Iranian Journal of Science & Technology, Transaction B: Engineering, Vol. 33, No. B6, pp 511-526
Printed in The Islamic Republic of Iran, 2009

DISTRIBUTED ASSOCIATION RULES MINING USING NON-DERIVABLE FREQUENT PATTERNS*

M. DEYPIR AND M. H. SADREDDINI**

Dept. of Computer Science and Engineering, Faculty of Engineering, Shiraz, I. R. of Iran
Email: sadredin@shirazu.ac.ir

Abstract– Mining association rules in distributed databases is an interesting problem in the context of parallel and distributed data mining. A number of approaches have, so far, been proposed for distributed mining of association rules. However, most of them consider all types of frequent itemsets the same, even though there are different types of itemsets in distributed databases, e.g., derivable and non-derivable. In this study, a new application of deduction rules is introduced for distributed mining of association rules which exploits the derivability of itemsets to reduce communication overhead and to enhance response time. A new algorithm is proposed which mines derivable and non-derivable frequent itemsets in a distributed database. Since the collection of derivable and non-derivable frequent itemsets form all frequent itemsets, our algorithm mines all frequent itemsets rather than a subset of them. In the algorithm, there is no need to scan local databases and exchange messages in order to obtain support counts of derivable frequent itemsets, since each site can produce them autonomously. Experimental evaluations on horizontally partitioned real-life datasets show that such exploitation drastically reduces communication and also improves response time. Therefore the new algorithm is useful when communication bandwidth is the main bottleneck.

Keywords– Distributed data mining, Association rules mining, Non-derivable frequent itemsets, distributed deduction rules

1. INTRODUCTION

Association rule mining (ARM) in large transactional databases is a central problem in the field of knowledge discovery and data mining, and has a wide application area such as market basket analysis, document clustering, web management, and profiling high frequency accident locations. The input of ARM is a database in which objects are grouped together in each transaction. ARM then requires us to find sets of objects which tend to associate with one another. Given two distinct sets of objects, X and Y, we say Y is associated with X if the appearance of X usually implies Y, we then say that rule $X \Rightarrow Y$ is confident in the database. X and Y are also called *itemsets* (set of items) or patterns. We would not usually be interested in an association rule unless it appears in more than a certain fraction of the context; if it does, we say that the rule is frequent. The thresholds of frequency (*minimum support*) and confidence (*minimum confidence*) are parameters of the problem and are usually determined by the user according to his or her needs.

The problem of mining association rules consists of two major steps, finding frequent itemsets from a database and generating rules based on found frequent itemsets. The first stage of this mining task is the most time consuming. This is because generating association rules based on the found frequent pattern is straightforward. Therefore, the problem of mining association rules is reduced to finding frequent

*Received by the editors July 23, 2008; Accepted July 21, 2009.
**Corresponding author

itemsets. The problem of mining association rules in large databases is well studied and numerous algorithms have been proposed [1, 24, 17, 3, 9, 20, 5, 7, 6]. When data is saved in a distributed database, a distributed data mining algorithm is needed to mine association rules. In distributed data mining it is impossible to move raw data to a central site and then use a centralized data mining algorithm. This is due to the security of the data, the privacy of each site and high communication costs. Therefore, mining association rules in this environment is a distributed problem and must be performed using a distributed algorithm that doesn't need raw data exchange between participating sites. Distributed Association Rules Mining, also called DARM, has been addressed by some researchers and a number of distributed algorithms have been proposed [2, 15, 22, 23, 4]. Run time and communication are two main factors that are considered in DARM. Performance of a DARM method as a distributed system can be evaluated using models presented in [19].

While all of the proposed DARM algorithms treat all types of itemsets the same, there are different types of itemsets. As mentioned in [12], in each dataset, itemsets are divided into two major groups, derivable and non-derivable. In a centralized database we can deduce support of derivable itemsets without a database scan. In this study, the idea of derivable and non-derivable frequent itemsets is extended in a distributed case where all frequent itemsets are found efficiently by the direct mining of non-derivable frequent itemsets, and indirect mining of derivable itemsets. A distributed algorithm called DDN, which uses such a strategy to mine all frequent itemsets is proposed here. Experimental evaluations of DDN on horizontally partitioned real-life datasets show the superiority of our approach in comparison with the previously proposed DARM algorithm.

