

$$P\left(X,S^i\right)=\sum_{j\in G(X)}\frac{\left|S^j\right|Freq\left(X,S^j\right)}{|S|}+$$
$$\sum_{j\notin G(X)}\frac{\left|S^j\right|Freq\left(X,S^i\right)}{|S|}$$

$$H\left(X\right)=\begin{cases}\frac{\sum_{j\in G(X)}\left|S^j\right|Freq\left(X,S^j\right)}{\sum_{j\in G(X)}\left|S^j\right|} & \mathrm{G}\left(X\right)\neq\emptyset\\ 0 & otherwise\end{cases}$$

**Main:**

Develop all the candidates which are more frequent than $fr$ according to $P$

Do

- Choose a candidate $X$ that was not yet chosen and for which either $H\left(X\right)<fr\leq P\left(X,S^i\right)$ or $P\left(X,S^i\right)<fr\leq H\left(X\right)$
- Broadcast $m=\left\langle id\left(X\right),Freq\left(X,S^i\right)\right\rangle$
- If no such itemset exists broadcast $\langle pass\rangle$

Until $|Passed|=N$

$R\leftarrow$all $X$ with $H\left(X\right)\geq fr$

Return $R$

**When node $i$ receives a message $m$ from party $j$:**

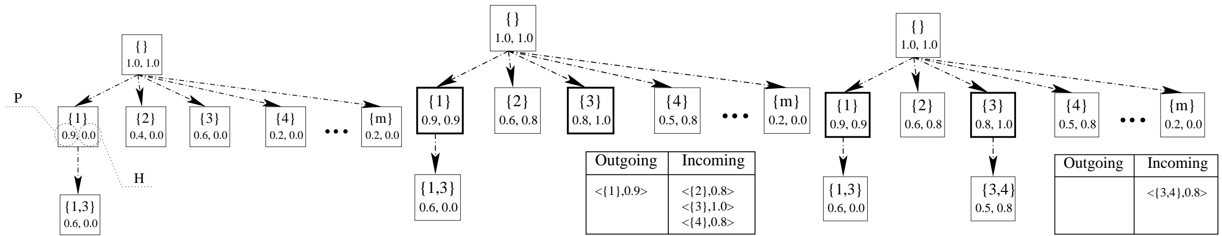
1. If $m=\langle pass\rangle$ insert $j$ into $Passed$
2. Else $m=\left\langle id\left(X\right),Freq\left(X,S^j\right)\right\rangle$

   If $j\in Passed$ remove $j$ from $Passed$

   If $X$ was not developed then: develop it, set $G\left(X\right)=\emptyset$, calculate $X.tid_list$ by intersecting the $TID$ lists of two of $X$'s immediate subsets and set $Freq\left(X,S^i\right)=\frac{|X.tid_list|}{|S^i|}$

   Insert $j$ to $G\left(X\right)$

   Recalculate $H\left(X\right)$ and $P\left(X,S^i\right)$




(a) The trie is initialized with the size-1 itemsets and then developed until no more locally frequent itemsets can be found. Written below the itemset are P – the private hypothesis and H – the global hypothesis.At first, the H values are all zero.

(b) Some of the itemsets may be selected and sent – as in the case of $\{1\}$. Others, like $\{2\}$, $\{3\}$ and $\{4\}$, may have the value of their hypotheses changed because of incoming messages.

(c) A message may arrive concerning an itemset which has not yet been developed. In that case, it is developed and inserted into the trie.

Figure 4: The development of the trie throughout MDDM

The frequency of the least frequent candidate can be controlled by reducing $low_fr$. However, this must be done with care: lowering the frequency threshold increases the number of candidates. This increase depends on the distribution of itemsets in the database and is therefore nondeterministic. The larger number of candidates affects the scan time: the more candidates you have, the more comparisons must be made per transaction. In a distributed setting, the number of candidates is also strongly tied to the communication complexity of the algorithm.

To better control the reduction of $low_fr$, we propose another version of DDM called M-Max (Algorithm 3). M-Max increases the number of frequent itemsets by a given factor rather than decreasing the threshold value by an arbitrary value. Although worst case analysis shows that an increase of even one frequent itemset may require that any number of additional candidates be considered, the number of such candidates tends to remain small and roughly proportional to the number of additional frequent itemsets. We complement this algorithm with a new bound for the error (presented in section 3). The combined scheme is both rigorous and economical in the number of candidates.

The M-Max algorithm is based on the inference that changing the $MinFreq$ threshold to the $H$-value of the $M$-largest itemset[3] every time an itemset is developed or a hypothesis value is changed will result in all parties agreeing on the $M$ most frequent itemsets when DDM terminates. This is easy to prove. Take any final state of the modified algorithm. The $H$ value of each itemset is equal in all parties; hence, the final $MinFreq$ is equal in all parties as well. Now compare this state to the corresponding state under DDM, with the static $MinFreq$ value set to the one finally agreed upon. The state attained by

[3] $P$ is used when the $M$ largest $H$ is zero.

