DOI 10.1007/s10115-004-0176-3
Springer-Verlag London Ltd. © 2004
Knowledge and Information Systems (2005) 7: 458–475

Knowledge and Information Systems

A high-performance distributed algorithm for mining association rules

Assaf Schuster[1], Ran Wolff[1], Dan Trock[2]

[1]Department of Computer Science, Technion—Israel Institute of Technology, Haifa, Israel
[2]Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, Israel

Abstract. We present a new distributed association rule mining (D-ARM) algorithm that demonstrates superlinear speed-up with the number of computing nodes. The algorithm is the first D-ARM algorithm to perform a single scan over the database. As such, its performance is unmatched by any previous algorithm. Scale-up experiments over standard synthetic 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. As a result of this tighter bound and by utilizing the combined memory of several computers, the algorithm generates far fewer candidates than comparable sequential algorithms—the same order of magnitude as the optimum.

Keywords: Association rule; Data mining; Distributed data mining; High-performance computing

1. Introduction

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.

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,

Received 19 November 2003
Revised 9 January 2004
Accepted 16 February 2004
Published online 31 August 2004

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 (Agrawal and Srikant 1994), to typically just a single scan in modern ARM algorithms such as Sampling and DIC (Toivonen 1996; Brin et al. 1997).

Much progress has also been made in parallelized algorithms, in which the architecture of the parallel system plays a key role. For instance, many of the proposed algorithms take advantage of the fast interconnect, or the shared memory, of parallel computers. Notable examples include Han et al. (2000) and Zaki et al. (1997b). The latest development is Zaiane et al. (2001), in which each process 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 (Agrawal and Shafer 1996; Cheung et al. 1996), both parallelized versions of Apriori, 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 (Cheung and Xiao 1998). The DDM algorithm (Schuster and Wolff 2001), 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 the 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, it also uses the combined memory to linearly increase the size of the (global) sample. This increase further improves the performance of the algorithm because the safety margin required in sampling decreases accordingly.

Extensive experiments on standard synthetic benchmarks show that D-sampling is superior to previous algorithms in every way. When compared with sampling—one of the best sequential algorithms known today—it offers superlinear speed-up. When compared with FDM, 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.

The rest of this paper is structured as follows: We conclude this section with some notations and a formal definition of the D-ARM problem. In the next section, we present relevant previous work. Section 3 describes the D-sampling algorithm, and Sect. 4 provides the required statistical background. Section 5 describes the experiments we conducted to verify D-sampling performance. We conclude with some open research problems in Sect. 6.

1.1. Notation and problem definition

Let $I = \{i_1, i_2, ..., i_m\}$ be the items in a certain domain. An itemset is a subset of I. A transaction t is also a subset of I that is associated with a unique transaction identifier—*TID*. A database DB is a list of such transactions. Let $\overline{DB} = \{DB^1, DB^2, ..., DB^n\}$ be a partition of DB into n parts. Let S be a list of transactions that were sampled uniformly from DB, and let $\overline{S} = \{S^1, S^2, ..., S^n\}$ be the partition of S induced by $\overline{DB}$. For any itemset X and any group of transactions A, $Support(X, A)$ is the number of transactions in A that contain all the items of X and $Freq(X, A) = \frac{Support(X,A)}{|A|}$. We call $Freq(X, DB^i)$ the local frequency of X in partition i and $Freq(X, DB)$ its global frequency; likewise, we call $Freq(X, S^i)$ the estimated local frequency of X in partition i and $Freq(X, S)$ its estimated global frequency.

For some frequency threshold $0 \leq MinFreq \leq 1$, we say that an itemset X is frequent in A if $Freq(X, A) \geq MinFreq$ 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 kth partition, globally infrequent, etc. The group of all itemsets with frequency above or equal to fr in A is called $\mathcal{F}_{fr}[A]$. The negative border of $\mathcal{F}_{fr}[A]$ is all those itemsets that are not themselves in $\mathcal{F}_{fr}[A]$ but have all their subsets in $\mathcal{F}_{fr}[A]$. 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 $Freq(X \cup Y, DB) \geq MinConf \cdot Freq(X, DB)$.