The remainder of the paper is as follows. In the following section some related works are reviewed. In Section 3, concept of non-derivable and derivable itemsets is reviewed and the problem of mining all non-derivable frequent itemsets is extended to the distributed setting. Section 4 exploits derivability to mine all frequent itemsets and proposes the DDN algorithm. Some experimental evaluations showing the superiority of our approach are presented in Section 5. Finally, Section 6 concludes the paper and Section 7 mentions some future works.

2. RELATED WORKS

To mine all association rules in distributed databases, a number of algorithms have been proposed in the literature. The first algorithm for the DARM problem is the Count Distribution (CD) algorithm, first proposed for parallel mining of association rules in the share nothing parallel systems [2]. It is a parallel version of the Apriori algorithm and assumes that data sets are horizontally partitioned among different sites. In each iteration of the CD, each site has an identical set of candidates. After counting their local support against the local database, the algorithm obtains a global count by exchanging local counts among participating sites. Cheung et al. proposed the Fast Distributed algorithm (FDM) to mine rules from distributed data sets partitioned among different sites [15]. In each site, FDM finds the local support counts and prunes all infrequent local candidate sets. After completing local pruning, each site broadcasts messages containing all the remaining candidate sets to all other sites to request their support counts. It then decides whether large itemsets are globally frequent and generates the candidate itemsets from the globally frequent itemsets. This process continues until no globally frequent itemset is generated or no candidate set is produced. FDM's main advantage over CD is its reduction of communication overhead. In order to compare our algorithms, we have implemented the FDM algorithm as one which mines all frequent itemsets in a distributed setting directly. The FDM algorithm is also a base algorithm for the recently proposed ODAM algorithm [4]. ODAM has added greater message optimization and performance enhancement to the FDM. Assaf Schuster and his colleagues proposed the Distributed Decision Miner

(DDM) [22]. It generates only those rules that have confidence above the threshold level without generating a rule's exact confidence, therefore considering all rules above the confidence threshold as being the same. Another algorithm proposed by Assaf et al. is Distributed Sampling [23] (D-Sampling), an extension of the sequential sampling algorithm [24] in distributed databases.

In transactional databases there are different kinds of itemsets. A number of researches have been proposed which classify itemsets in two groups from different points of view. The most popular of these are Closed itemsets [21], Free sets [8] or Generators [18], Disjunction-free sets [10] or Disjunction-free generators [18], which are extensions of free sets, Non-Derivable Itemsets [12] and the unified framework presented in [13]. Calders et al. have surveyed the core concepts used in the recent works on concise representation for frequent sets [14]. All the above categorization of itemsets are aimed at dividing the frequent itemsets into two groups, thus providing a concise representation. This concise representation is a compact form of all the frequent itemsets. For example in [12], itemsets are classified to derivable and non-derivable. Experimental evaluations show that non-derivable frequent itemsets are one of the most successful classifications of frequent itemsets since a large number of frequent itemsets could be deduced without database scans [11]. If support of an itemset can be obtained without a database scan by using deduction rules, the itemset is derivable, if not it is non-derivable. In this study the derivability concept is referred to as the classification of an itemset in two groups, i.e., non-derivable and derivable. A new DARM algorithm is proposed here which benefits from the derivability of an itemset in a distributed environment. The following section reviews the problem of mining non-derivable frequent itemsets and extends the concept of derivability in distributed setting.

3. MINING ALL NON-DERIVABLE FREQUENT ITEMSETS IN DISTRIBUTED DATABASES

In this section, derivable and non-derivable itemsets are briefly reviewed. After that, we propose the concept of non-derivable frequent pattern mining in distributed environments and consequently derive distributed deduction rules. In [12], rules were given to derive bounds on the support of an itemset I if the supports of all strict subsets of I are known. For any subset $J \subseteq I$, we obtain lower and upper bounds on the support of I using the following formulas, respectively [11].

$$\text{If } |I \setminus J| \text{ is odd, then} \qquad I.\text{supp} \leq \sum_{J \subseteq X \subset I} (-1)^{|I \setminus X|+1} X.\text{supp} \tag{1}$$

$$\text{If } |I \setminus J| \text{ is even, then} \qquad I.\text{supp} \geq \sum_{J \subseteq X \subset I} (-1)^{|I \setminus X|+1} X.\text{supp} \tag{2}$$

where $\perp$.supp is support of the itemset I. The rule involving I and X is referred by $R_X(I)$. It was shown that the above rules deduce tight bounds on the support of a given itemset in the database D. If the lower bound of the itemset is equal to its upper bound, then the itemset is derivable.