M-Max is also a valid final state for this DDM. Thus, by virtue of DDM correctness, all parties must be in agreement on the same set of frequent itemsets.

As a stand-alone ARM algorithm, M-Max may be impractical because a node may be required to refer to itemsets it has not yet developed. If the database is large, this would require an additional disk scan whenever new candidates are developed. Nevertheless, at the $low_fr$ correction stage of D-Sampling, the database is the memory-resident sample. It is thus possible to evaluate the frequency of arbitrary itemsets with no disk-I/O.

# 3 Statistical Analysis

Two statistical issues should be settled in order to validate that D-Sampling has the required failure probability. The first is bounding the probability of failure that follows the error adjustment phase. The second is showing how a distributed database can be sampled uniformly.

## 3.1 A Bound on the Sampling Error

Let $0 < fr < 1$ be the frequency of some arbitrary itemset $X$ in $DB$. Consider a random sample $S$ of size $N$ from $DB$. We will assume that transactions in the sample are independent. Hence, the number of rows in $S$ which contain $X$ can be seen as a random variable, $x \sim Bin(N, fr)$.

The frequency of $X$ in $N$ transactions, $s_fr = x/N$, is an estimate for $fr$, which improves as $N$ increases. The best-known way to bound the chance that $s_fr$ will deviate from $fr$ is with the Chernoff bound. We use a tighter bound for the case of binomial distributions (see Hagerup and Rub [8]):

$$Pr\left(|fr - s_fr| > \varepsilon\right) \leq \left[\left(\frac{1-fr}{1-s_fr}\right)^{1-s_fr}\left(\frac{fr}{s_fr}\right)^{s_fr}\right]^{N} \quad (1)$$

**Lemma:** Given a random uniform sample $S$ of N transactions from $DB$, a frequency threshold $MinFreq$, the lowered frequency threshold $low_fr$, and the negative border of $\mathcal{F}_{low_fr}[S]$, denoted $NB$, the probability $p_{failure}$ that any $X \in NB$ will have frequency larger than or equal to $MinFreq$ (hence causing failure) is bounded by:

$$|NB| \cdot \left[\left(\frac{1-MinFreq}{1-low_fr}\right)^{1-low_fr}\left(\frac{MinFreq}{low_fr}\right)^{low_fr}\right]^{N} \quad (2)$$

**Proof:** For any specific itemset in $NB$, the probability that this itemset will cause failure is the probability that its estimated frequency is below $low_fr$ while its actual frequency is above $MinFreq$.

**Algorithm 3** M-Max

**For node $i$ out of $n$**
**Input:**
$low_fr$
**Output:**
The M most frequent itemsets not yet in $\mathcal{F}_{low_fr}\left[\overline{S}\right]$
**Definitions:** same as for algorithm 2.2
Let $B$ denote the initial size of $\mathcal{F}_{low_fr}\left[\overline{S}\right]$, $fr = low_fr$
**Main:**
Do

1. call $set_fr$
2. Choose $X$ that was not yet chosen and for which either $H(X) < fr \leq P(X, S^i)$ or $P(X, S^i) < fr \leq H(X)$
   Broadcast $m = \langle id(X), Freq(X, S^i) \rangle$
3. If no such itemset exists broadcast $\langle pass \rangle$

Until $|Passed| = N$
$R \leftarrow$all $X$ in the trie with $H(X) \geq fr$ which are not in $\mathcal{F}_{low_fr}\left[\overline{S}\right]$
Return $R$
**When node $i$ receives a message $m$ from party $j$:**

1. If $m = \langle pass \rangle$ insert $j$ into $Passed$
2. Else $m = \langle id(X), Freq(X, S^j) \rangle$
   If $j \in Passed$ remove $j$ from $Passed$
   If $X$ was not developed then: develop it, set $G(X) = \emptyset$, Calculate $X.tid_list$ by intersecting the $TID$ lists of two of $X$'s immediate subsets and set $Freq(X, S^i) = \frac{|X.tid_list|}{|S^i|}$
   Insert $j$ to $G(X)$
   Recalculate $H(X)$ and $P(X, S^i)$
   call $set_fr$

**procedure $set_fr$:**
Do M times:

- Select the next most frequent itemset outside $\mathcal{F}_{low_fr}\left[\overline{S}\right]$ and develop its descendants if they were not developed yet

Set $fr$ to the $H$ value of the last itemset selected. For itemsets with $H = 0$ consider $P$ instead.

Substituting $MinFreq$ for $fr$ and $low_fr$ for $s_fr$, the bound gives us:

$$Pr\left(\left|Freq\left(X,DB\right)-Freq\left(X,S\right)\right|>\epsilon\right)\leq$$

$$Pr\left(\left|MinFreq-low_fr\right|>\epsilon\right)\leq$$

$$\left[\left(\frac{1-MinFreq}{1-low_fr}\right)^{1-low_fr}\left(\frac{MinFreq}{low_fr}\right)^{low_fr}\right]^{N} \tag{3}$$

As for the entire $NB$:

$$Pr\left(\exists X\in NB: X\ fails\right)\leq\sum_{X\in NB}Pr\left(X\ fails\right)\leq$$

$$\left|NB\right|\left[\left(\frac{1-MinFreq}{1-low_fr}\right)^{1-low_fr}\left(\frac{MinFreq}{low_fr}\right)^{low_fr}\right]^{N} \tag{4}$$

Since calculating the negative border is by itself a costly process, we choose to relax this bound by substituting $|I|\left|\mathcal{F}_{low_fr}[S]\right|$ for $|NB|$. Obviously, any itemset in $\mathcal{F}_{low_fr}[S]$ can only be extended by at most $|I|$ items, and thus this relaxed bound holds.