Definition 1.1. Given a partitioned database $\overline{DB}$ and given *MinFreq* and *MinConf*, the D-ARM problem is to find all the confident rules between frequent itemsets in $\mathcal{F}_{MinFreq}[\overline{DB}]$.

2. Previous work

Since its introduction in 1993, the ARM problem (Agrawal et al. 1993) has been studied intensively. Many algorithms, representing several different approaches, were suggested. Some algorithms, such as Apriori, Partition, DHP, DIC, and FP-growth (Agrawal and Srikant 1994; Savasere et al. 1995; Park et al. 1995a; Brin et al. 1997; Han et al. 1999), are bottom-up, starting from itemsets of size 1 and working up. Others, like Pincer-Search (Lin and Kedem 1998), use a hybrid approach, trying to guess large itemsets at an early stage. Most algorithms, including those cited above, adhere to the original problem definition, while others search for different kinds of rules. These may be implication rules (Brin et al. 1997), generalized rules (Srikant and Agrawal 1994; Han and Fu 1995), quantitative rules (Srikant and Agrawal 1996) or rules constrained to some meta-form (Srikant et al. 1997; Pei and Han 2000; Thomas and Chakravarthy 2000). Finally, the algorithms also differ in the way the data are stored: horizontally as a *TID* with the list of items in that transaction, vertically as an itemset with the list of TIDs in which it appears (Savasere et al. 1995; Ananthanarayana et al. 2000), or a combination of the two (Zaki et al. 1997a).

Algorithms for the D-ARM problem usually can be seen as parallelizations of sequential ARM algorithms. The CD, FDM, FPM and DDM (Agrawal and Shafer 1996; Cheung et al. 1996; Cheung and Xiao 1998; Schuster and Wolff 2001) algorithms parallelize Apriori (Agrawal and Srikant 1994), and PDM (Park et al. 1995b)

parallelizes DHP (Park et al. 1995a). The major difference between parallel algorithms is in the architecture of the parallel machine. This may be shared memory, as in the case of Zaki et al. (1996), Cheung and Xiao (1998) and Zaiane et al. (2001), distributed shared memory, as in Jarai et al. (1998), or shared nothing, as in Agrawal and Shafer (1996), Cheung et al. (1996) and Schuster and Wolff (2001).

The algorithm presented here combines ideas from several groups of algorithms. It first mines a sample of the database and then validates the result. It can thus be seen as a parallelization of the sampling algorithm (Toivonen 1996). The sample is stored in a vertical trie structure that resembles the one in Savasere et al. (1995) and Ananthanarayana et al. (2000), and it is mined using modifications of the DDM (Schuster and Wolff 2001) algorithm, which is Apriori based. We thus include a short description of Apriori and its parallelizations and of the sequential sampling algorithm.

Apriori: A year after the 1993 paper that introduced the ARM problem, Agrawal and Srikant presented Apriori (Agrawal and Srikant 1994). Apriori is a levelwise algorithm for identifying frequent itemsets. It begins by assuming that each item is a candidate to be a frequent itemset of size 1. Then Apriori performs several rounds of a two-phased computation. In the first phase of the kth round, the database is scanned and frequency counts are calculated for all k-sized candidate itemsets (itemsets containing k items). Those candidate itemsets with a frequency above the user-supplied MinFreq threshold are inserted into $\mathcal{F}_{MinFreq}[DB]$. In the second phase, candidate itemsets of size $k+1$ are generated from the frequent itemsets of size k if and only if all their size-k subsets are frequent. The rounds terminate when there are no candidates of size $k+1$. Because it is a levelwise algorithm, Apriori performs exactly k database scans.

Sampling: In 1996, Toivonen presented a single-scan algorithm called sampling (Toivonen 1996). 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 are the frequent itemsets counted in the scan but also their negative border. If the scan reveals that itemsets that were predicted to belong to the negative border are frequent, a second scan is performed to discover whether any superset of these itemsets is also frequent. To further reduce the chance of failure, Toivonen suggests 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 that 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 et al. 1997) are the only single-scan ARM algorithms known today. Their performance is thus unrivaled by any other sequential ARM algorithm.