When for an itemset I the smallest upper bound (u_I) equals the highest lower bound (l_I), then we have actually obtained the exact support of the set based solely on the support of its subsets. Such a set I will be called Derivable Itemsets (DI), all other itemsets are called Non-Derivable Itemsets (NDIs). Generating all frequent itemsets using non-derivable frequent itemsets is more efficient than mining all frequent itemsets at once. This is due to the fact that the derivable frequent itemsets do not need a database scan. A level-wise Apriori-like algorithm called NDI was given in [12] to compute non-derivable frequent itemsets. In the NDI algorithm, in addition to the monotonicity check of Apriori candidate generation, the lower and upper bounds on the candidate itemsets are computed. Such a check is possible since, in Apriori a set I can only be a candidate after all its strict subsets have been computed. The candidate itemsets that have an upper bound below the minimal support threshold are pruned, because they cannot be frequent.

The actual support count is specified for an itemset that has a lower bound equal to its upper bound. Therefore, for such itemsets, a support value is determined with no database scan. It was shown experimentally that mining all frequent itemsets using non-derivable frequent itemset mining is more efficient than the direct extraction of frequent itemsets if rules up to a suitable depth are used [12]. If we use only rules $R_X(I)$ for $|I\text{-}X| \leq k$, we say that we use rules up to depth k. In practice most pruning is done by the rules of limited depth.

Based on Non-derivable frequent itemsets, it is possible to generate not only all frequent itemsets, but also their support values. The following two corollaries from [12] allow us to generate derivable frequent itemsets. It is noted that Corollaries 1 and 2 are Corollaries 2 and 3 in [15].

Corollary 1. *If I is an NDI, but it turns out that $R_X(I)$ equals the support of I, then all supersets $I \cup \{i\}$ of I will be DI, with rules $R_{X \cup \{i\}}(I)$ and $R_{X \cup \{i\}}(I \cup \{i\})$.*

In other words, if a non-derivable itemset *I* has support equal to the lower or upper bounds, then all its supersets are derivable as well and their support is calculated by extending whichever deduction rule that produces the exact support of *I*.

Corollary 2. *If we know that I is DI, and that rule $R_X(I)$ gives the exact support of I, then $R_{X \cup \{i\}}(I \cup \{i\})$ gives the exact support for $I \cup \{i\}$.*

The above two corollaries help us to produce all frequent itemsets because by using them, it is not necessary to try all possible deduction rules for the above mentioned situations. In this study, non-derivable frequent itemset mining and the above two corollaries are utilized to mine all frequent itemsets in distributed databases where the new algorithm DDN is proposed.

Mining all frequent itemsets in distributed databases has some challenges. First, each site has to scan its local database for all candidates. Second, generating all frequent itemsets imposes excessive communication overhead on the network. These are due to the fact that there is a large amount of fragmented data distributed in participating sites, and especially when the low minimum support threshold is used and/or correlated data are spread in various sites. The aim is to reduce the I/O process and improve communication on the network. In fact, the distributed algorithm needs to scan local databases and to exchange messages not only for NDIs, but also for DIs. So far the proposed DARM algorithms consider the two types of itemsets the same and do not distinguish between them. By considering and utilizing this difference in distributed settings, the performance of the DARM algorithm is enhanced. In our new distributed algorithm, derivable frequent itemsets as a subset of all frequent itemsets, are generated in each site without any local database scan or message transmission. Based on exploiting the derivability concept, in this study a new approach named DDN is proposed to mine all frequent itemsets in horizontally fragmented data sets. In DDN, derivable frequent itemsets are indirectly generated in each site based on non-derivable frequent itemsets. It is shown here that such indirect extraction of frequent itemsets in distributed databases outperforms the direct mining of frequent itemsets. In DDN, the derivability concept is utilized within the FDM algorithm to achieve better performance. The choice of FDM derived from the fact that, in practice, it remains one of the most general algorithms for which other DARM algorithms such as ODAM are based. It is also used in the literature as a benchmark to compare the new DARM algorithm. Indeed, the derivability concept can be exploited in any DARM algorithm.