**Corollary (Toivonen 1996):** If none of the itemsets in the negative border caused failure, then no other itemset can cause failure.

**Proof:** Any other itemset $X$ outside $\mathcal{F}_{low_fr}[S]$ and $NB$ must include a subset from $NB$. Hence its frequency must be less than or equal to the frequency of this subset. It follows that if the frequency of each itemset in $NB$ is below $MinFreq$, so is the frequency of $X$.

### 3.2 Uniformly Sampling a Partitioned Database

Uniform sampling is not a simple task in any database. At worst it may require as much as a full scan of the database to ensure uniformity. Partitioning the database, as we do, adds a further complication. Here we show that any existing method for uniformly sampling a single database can be leveraged into a scheme for sampling partitioned databases.

The scheme we use is simple. In order to randomly choose a single transaction from the partitioned database, we first uniformly choose a partition[4] and then uniformly choose a transaction from the chosen partition. Extending this to a sample of size $|S|$, we first choose randomly, for each transaction in the

[4]If the partitions are not equal in size, this choice is weighted according to the partition sizes.

sample, the partition from which it will be sampled. Then, knowing exactly how many transactions should be sampled from each partition, we randomly choose the correct number. Note that the theoretical bound we use allows sampling with repetitions; the algorithm, however, will require slight modifications for a single $TID$ to appear twice in the sample.

This does not yet mean that D-Sampling works well with every partitioned sample. Since local sample sizes are selected randomly, one of these local samples may be small. Small samples are, by definition, noisier than large ones. Since the performance of DDM depends on $Pr_{above}$ and hence on the noisiness of the data, a sample which is biased against a specific partition may result in a longer run time.

The choice of the number of transactions to be sampled from each partition is distributed multinomially. The expected number of transactions from each of the $n$ partitions is hence $\frac{|S|}{n}$. Since we choose the partitions independently, we can apply the Chernoff bound to the size of the sample from a specific partition:

$$Pr\left(\left|S^i\right| \leq (1-\epsilon)\frac{|S|}{n}\right) \leq e^{-\frac{\epsilon^2|S|}{2n}} \quad (5)$$

Taking $\epsilon = 10\%$, we get $Pr\left(\left|S^i\right| \leq 0.9\frac{|S|}{n}\right) \leq e^{-\frac{|S|}{200n}}$. In our experiments, $|S| = 80,000 \cdot n$. This is based on the size of the sample in Toivonen's experiments: between $20,000$ and $80,000$ transactions. The chance of having a 10% smaller sample with these figures is negligible: less than $e^{-400}$. Obviously, a 10% difference in sample size will not have any noticeable effect on the noise level or on the run time.

Since the chances of a sample that is largely biased toward a specific partition are slim, the best thing to do if such a sample does occur is to sample once again. Moreover, in many practical scenarios it is known that the partitioning of the data was random. In that case, it is justified to simply sample an equal portion of each partition. In our experiments, we used this last method.

# 4 Experiments

We carried out four sets of experiments. The first set tested D-Sampling to see how much faster it is to run the algorithm with the database split among $n$ machines than to run it on a single node. The second set compared D-Sampling with FDM and DDM on a range of $MinFreq$ values. The third set checked scale-up: the change in runtime when the number of machines is increased together with the size of the database. The fourth set tested how many candidates are generated by D-Sampling in comparison to the minimal amount of candidates in FDM and Apriori.

We ran our experiments on two clusters: the first cluster, which was used for the first, second and

fourth sets of experiments, is a cluster of 15 Pentium computers with dual 1.7GHz processors. Each computer has at least 1 gigabyte of main memory. The computers are connected via an Ethernet-100 network. The second cluster, which we used for the scale-up experiments, is composed of 32 Pentium computers with dual 500MHz processors. Each computer has 256 megabytes of memory. The second cluster is also connected via an Ethernet-100 network.

All of the experiments were performed with synthetic databases produced by the standard gen tool [17]. The databases were built with the same parameters used by Toivonen in [23]. The only change we made was to enlarge the databases to about 18 gigabytes each; had we used the original sizes, the whole database would fit, when partitioned, into the memory of the computers. The database T5.I2.D600M has 600M transactions, each containing an average of five items, and patterns of length two. T10.I4.D375M and T20.I6.D200M follow the same encoding. When the database was to be partitioned, we divided it arbitrarily by writing transaction number $TID$ into the $TID\%n$ partition.