FDM: Also in 1996, Cheung et al. presented an algorithm called FDM (Cheung et al. 1996). FDM is a parallelization of Apriori to n shared-nothing machines, each with its own partition of the database. At every level and on each machine, the database scan is performed independently on the local partition. Then a distributed pruning technique is employed. The pruning technique is based on the inference that, in order for an itemset to appear in the database at a certain frequency, it must appear with at least that frequency in at least one partition of the database. Thus, in FDM, every party first names those candidate itemsets that are locally frequent in its partition. Next, support counts are globally summed for those candidate itemsets that were named by at least one party. According to the global counts, itemsets are

identified as globally frequent. Those frequent itemsets are used to generate the next level candidates.

If the probability that an itemset has the potential to be frequent is $Pr_{potential}$, then FDM only communicates $Pr_{potential}|C|$ of the itemsets, where C is the group of all candidate itemsets considered by Apriori. The communication complexity of FDM is thus $O(Pr_{potential}|C|n)$. The main problem with FDM is that $Pr_{potential}$ is not scalable in n. It has been shown by Cheung and Xiao that $Pr_{potential}$ quickly increases to 1 as n increases (Cheung and Xiao 1998). The convergence to 1 is especially fast in nonhomogeneous databases: as the nonhomogeneity of the database (measured by a skewness measure) increases or the number of partitions grows, FDM pruning techniques are rendered increasingly ineffective.

DDM: In a previous paper (Schuster and Wolff 2001), we described another Apriori-based D-ARM algorithm—DDM. As in FDM, candidates in DDM are generated levelwise 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 some of the nodes to choose to publish the local frequency of a candidate and others not to. The protocol is directed by two hypotheses that 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 that 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 $O(Pr_{above}|C|n)$, 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.

3. D-sampling algorithm

The distributed algorithms described in the previous section are based on Apriori. Indeed, all parallel algorithms that have been presented until now are levelwise and require multiple database scans[1]. The reason why no distributed form of sampling was suggested in the 6 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 the sampling algorithm samples the database and lowers the *MinFreq* threshold, it greatly increases both the number of candidates and the noise level. This may render a distributed algorithm useless.

[1] The only exception is a parallelization (Zaiane et al. 2001) of the two-scans FP-Growth algorithm (Han et al. 1999). But that algorithm is intended for shared memory machines. When it is executed over clusters of share-nothing machines, its performance quickly degrades as the number of computers grows (Iko and Kitsuregawa 2003).

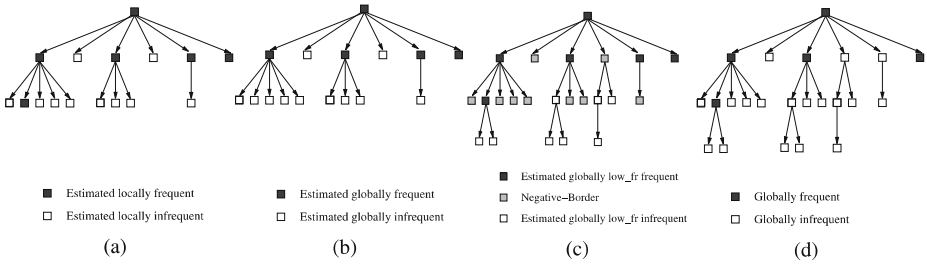
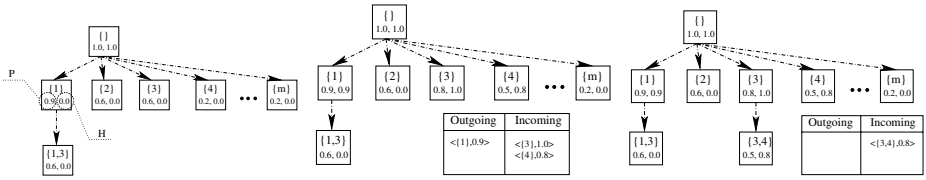


Fig. 1. The development of the trie throughout D-sampling: first (a) the trie is developed according to the local frequencies of the itemsets. Then (b) MDDM is performed once and the estimated globally frequent itemsets are identified. The error reduction phase (c) follows, by the end of which low_fr is set and the itemsets that 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, (d) 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



(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 that has not yet been developed. In that case, it is developed and inserted into the trie.