As noted above in the new DARM algorithm, non-derivable frequent itemsets are mined in a distributed manner while derivable frequent itemsets are mined autonomously by each site. Here, it is shown how non-derivable frequent itemsets in a distributed database are extracted. Consequently, we investigate the application of deduction rules in a distributed environment to mine all frequent itemsets and introduce the concept of distributed deduction rules which are the extensions of ordinary deduction rules.

Let DB be a distributed database with *D* transactions. Suppose that there are *n* sites $S_1, S_2, ..., S_n$ in a distributed database system where $\{DB_1, DB_2, ..., DB_n\}$ are local databases respectively. We are going to mine non-derivable frequent itemsets in the distributed database. Assume that it is possible to move the data of every site into a centralized database system. If we do so, mining all non-derivable frequent itemsets using the NDI algorithm is a straightforward process. However, moving data is costly and in some cases impossible due to the security of the data and the privacy of each site. As a result, a distributed algorithm which performs the mining task without any raw data exchange is proposed. First, deduction rules in the distributed setting are introduced.

Distributed Deduction rules: Two formulas, 1 and 2 are extended here for the distributed case as follows: suppose that for an itemset *I*, $I.\text{supp}_i$ be the support of *I* on site S_i. We obtain the following set of inequalities to derive tight bounds on the support of an itemset *I* in distributed database *D*.

If |I \ J| is odd, then

$$I.\text{supp} = I.\text{supp}_1 + I.\text{supp}_2 + ... + I.\text{supp}_n \leq$$
$$\sum_{J \subseteq X \subseteq I} (-1)^{|I \setminus X|+1} (X.\text{supp}_1 + X.\text{supp}_2 + ... + X.\text{supp}_n) = \sum_{J \subseteq X \subset I} ((-1)^{|I \setminus X|+1} \sum_{1 \leq i \leq n} X.\text{supp}_i).$$

Similarly to the above, if |I \ J| is even, then

$$I.\text{supp} = I.\text{supp}_1 + I.\text{supp}_2 + ... + I.\text{supp}_n \geq$$
$$\sum_{J \subseteq X \subseteq I} (-1)^{|I \setminus X|+1} (X.\text{supp}_1 + X.\text{supp}_2 + ... + X.\text{supp}_n) = \sum_{J \subseteq X \subset I} ((-1)^{|I \setminus X|+1} \sum_{1 \leq i \leq n} X.\text{supp}_i).$$

Consequently, the deduction rules in the distributed database can be summarized as follows:

$$\text{If } |I \setminus J| \text{ is odd, then} \qquad I.\text{supp} \leq \sum_{J \subseteq X \subset I} ((-1)^{|I \setminus X|+1} \sum_{1 \leq i \leq n} X.\text{supp}_i). \tag{3}$$

$$\text{If } |I \setminus J| \text{ is even, then} \qquad I.\text{supp} \geq \sum_{J \subseteq X \subset I} ((-1)^{|I \setminus X|+1} \sum_{1 \leq i \leq n} X.\text{supp}_i). \tag{4}$$

The above two formulas show that, instead of moving the distributed data to a central site in order to determine the derivability of an itemset *I*, we can compute the deduction rules distributively. The new DARM algorithm uses the above deduction rules to decide whether an itemset *I* is derivable or non-derivable in the distributed setting.

As the above two formulas show, each site evaluates deduction rules for an itemset using information that is gathered from all sites about the subsets of that itemset. Since in a DARM algorithm each site has complete information about the global support of strict subsets of *I*, there is no need to actually compute the internal sigma, and in fact, they are actually computed before. The method used to generate all frequent itemsets using all non-derivable frequent itemsets is described in [12]. A similar procedure can be considered in a distributed setting using the distributed deduction rules. The Corollary 1 and 2 are used to efficiently generate all frequent itemsets.

In the following section a new efficient method is proposed to mine all frequent itemsets in a distributed environment utilizing the derivability of the itemsets. Experimental evaluations presented in Section 5 show that this approach considerably reduces communication and improves efficiency.