## 4.1 Speedup Results

The speedup experiments were designed to demonstrate that parallelization works well for Sampling. We thus ran D-Sampling with $n = 1$ (with $n = 1$, D-Sampling reverts to Sampling) on a large database. Then we tested how splitting the database between $n$ computers affects the algorithm's performance.

As figure 5 shows, the basic speedup of D-Sampling is slightly sublinear. However, when the number of candidates is large, the speed-up becomes superlinear. This is because the global sample size increases with the number of computers. This larger sample size translates into a higher $low_fr$ value and thus to a smaller number of candidates than with $n = 1$.

For completeness, we included FDM and DDM speedup graphs. They start off at $n = 1$ with roughly double the runtime of D-Sampling and then speed up sublinearly. For proper comparison, we normalized their speedup relative to the runtime of D-Sampling with $n = 1$. That is why, for $n = 1$, the graph shows that FDM and DDM have a 50% slowdown.

## 4.2 Dependency on $MinFreq$

The second set of experiments (figure 6) demonstrates the dependency of D-Sampling performance on $MinFreq$, which determines the number and size of the candidates. We compare the D-Sampling runtime to that of FDM and DDM. D-Sampling turns out to be insensitive to the reduction in $MinFreq$; its runtime increases by no more than 50% across the whole range. On the other hand, the runtimes of FDM

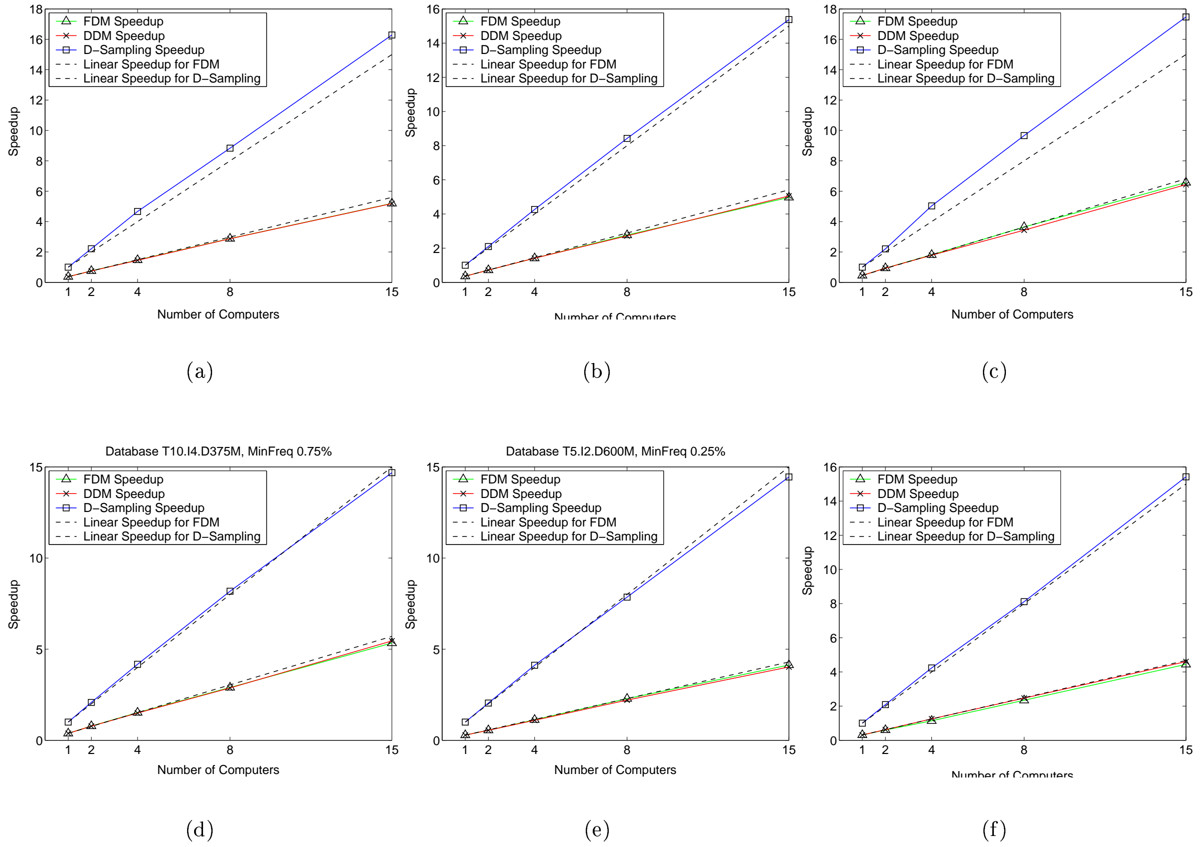
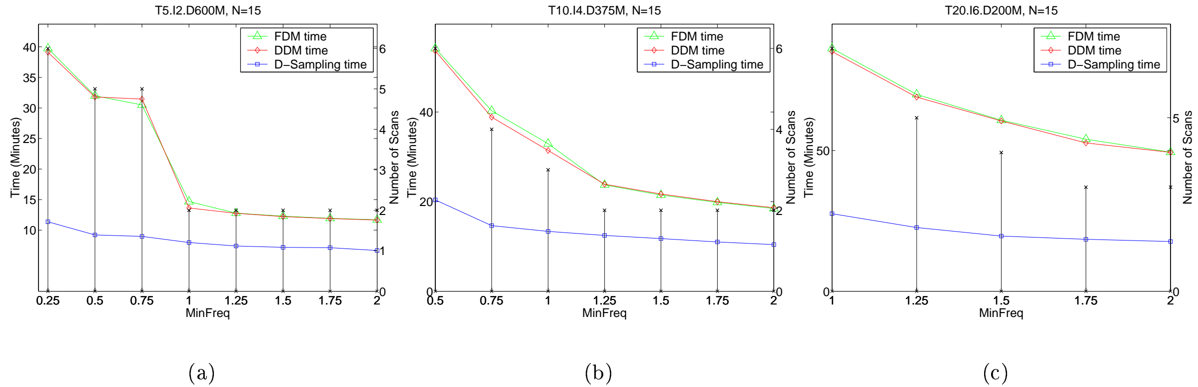