Fig. 2. The development of the trie during MDDM, assuming two nodes

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 $\mathcal{F}_{low\_fr}[\overline{S}]$ has been identified, the partitioned database is scanned once in parallel to find the actual frequencies of $\mathcal{F}_{low\_fr}[\overline{S}]$ 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 levelwise, 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 has two additional benefits: it uses a rigorous error bound, compared with the heuristic one used in sampling, and it produces far fewer candidates than the rigorous method suggested previously. Third, after scanning the database, it would not be wise to merely collect the frequencies of all candidates. Because 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 modified algorithm D-sampling (Algorithm 1).

Algorithm 1 D-sampling

For node i **out of** n
Input:
$MinFreq$, $MinConf$, DB^i, s, M, δ
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
$\mathcal{F}_{low\_fr}[\overline{S}] \leftarrow MDDM(MinFreq)$
While $p\_error > \delta$

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

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

3.1. 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 TIDs 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 1 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 Sect. 4. When this probability drops below the required error probability, the loop ends. Then D-sampling creates the final candidate set C by adding to $\mathcal{F}_{low\_fr}[\overline{S}]$ 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 to be used. If indeed no failure occurs, then all frequent itemsets will be evaluated according to the actual frequencies that 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 $\mathcal{F}_{MinFreq}[\overline{DB}]$ using the known global frequencies.

If an itemset belonging to the negative border of $\mathcal{F}_{low\_fr}[\overline{S}]$ 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. Because failures are very rare events and the probability of multiple failure is exponentially small, the additional scan will incur costs that are of the same scale as the first scan.

3.2. MDDM—a modified distributed decision miner

The original DDM algorithm, as described in Sect. 2, is levelwise. When the database is small enough to fit into memory, the levelwise 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 in MDDM are the same as in 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 that are predicted to be frequent according to the public hypothesis H are the estimated globally frequent ones.

Figure 2 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 (not all eligible candidates have to be sent on each communication cycle). When a message is received about an itemset that 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 that has not yet been developed (as is the case for $\{3, 4\}$), it is developed on the fly and its local frequency is calculated.

3.3. 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 that are included in the database scan. The reason for this, as we shall see in Sect. 4, is that three parameters affect the chances for failure. These are the size of the sample, N,

Algorithm 2 Modified distributed decision miner

For node i **out of** n
Input:
fr—the target frequency
Output:
$\mathcal{F}_{fr}[S]$
Definitions:

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

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

Main:
Develop all the candidates that are more frequent than fr according to P
Do

- Choose a candidate 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 $m = \langle id(X),\ Freq(X, S^i) \rangle$
- If no such itemset exists, broadcast $\langle pass \rangle$

Until $|Passed| = N$
$R \leftarrow$ all X with $H(X) \geq fr$
Return R
When node i **receives a message** m **from party** j**:**

1. If $m = \langle pass \rangle$, insert j into $Passed$
2. Else $m = \langle id(X),\ Freq(X, S^j) \rangle$
   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 into $G(X)$
   Recalculate $H(X)$ and $P(X, S^i)$

the size of the negative border, and the estimated frequency of the least frequent candidate. The first parameter is given, the second we can calculate or bound and the last parameter is the one we can control.

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 Sect. 4). The combined scheme is both rigorous and economical in the number of candidates.