4. EXPLOITING DERIVABILITY TO MINE ALL FREQUENT ITEMSETS IN DISTRIBUTED DATABASES

In every algorithm used to mine all frequent itemsets in a distributed database, e.g. CD and FDM, at each iteration there is a number of candidate itemsets for which the distributed algorithm has to scan the local

datasets and exchange the local support counts to determine globally frequent itemsets. At each iteration, a potential number of candidate itemsets are derivable. For such candidates there is no need to scan local databases and exchange support counts as each site can determine whether a derivable candidate is frequent or not, using previous information that the site has. In other words, in each iteration we have two types of candidates, non-derivable and derivable. For the former, local databases have to be scanned and the support counts have to be exchanged among sites, but for the latter there is no need for local database scans and support count exchanges. However, after a specified iteration it is probable that there is no non-derivable candidate itemset, and thus each site can continue its operation autonomously without any local database scan, communication and synchronization.

a) Motivating example

The following example shows more clearly the effect of exploiting the derivability concept to mine all frequent patterns in a distributed database.

Example 1: Consider a distributed database having two sites, S_1 and S_2, represented in Fig. 1, in which local transactional databases are D_1 and D_2, respectively. We are going to mine all frequent itemsets in the distributed database by using minimal support threshold 2.

D_1=

TID	Items
1	D,E
2	A,B,C,F,G

D_2=

TID	Items
1	A,B,C,F
2	D,E,H

Fig. 1. A distributed database with two sites

For the sake of simplicity and presentation, in this example the CD algorithm is adopted to mine frequent itemsets, although more efficient DARM algorithms, notably FDM and ODAM, have been proposed in the literature. Indeed, exploiting the derivability of itemsets for distributed pattern mining is independent of the algorithm. In the running example, after the local database scan of D_1 and D_2, and the exchange of the support counts of 1-itemsets between S_1 and S_2, the {A, B, C, D, E, F} frequent itemsets are found.

All of them have support 2, and thus are frequent. Therefore the set of candidate 2 itemsets in each site are {AB, AC, AD, AE, AF, BC, BD, BE, BF, CD, CE, CF, DE, DF, EF}. Before any local database scan, each site checks for derivability of the candidate itemsets by deduction rules evaluation. The evaluation is performed based on global information about the previous iteration that is available at each site. In the running example tight bounds are computed for each candidate itemset, and deduction rule evaluation for candidate itemset AB is as follows:

$$supp(AB) \geq supp(A) + supp(B) - supp(\{\}) = 2 + 2 - 4 = 0 \qquad R_{\{\}}$$
$$supp(AB) \leq supp(A) = 2 \qquad R_{\{A\}}$$
$$supp(AB) \leq supp(B) = 2 \qquad R_{\{B\}}$$
$$supp(AB) \geq 0$$

As a result, interval [0, 2] is tight bound for AB, therefore AB is non-derivable. The similar deduction rules evaluation is applied to all other candidates. The remaining process consisting of local database scans and support exchange between S_1 and S_2 is continued according to the CD algorithm, hence global large 2- itemsets are found at each site after iteration 2. This process results in the following set of

frequent itemsets with 2 as the support value.

{AB, AC, AF, BC, BF, CF, DE}

In the third iteration, the set of candidate 3 itemsets based on Apriori's candidate generation approach are {ABC, ACF, ABF, BCF}. Since both sites have complete information of iteration 1 and 2, each of them computes deduction rules independently. For example deduction rule evaluation for candidate itemset ABC is presented here:

$$supp(ABC) \leq supp(AB) + supp(AC) + supp(BC) - supp(A) - supp(B) - supp(C) + supp(\{\})$$
$$= 2+2+2-2-2-2+4 = 4 \qquad R_{\{\}}$$
$$supp(ABC) \geq supp(AB) + supp(AC) - supp(A) = 2+2-2 = 2 \qquad R_{\{A\}}$$
$$supp(ABC) \geq supp(AB) + supp(BC) - supp(B) = 2+2-2 = 2 \qquad R_{\{B\}}$$
$$supp(ABC) \geq supp(AC) + supp(BC) - supp(C) = 2+2-2 = 2 \qquad R_{\{C\}}$$
$$supp(ABC) \leq supp(AB) = 2 \qquad R_{\{AB\}}$$
$$supp(ABC) \leq supp(AC) = 2 \qquad R_{\{AC\}}$$
$$supp(ABC) \leq supp(BC) = 2 \qquad R_{\{BC\}}$$
$$supp(ABC) \geq 0$$