Figure 5: Figures (a) through (f) show the speedup of D-Sampling versus that of FDM and DDM when the database is partitioned among ever larger numbers of computers. Typical speedup of D-Sampling is superlinear: the larger communication load is more than compensated for by the reduction in the number of candidates resulting from the larger sample. When the number of candidates is low to start with, the larger sample does not reduce this number to an extent that compensates for the larger communication load. Hence, the slight sublinearity in some of the tests.

and DDM increase rapidly as $MinFreq$ is decreased. This is because of the additional scans required as increasingly larger itemsets become frequent. Because it performs just one database scan, D-Sampling is expected to be superior to any level-wise D-ARM algorithm, just as Sampling is superior to all level-wise ARM algorithms.




(a) (b) (c)

Figure 6: Figures (a) through (c) compare D-Sampling, FDM and DDM runtimes with varying $MinFreq$. D-Sampling is consistently better than FDM and DDM. Its superiority increases when $MinFreq$ is reduced and larger itemsets become frequent. This is because FDM and DDM performance is tightly linked with the number of database scans they perform, while D-Sampling performance is only mildly affected by the larger number of candidates. The same results were achieved for every number of machines we tested.

## 4.3 Scale-up

The third set of tests is aimed at testing the scalability of D-Sampling. Here the partition size is fixed. We use a database of about 1.5 gigabytes on each computer. A scalable algorithm should have the same runtime regardless of the number of computers.

D-Sampling creates the same communication load per candidate as DDM. However, because it generates more candidates, it uses more communication. As can be seen from the graphs in figure 7, D-Sampling is scalable in two of the tests. In fact, for mid-range numbers of computers, D-Sampling runs even faster than with $n = 1$; this is due to the superlinear speed-up discussed earlier. The mild slowdown seen in figure 7.c is due to the smaller average pattern size and the smaller number of candidates in T5.I2.D600M. The larger the number of candidates, the greater the saving in candidates when the number of computers increases. If there are enough large patterns, this saving will compensate for the increasing communication overhead. Such is not the case, however, with T5.I2.D600M.

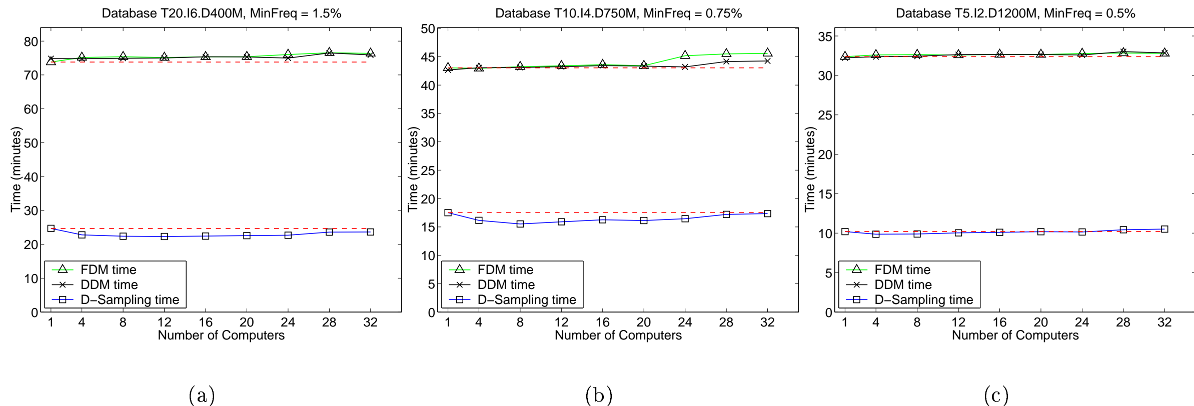



Figure 7: D-Sampling has almost fixed runtime regardless of the number of machines. FDM, in comparison, degrades with the number of machines. The reason for the mild slowdown of D-Sampling in figure (c) is due to the smaller patterns in this database. These smaller patterns decrease the scan time, leaving D-Sampling little room for improvement when the number of computers increases. The slight speedup for FDM seen in figure (a) between $n = 1$ and $n = 4$ occured because the larger combined database did not produce any size-4 candidates. This reduced the number of scans from 4 to 3.