Algorithm 3 M-Max

For node i **out of** n
Input:
low_fr
Output:
The M most frequent itemsets not yet in $\mathcal{F}_{low\_fr}[\overline{S}]$
Definitions: same as for Algorithm 2
Let B denote the initial size of $\mathcal{F}_{low\_fr}[\overline{S}]$, $fr = low\_fr$
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 $H(X) \geq fr$ that are not in $\mathcal{F}_{low\_fr}[\overline{S}]$
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 into $G(X)$
   Recalculate $H(X)$ and $P(X, S^i)$
   Call *set_fr*

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

- Select the next most frequent itemset outside $\mathcal{F}_{low\_fr}[\overline{S}]$ and develop its descendants if they have not been developed yet

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

The M-Max algorithm is based on the inference that changing the *MinFreq* threshold to the H value of the M largest itemset[2] 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 with the corresponding state under DDM, with the static *MinFreq* value set to the one finally agreed upon. The state attained by 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.

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

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.

4. 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.

4.1. 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 that 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 (1989/90)):

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

Lemma 4.1. 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:

$$|NB| \cdot \left[\left(\frac{1 - MinFreq}{1 - low\_fr} \right)^{1 - low\_fr} \left(\frac{MinFreq}{low\_fr} \right)^{low\_fr} \right]^N .$$

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

$$Pr(|Freq(X, DB) - Freq(X, S)| > \epsilon) \leq$$
$$\left[\left(\frac{1 - MinFreq}{1 - low\_fr} \right)^{1 - low\_fr} \left(\frac{MinFreq}{low\_fr} \right)^{low\_fr} \right]^N .$$

As for the entire NB,

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

$$|NB| \left[\left(\frac{1 - MinFreq}{1 - low\_fr} \right)^{1-low\_fr} \left(\frac{MinFreq}{low\_fr} \right)^{low\_fr} \right]^N .$$

Because calculating the negative border is in itself a costly process, we choose to relax this bound by substituting $|I||\mathcal{F}_{low\_fr}[S]|$ 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 4.1 (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. □

4.2. 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[3] 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 sample, the partition from which it will be sampled. Then, knowing exactly how many transactions should be sampled from each partition, we randomly choose that number of transactions. 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. Because local sample sizes are selected randomly, one of these local samples may be small. Small samples are, by definition, noisier than large ones. Because the performance of DDM depends on Pr_{above} and hence on the noisiness of the data, a sample that 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 $\frac{|S|}{n}$. Because we choose the partitions independently, we can apply the Chernoff bound to the size of the sample from a specific partition,

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

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

Taking $\epsilon = 10\%$, we get $Pr\left(|S^i| \leq 0.9\frac{|S|}{n}\right) \leq e^{-\frac{|S|}{200n}}$. 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.

Because 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.

5. 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 and FDM 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 last one examined the number of redundant itemsets D-sampling generates and compared it with FDM, which generates no redundant candidates.

We ran our experiments on two clusters: the first cluster, which was used for the first, second and fourth sets of experiments, consisted of 15 Pentium computers with dual 1.7-GHz processors. Each of the computers had at least 1 gigabyte of main memory. The computers were connected via an Ethernet-100 network. The second cluster, which we used for the scale-up experiments, was composed of 32 Pentium computers with a dual 500-MHz processor. Each computer had 256 megabytes of memory. The second cluster was also connected via an Ethernet-100 network.

All of the experiments were performed with synthetic databases produced by the standard gen tool (Srikant 1993). The databases were built with the same parameters that were used by Toivonen in Toivonen (1996). 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 600 M 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 *TID* into the *TID%n* partition.

5.1. Speed-up results

The speed-up 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 Fig. 3 shows, the basic speed-up 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 $n = 1$.