These sets of inequalities give us equality between the least upper bound and the greatest lower bound. Therefore, each site independently finds that the support value of ABC is 2, which satisfies the minimal support threshold. Consequently, there is no need to scan local databases and exchange support values across the network in order to compute the support of ABC. Other candidate 3–itemsets of the above distributed database are also derivable and frequent. Now, we come up with iteration 4, where the set of candidate 4-itemsets contains only ABCF. Since this set is an extension of derivable frequent itemset ABC as mentioned in [12], it is possible to compute the support of such itemset by using one of the deduction rules that gave the exact support of ABC. For example, we use the above R_A rule and extend it by F as follows:

$$supp(ABCF) = supp(ABF) + supp(ACF) - supp(AF) = 2+2-2 = 2 \qquad R_{AF}$$

Therefore, it is also a frequent itemset. To summarize, the set of frequent itemsets which are found directly are

{A, B, C, D, E, F, AB, AC, AF, BC, BF, CF, DE}, and other frequent itemsets {ABC, ACF, ABF, BCF, ABCF} are found indirectly by utilizing the derivability concept. If the above example was solved by CD without using the derivability concept, we would have additional communication and I/O costs in iteration 3 and 4, where the candidate sets contain derivable itemsets.

Consequently, in a partitioned database there is no need to scan local databases and exchange support counts across the network in order to find global support counts of derivable itemsets. Since in distributed databases there are usually many derivable itemsets (quantity of such a derivable depends on the nature of data), we can save communication bandwidth and also achieve less response time by utilizing the derivability of itemsets. However, if we are going to find all frequent itemsets based on the non-derivable frequent itemsets, the amount of performance that is achieved depends on the nature of the distributed data. The distributed data mining algorithm can achieve better performance when there are more derivable frequent itemsets in the data set.

b) The DDN algorithm

In this section a distributed algorithm called DDN is proposed for efficient mining of all frequent itemsets. The DDN is an acronym for distributed mining of Derivable and Non-derivable frequent itemset

mining. Since the collection of derivable and non-derivable frequent itemsets form all frequent itemsets, the DDN mines all frequent itemsets and not a subset of them. At each iteration of the distributed DDN, there is a large number of candidate itemsets for which each site deduces their support and identifies whether they are frequent or infrequent. After some iteration, all candidates are potentially derivable, and each site continues to independently compute support of the candidates. This trend continues until there is no candidate itemset to generate and evaluate.

In Example 2, from iteration 3 to the end all candidate itemsets are derivable. Although in the above example all deduction rules are evaluated, in practice, for most of the itemsets, it is enough to use only rules up to a limited depth, e.g., 3 or 4 [12]. Experimental evaluation in [12] shows that generating all frequent itemsets based on non-derivable are worthwhile in terms of run time when rules up to a specified depth are used. The reason for this is that the evaluation of all deduction rules for long pattern takes considerable time. The problem of determining the depth depends on the size of the database. For larger databases it is better to use a smaller depth. In the implementation of DDN, rules up to depth 3 are evaluated because the datasets which are used in our experiments are large enough to use this depth. In fact, depth of rules is a simple parameter of the program and can be set to a suitable number.

DDN utilizes the message optimization that was proposed in the FDM algorithm. The FDM's message optimization is based on the following important relationship between large itemsets and the sites in a distributed database: Every globally large itemset must be locally large at some site(s).

This relationship can significantly enhance communication among sites. Each site is responsible for gathering local support counts of candidates that are locally large at that site. Since a candidate itemset can be locally large at more than one site, to avoid doing the same work in two or more sites, for each candidate a special type of polling site is determined based on a hash function on the candidate. Since this hash function is identical in all sites, the unique polling site is determined by all sites on which the candidate is found locally frequent. Each polling site is responsible for collecting support counts of a specified set of candidates which are locally large at one or more site(s). The DDN benefits both from NDI [12] and FDM [15]. The DDN algorithm is depicted in Figure 2. For the ease of presentation the level of abstraction is heighten by using multiple procedures. For the completeness of the explanation, the entire algorithm including message optimization of FDM is illustrated. For complete information about the FDM algorithm interested readers are referred to [15].