## 4.4 Number of Candidates

| Trans. length | No. items | MinFreq (%) | FDM | D-Sampling Hagerup | D-Sampling Chernoff |
|---|---|---|---|---|---|
| 5 | 1000 | 0.5 | 66172 | 90803 (37%) | 231080 (249%) |
| 5 | 2000 | 0.5 | 72841 | 111868 (53%) | 469169 (544%) |
| 10 | 1000 | 0.75 | 104623 | 121864 (16%) | 214164 (104%) |
| 10 | 2000 | 0.75 | 122721 | 149220 (21%) | 406376 (231%) |
| 20 | 1000 | 1 | 170314 | 183348 (7%) | 266502 (56%) |
| 20 | 2000 | 1 | 248995 | 279910 (12%) | too many |

Table 1: The number of candidates produced by FDM, D-Sampling using the Chernoff bound (as suggested by Toivonen), and D-Sampling using the Hagerup bound, for various databases.

Since the main disadvantage of D-Sampling algorithm is the large number of candidates it generates, our last set of experiments is aimed at testing how many of the candidates it generates are actually redundant. We first obtained the optimal number of candidates by running FDM on a set of small databases and then ran D-Sampling on these databases. As before, we used samples of $80K$ transactions and maximum error probability $\delta = 0.001$.

The bound we use produces less candidates than Toivonen's one. Figure 8 compares the number of candidates resulting from Chernoff and from Hagerup error bounds in D-Sampling, as opposed to the number of candidates in FDM. Both $n = 1$ (sequential) and $n = 8$ cases are shown. It can be seen that the

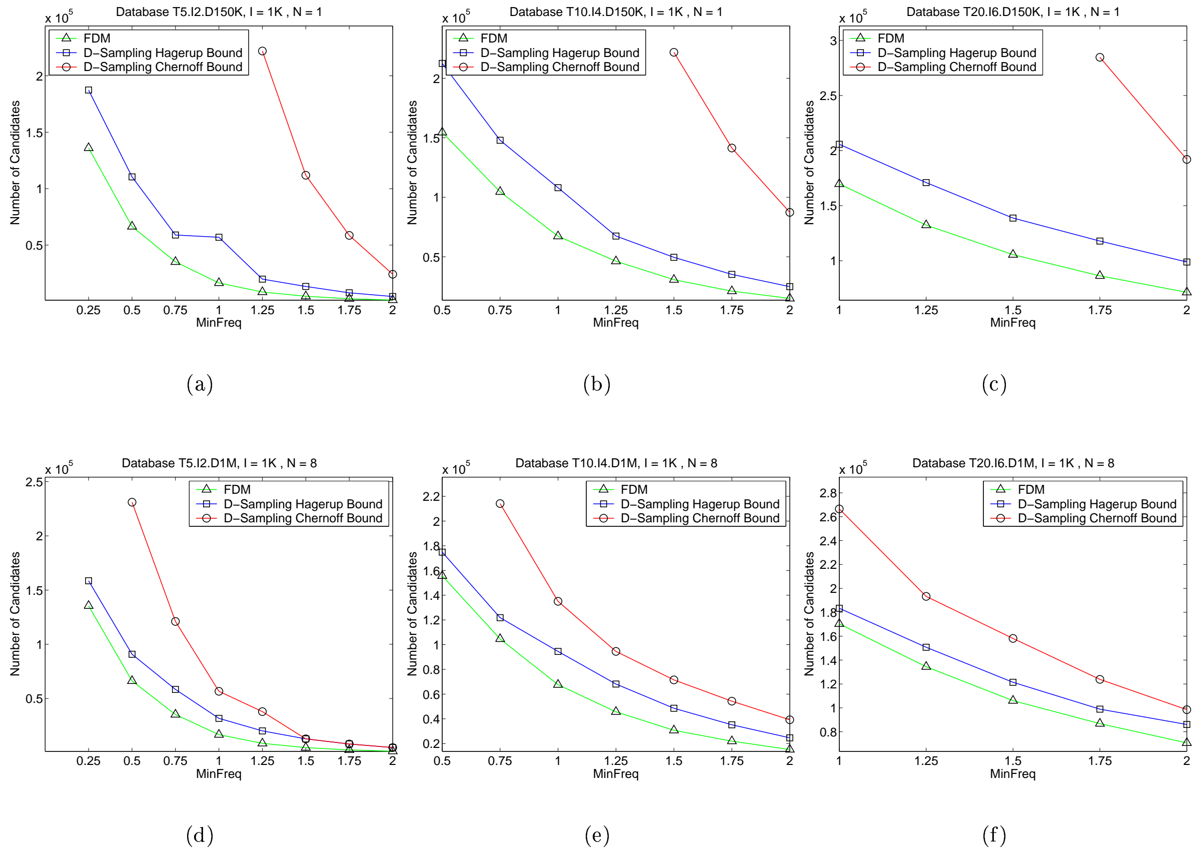



Figure 8: A comparison: the number of candidates produced by FDM, D-Sampling using the Chernoff bound (as suggested by Toivonen), and D-Sampling using the Hagerup bound, for various databases. Figures (a), (b) and (c) compare the sequential algorithms. Figures (d), (e) and (f) compare their parallel counterparts with $N = 8$. Clearly, the Hagerup bound is tighter than the common Chernoff bound, and only improves as the number of computing nodes increases. The overhead of candidates posed by D-Sampling with Hagerup bound is incremental.