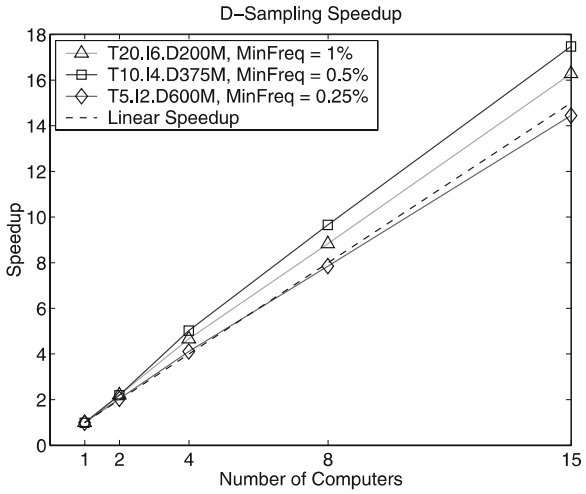


Fig. 3. D-sampling speed-up

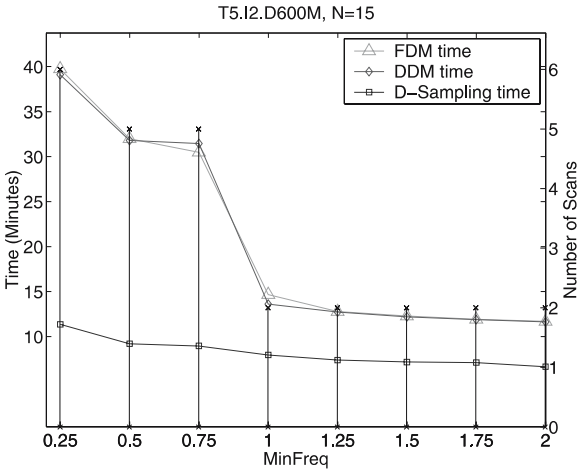


Fig. 4. Runtime of D-sampling, DDM, and FDM for varying MinFreq

5.2. Dependency on *MinFreq*

The second set of experiments (Fig. 4) investigates D-sampling's performance dependency on *MinFreq*, which determines the number and size of the candidates. We compared the D-sampling runtime with that of both DDM and FDM. D-sampling turned out to be insensitive to the reduction in *MinFreq*; its runtime increased by no more than 50% across the whole range. On the other hand, the runtime of DDM and FDM increased rapidly as *MinFreq* 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 levelwise D-ARM algorithm, just as sampling is superior to all levelwise ARM algorithms.

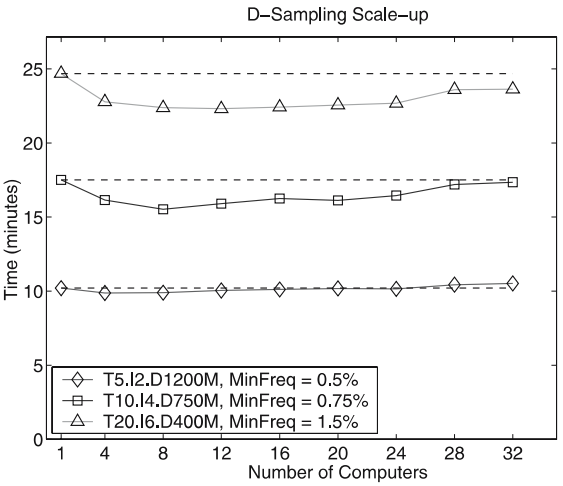


Fig. 5. D-sampling scale-up

5.3. Scale-up

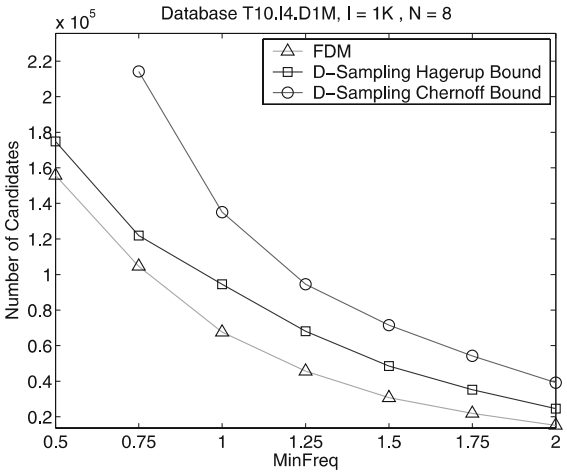
The third set of tests was aimed at testing the scalability of D-sampling. Here the partition size was fixed. We used a database of about 1.5 gigabytes on each computer. A scalable algorithm should have the same runtime regardless of the number of computers.