In each iteration k > 1, non-derivable candidate itemsets ($CG_{(k)}$) are determined by the *candidateGen* procedure which receives globally large non-derivable frequent itemsets of a previous iteration ($NDG_{i(k-1)}$) as input. In addition to generating non-derivable candidate itemsets, *candidateGen* computes derivable large itemsets of the iteration *(*$DL_{(k)}$). In each iteration, computation of derivable frequent itemsets do not need a database scan or message exchange. These itemsets are the same in every site. The *derivableFrequents* function computes the remainder of the derivable frequent itemsets which resulted from derivable and non-derivable frequent itemsets of a previous iteration. This procedure inserts its result to $DL_{(k)}$.

In Step 7, the local support of candidate itemsets are determined. If there are candidate non-derivable frequent itemsets, then message exchange between sites based on the FDM message optimization is started. The *get_localFrequents* function computes locally large frequent non-derivable itemsets, and then the polling site of each local large candidate is determined by *determine_pollingSites*. In Step 11 candidates are sent to a corresponding polling site.

In Step 12 through 14 each site receives its assigned non-derivable locally large candidate itemsets and stores all of them in the $LP_{i(k)}$ variable. If an itemset is already inserted in the variable, then just the support value is updated in *insertorUpdate* function. After that, in Step 15 the polling requests of candidates are sent. Polling requests are sent only for those itemsets for which the site does not have their support value.

In Step 16 each site sends a reply back to the polling requests of the other sites by using information stored in $T_{i(k)}$. Having received polling responses and updated corresponding information in the $LP_{i(k)}$ variable, at Step 18 each site determines global non-derivable frequent itemsets of its assigned candidates. In Steps 19 and 20 each site broadcasts its gained global non-derivable frequent itemsets to the others and receives theirs as well.

In Step 21 every site has all global non-derivable k-itemsets. Finally, all large k-itemsets are computed by the union of derivable and non-derivable large itemsets of the iteration in Step 22 and are returned in Step 23. At iteration 1 of the DDN, i.e., k=1, only the frequency of singletons are determined, and since all of them are non-derivable candidates, Steps 3 through 7 are bypassed.

Input: $DB_i(i=1,\ldots,n)$: the database partition at each site S_i.
Output: The set of all globally large itemsets L.
Method: Iteratively execute the following program fragment (for the *k*-th iteration) distributively at each site S_i. The algorithm terminates when $L_{(k)} = \varnothing$.

(1) if k = 1 then
(2) $T_{i(1)} = get\_local\_count(D_i,0,1)$
(3) else{
(4) $candidateGen(NDG_{i(k-1)}, CG_{(k)}, DL_{(k)})$
(5) if $DL_{(k-1)} \neq \varnothing$ *then*
(6) $derivableFrequents(DL_{(k-1)}, NDL_{(k-1)}, DL_{(k)})$
(7) $T_{i(k)} = get\_local\_count(D_i,CG_{(k)},i)\}$
(8) if $CG_{(k)}..size > 0$ *then{*
(9) $LL_{i(k)} = get\_localFrequents(D_i,CG_{(k)},i)$
(10) $LL_{i,j(k)} = determine\_pollingSites(LL_{i(k)})$
(11) for j=1 to n send $LL_{i,j(k)}$ *to site* S_j
(12) for j=1 to n {
(13) recive $LL_{j,i(k)}$
(14) $insertorUpdate(LL_{j,i(k)}, LP_{i(k)})\}$
(15) $send\_pollingRequets(LP_{i(k)})$
(16) $reply\_polling\_Requests(T_{i(k)})$
(17) $res = recive\_pollingResponsesAndUpdate(LP_{i(k)})$
(18) $G_{i(k)} = get\_globalndFrequents(res)$
(19) broadcast $G_{i(k)}$;
(20) receive $G_{j(k)}$ *from all other sites* S_j $(j \neq i)$
(21) $NDL_{(k)} = \bigcup_{i=1}^{n} G_{i(k)}$, $(i = 1,\ldots, n)\}$
(22) $L_{(k)} = NDL_{(k)} \cup DL_{(k)}$.
(23) return $L_{(k)}$