number of candidates in D-Sampling is strongly tied to both the bound the algorithm uses for calculating the probability of error and the number of computers running the algorithm. First, the Chernoff bound suggested by Toivonen in sequential Sampling produces relatively many candidates to satisfy the error probability condition. The Hagerup bound we use is tighter and produces significantly fewer candidates. Second, the parallelization drastically reduces the number of candidates. The reduction is up to orders of magnitude as the frequency threshold decreases and the total number of candidates increases.

Table 1 summarizes the overhead of candidates posed by D-Sampling for some databases and values of $MinFreq$. Our experiments show, that D-Sampling does not pose large candidates overhead when compared to the number of candidates generated by FDM.

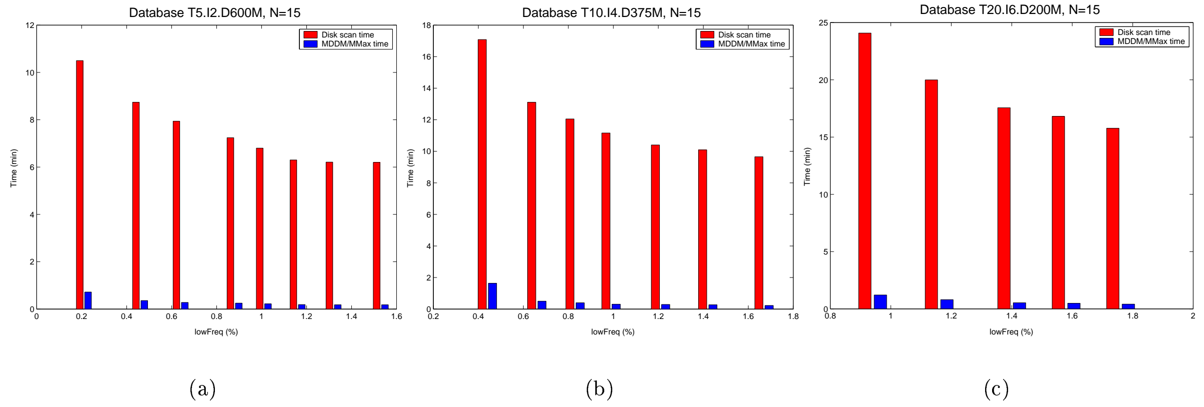
# 5 Discussion

In this section we examine the various stages of the D-Sampling algorithm. In particularly, we are interested in the portion of time each stage borrows from the overall runtime. We show that M-Max is essential for the algorithm's performance, while alternative methods degrade it.

The basic stages of D-Sampling are: A. Use MDDM to mine a sample with frequency threshold $MinFreq$, B. Use M-Max to reduce the frequency threshold, C. Scan the database to collect counts, and D. Use MDDM once more to finally obtain the frequent itemsets. In our experiments we found that stage D of the algorithm takes less than 3% of the total runtime in the worst case. Stages A and B each take less than 5% of the total runtime as well. Figure 9 shows the time spent on A and B together compared to the time of the disk scan (stage C) as the frequency of the least frequent candidate $lowFreq$ varies.

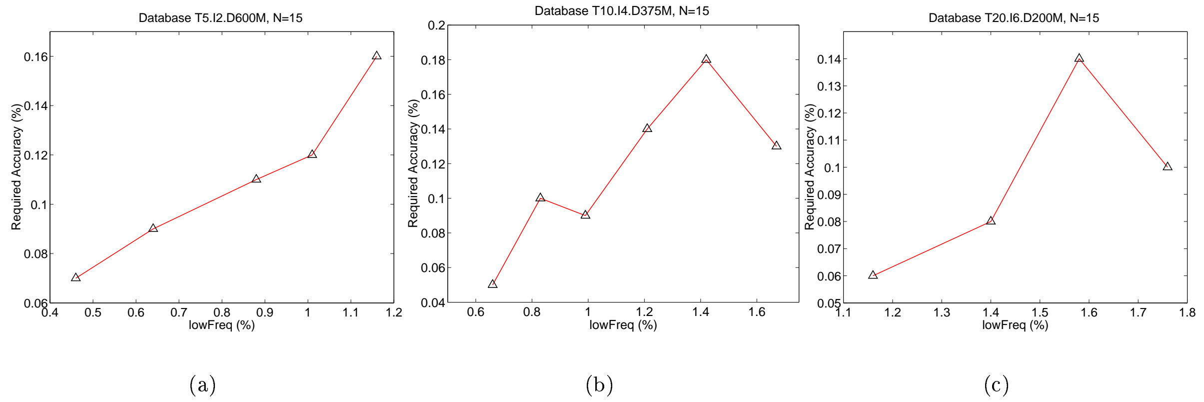
It is obvious from the graphs that the main bottleneck of the algorithm is the disk scan. Moreover, the time of the scan is also strongly tied to $lowFreq$. Lower frequency thresholds produce more candidates, more candidates translate to more checks be made per transaction so the scan time becomes longer. Hence, it is important to fine-tune and provide the minimum number of candidates from sample in order to shorten the scan time and improve performance. M-Max provides us with a mean of disengaging from $lowFreq$ and the arbitrary distribution of itemsets in the database, and building a minimal safety margin gradually and controllably by sticking only to the number of candidates we mine from the sample. The time spent on M-Max is negligible and worthwhile spending compared to the penalty of slowing down the disk scan that may be caused by less accurate techniques which drop the frequency threshold to some arbitrary value.