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 Fig. 5, D-sampling is scalable in two of the tests. In fact, for midrange numbers of computers, D-sampling runs even faster than with $n = 1$ due to the superlinear speed-up discussed earlier. The mild slowdown seen in Fig. 5c is due to the smaller average pattern size and the smaller number of candidates in T5.I2.D1200M. 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.D1200M.

5.4. Number of candidates

Because the main disadvantage of the sequential ing algorithm is the large number of candidates it generates, our last set of experiments was aimed at testing how many of the candidates 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 80 K transactions and maximum error probability $\delta = 0.001$.

Figure 6 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. It can be seen that the number of candidates in D-sampling is strongly tied to the bound the algorithm used for calculating the probability of error. 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



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

Fig. 6. The number of candidates produced by FDM and D-sampling using the Chernoff bound (as suggested by Toivonen) and D-sampling using the Hagerup bound for various databases and when using eight computers

tighter and produces significantly fewer candidates. The table 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 candidate overhead when compared with the number of candidates generated by FDM.

6. Conclusions and future research

We presented a new D-ARM algorithm that uses the communication efficiency of the DDM algorithm to parallelize the single-scan sampling algorithm. Experiments prove that the new algorithm has superlinear speed-up and outperforms FDM with any *MinFreq* value. The exact improvement in relation to FDM or DDM depends on the number of database scans they require. Experiments demonstrate good scalability, provided the database scan is the major bottleneck of the algorithm.

Some open questions still remain. First, it would be interesting to continue partitioning the database until every partition becomes memory resident. This approach may lead to a D-ARM algorithm that mines a database by loading it into the memory of a large number of computers and then runs with no disk-I/O at all. Second, it would be interesting to have a parallelized version of the other single-scan ARM

algorithm—DIC—on a share-nothing cluster, or of the two-scans partition algorithm. Finally, we feel that the full potential of the M-Max algorithm has not yet been realized; we intend to research additional applications for this algorithm.

Acknowledgements. We thank Intel (Israel) and the Israeli ministry of defence (Mafaat) for their generous support of this research.

References

Agrawal R, Imielinski T, Swami AN (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC, pp 207–216

Agrawal R, Shafer J (1996) Parallel mining of association rules. IEEE Trans Knowl Data Eng 8:962–969

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th international conference on very large databases (VLDB'94), Santiago, Chile, pp 487–499

Ananthanarayana VS, Subramanian DK, Murty MN (2000) Scalable, distributed and dynamic mining of association rules. In: Proceedings of HiPC'00, Bangalore, India, pp 559–566

Brin S, Motwani R, Ullman J, Tsur S (1997) Dynamic itemset counting and implication rules for market basket data. SIGMOD Rec 6:255–264

Cheung D, Han J, Ng V, Fu A, Fu Y (1996) A fast distributed algorithm for mining association rules. In: Proceedings of the 1996 international conference on parallel and distributed information systems, Miami Beach, Florida, pp 31–44

Cheung D, Xiao Y (1998) Effect of data skewness in parallel mining of association rules. In: 12th Pacific-Asia conference on knowledge discovery and data mining, Melbourne, Australia, pp 48–60

Hagerup T, Rub C (1989/90) A guided tour of Chernoff bounds. Inf Process Lett 33:305–308

Han E-HS, Karypis G, Kumar V (2000) Scalable parallel data mining for association rules. IEEE Trans Knowl Data Eng 12:352–377

Han J, Fu Y (1995) Discovery of multiple-level association rules from large databases. In: Proceedings of the 21st international conference on very large data bases (VLDB'95), Zurich, Switzerland, pp 420–431

Han J, Pei J, Yin Y (1999) Mining frequent patterns without candidate generation. Technical Report 99-12, Simon Fraser University

Iko P, Kitsuregawa M (2003) Parallel fp-growth on PC cluster. In: Seventh Pacific-Asia conference of knowledge discovery and data mining (PAKDD03)

Jarai Z, Virmani A, Iftode L (1998) Towards a cost-effective parallel data mining approach. Orlando, Florida

Lin D-I, Kedem ZM (1998) Pincer search: a new algorithm for discovering the maximum frequent set. In: Extending database technology, pp 105–119