To estimate the required accuracy of a heuristic that sets $lowFreq$ and does not degrade the scan




(a) (b) (c)

Figure 9: The time spent on MDDM and M-Max together compared to the disk scan time of D-Sampling




(a) (b) (c)

Figure 10: The required accuracy for a potential zero-time heuristic that would set $lowFreq$ and not degrade the disk scan performance of D-Sampling

time by more than the time spent on MDDM and M-Max, we calculated the sensitivity of the scan time at each point from the average of its left- and right-hand side derivatives as function of $lowFreq$. Figure 10 suggests that as $lowFreq$ becomes smaller the accuracy needed for it generally becomes higher due to higher sensitivity of the disk scan time. The graphs show that, in most cases, even a zero-time algorithm replacing MDDM and M-Max with a mistake of less than 0.2% in $lowFreq$ will not produce higher performance.

## 6 Conclusions

We presented a new D-ARM algorithm that uses the communication efficiency of the DDM algorithm to parallelize the single-scan Sampling algorithm. A tight bound on error probability is provided to improve performance. Experiments prove that the new algorithm has superlinear speedup and outperforms FDM and DDM with any $MinFreq$ value. The exact improvement in relation to these algorithms depends on the number of database scans they require. Experiments demonstrate good scalability, provided the database scan is a major bottleneck of the algorithm.

## 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, Melbourne, Australia, 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] Yongjian Fu and Jiawei Han. Meta-rule-guided mining of association rules in relational databases. In *KDOOD/TDOOD*, pages 39–46, 1995.

[8] T. Hagerup and C. Rub. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305 – 308, 1989/90.

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

[10] 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.

[11] P. Iko and M. Kitsuregawa. Parallel fp-growth on pc cluster. In *Seventh Pacific-Asia Conference of Knowledge Discovery and Data Mining (PAKDD03)*, pages 467–473, 2003.

[12] Dao-I Lin and Zvi M. Kedem. Pincer search: A new algorithm for discovering the maximum frequent set. In *Extending Database Technology*, pages 105–119, 1998.

[13] Jun-Lin Lin and M. H. Dunham. Mining association rules: Anti-skew algorithms. In *Proceedings of the 14th Int'l. Conference on Data Engineering (ICDE'98)*, pages 486–493, 1998.

[14] Jian Pei and Jiawei Han. Can we push more constraints into frequent pattern mining? In *Proc. of the ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 350–354, Boston, MA, 2000.

[15] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. *The VLDB Journal*, pages 432–444, 1995.

[16] 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.

[17] 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/.

[18] 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.

[19] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proc. of the 1996 ACM SIGMOD Int'l. Conference on Management of Data*, pages 1–12, Montreal, Quebec, Canada, June 1996.

[20] Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proc. of the ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 67–73. AAAI Press, August 1997.

[21] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti, and Sanjay Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Proc. of the ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 263 – 266, New Port Beach, CA USA, August 1997.

[22] Shiby Thomas and Sharma Chakravarthy. Incremental mining of constrained associations. In *Proceedings of HiPC'00*, pages 547–558, Bangalore, India, 2000.

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

[24] Osmar R. Zaiane, Mohammad El-Hajj, and Paul Lu. Fast parallel association rules mining without candidacy generation. In *IEEE 2001 International Conference on Data Mining (ICDM'2001)*, pages 665–668, 2001.

[25] Mohammed J. Zaki, Srinivasan Parthasarathy, and Wei Li. A localized algorithm for parallel association mining. In *9th Annoual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 321–330, Newprot, Rhode Island, June 1997.

[26] Mohammed Javeed Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency, Special Issue on Parallel Mechanisms for Data Mining*, 7(4):14–25, 1999.

[27] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. Technical Report TR651, Rensselaer Polytechnic Institute, 1997.

[28] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, 1(4):343–373, 1997.

# A A High-Performance Distributed Algorithm for Mining Association Rules

To be presented at the IEEE International Conference on Data Mining (ICDM'03), Florida, USA.