Park JS, Chen M-S, Yu PS (1995a) An effective hash-based algorithm for mining association rules. In: Proceedings of ACM SIGMOD international conference on management of data, San Jose, CA, pp 175–186

Park JS, Chen M-S, Yu PS (1995b) Efficient parallel data mining for association rules. In: Proceedings of the ACM international conference on information and knowledge management, Baltimore, MD, pp 31–36

Pei J, Han J (2000) Can we push more constraints into frequent pattern mining? In: Proceedings of the ACM SIGKDD conference on knowledge discovery and data mining, Boston, MA, pp 350–354

Savasere A, Omiecinski E, Navathe SB (1995) An efficient algorithm for mining association rules in large databases. In: Proceedings of the 21st international conference on very large databases (VLDB'95), pp 432–444

Schuster A, Wolff R (2001) Communication-efficient distributed mining of association rules. In: Proceedings of the 2001 ACM SIGMOD international conference on management of data, Santa Barbara, CA, pp 473–484

Srikant R (1993) Synthetic data generation code for association and sequential patterns. Available from the IBM Quest web site at http://www.almaden.ibm.com/cs/quest/

Srikant R, Agrawal R (1994) Mining generalized association rules. In: Proceedings of the 20th international conference on very large databases (VLDB'94), Santiago, Chile, pp 407–419

Srikant R, Agrawal R (1996) Mining quantitative association rules in large relational tables. In: Jagadish HV, Mumick IS (eds) Proceedings of the 1996 ACM SIGMOD international conference on management of data, Montreal, Quebec, Canada, pp 1–12

Srikant R, Vu Q, Agrawal R (1997) Mining association rules with item constraints. In: Heckerman D, Mannila H, Pregibon D, Uthurusamy R (eds) Proceedings of the ACM SIGKDD conference on knowledge discovery and data mining. AAAI Press, pp 67–73

Thomas S, Chakravarthy S (2000) Incremental mining of constrained associations. In: Proceedings of HiPC'00, Bangalore, India, pp 547–558
Toivonen H (1996) Sampling large databases for association rules. In: Proceedings of the 22nd international conference on very large databases (VLDB'96), pp 134–145
Zaiane OR, El-Hajj M, Lu P (2001) Fast parallel association rules mining without candidacy generation. In: IEEE 2001 international conference on data mining (ICDM'2001), pp 665–668
Zaki MJ, Ogihara M, Parthasarathy S, Li W (1996) Parallel data mining for association rules on shared-memory multi-processors. In: Proceedings of the Supercomputing'96, Pittsburg, PA, pp 17–22
Zaki MJ, Parthasarathy S, Ogihara M, Li W (1997a) New algorithms for fast discovery of association rules. Technical Report TR651, Rensselaer Polytechnic Institute
Zaki MJ, Parthasarathy S, Ogihara M, Li W (1997b) Parallel algorithms for discovery of association rules. Data Min Knowl Discov 1:343–373

Author biographies

Professor Assaf Schuster (http://www.cs.technion.ac.il/~assaf) received his B.Sc., M.Sc. and Ph.D. degrees in mathematics and computer science from the Hebrew University of Jerusalem. Since being awarded his Ph.D. degree in 1991, he has been with the Computer Science Department at the Technion—The Israel Institute of Technology. His interests include all aspects of parallel and distributed computing.

Ran Wolff (http://www.cs.technion.ac.il/~ranw) received his B.A. in computer science from the Technion—Israel Institute of Technology—and is currently studying toward a Ph.D. in computer science at that same institute. Ran has expertise in large-scale and high-performance data mining. He has authored several papers on data mining in Grid and other distributed environments, including publications in SIGMOD, ICDM and CCGRID.

Dan Trock received his B.S. in physics and M.Sc. in electrical engineering from the Technion—Israel Institute of Technology. He is currently a senior networking and communications systems design engineer at Motorola Semiconductor (Israel).

Correspondence and offprint requests to: Ran Wolff, Department of Computer Science, Technion—Israel Institute of Technology, Haifa, 32000, Israel. Email: ranw@cs.technion.ac.il