Fig. 2. The DDN algorithm

As shown in Fig. 2, DDN consists of many subprograms. Two important subprograms are *candidateGen* and *derivableFrequents*. These subprograms are shown in Figs. 3 and 4, respectively. The former generates non-derivable candidates and frequent derivable itemsets, while the latter produces the remaining derivable frequent itemsets. The derivable frequent itemsets generated by the *candidateGen* are those that are identified by computing deduction rules up to depth 3, but the derivable frequent itemsets produced by the *derivableFrequents* are those that are computed using corollary 1 and 2.

As shown in Fig. 3, in line 1 through 7 of *candidateGen* the subset of NDIs of the previous iteration which have the support equal to the lower or upper bound are identified and are inserted in the $DL_{(k-1)}$ variable. This is due to the fact that as noted in corollary 1, supersets of such itemsets are derivable and must be pruned from non-derivable frequent itemsets. The *derivableFrequents* function uses these itemsets in addition to derivable frequent itemsets to produce derivable frequent itemsets.

In line 8 and 9 candidates are provided in a similar fashion to the FDM. In line 10 through 15 bounds

are determined on the support of the candidates. Non-derivable frequent itemsets are inserted to the list of candidates, i.e., $CG_{(k)}$ and derivable frequent itemsets are added to the list of frequent derivable itemsets, i.e, $DL_{(k)}$.

In Figure 4 the procedure for generating the remainder of frequent derivable itemsets is shown. As mentioned above, this type of frequent itemsets are those which are generated using corollary 1 and 2. For the sake of simplicity in the implementation, $R_{X\cup\{i\}}(I\cup\{i\})$ is used to compute the support. This is common in the two types of derivable frequent itemsets mentioned in corollary 1 and 2.

Procedure candidateGen($NDL_{(k-1)}$, $CG_{(k)}$,$DL_{(k)}$, $DL_{(k-1)}$)
(1) *for all* $X \in NDG_{i(k-1)}$ *do*
(2) *if X.sup = X.l then*
(3) *X.sr =X.lr*
(4) *prune X from* $NDL_{(k-1)}$ *and insert it into* $DL_{(k-1)}$ *;*
(5) *If X.sup= X.u then*
(6) *X.sr = X.u*
(7) *prune X from* $NDL_{(k-1)}$ *and insert it into* $DL_{(k-1)}$ *;*
(8) *divide* $NDL_{(k-1)}$ *into* $NDGL_{i(k-1)}$*, (i=1,…,n);*
(9) $preCG_{(k)} = \cup_{i=1}^{n} CG_{i(k)} = \cup_{i=1}^{n} aprioriGen(NDGL_{i(k-1)})$;
(10) *for all Y in* $PreCG_{(k)}$ *do*
(11) *compute bounds [l,u] on support of Y ;*
(12) *if* $l \neq u$ *then*
(13) *Y.l = l ; Y.u = u; insert Y into* $CG_{(k)}$ *;*
(14) *else*
(15) *if* $l \geq minSup$ *insert Y into* $DL_{(k)}$ *; Y.sr =l // can be Y.sr = u*
end Procedure

Fig. 3. Sub procedure candidateGen of DDN algorithm

The first step in the *derivableFrequents* is generating candidate derivable itemsets $DC_{(k)}$. The *aprioriGen2* joins derivable frequent itemsets of the previous iteration with each other and with remaining non-derivable frequent itemsets of the previous iteration. In this way all extensions of derivable frequent itemsets are generated. As noted in corollary 1 and 2, such extensions are appropriate, thus in the *derivableFrequents*, all extensions are first generated, then their support is computed using the rule number included in the candidate itemset property. The rule number is included during candidate generation in the *aprioriGen2*. At the end of the *derivableFrequents*, the remainder of derivable frequent itemsets are determined and are inserted in the list of derivable frequent itemsets ($DL_{(k)}$). In DDN, frequent non-derivable itemsets and derivable frequent itemsets together form all of the frequent itemsets.

Procedure derivableFrequents($DL_{(k-1)}$, $NDL_{(k-1)}$, $DL_{(k)}$)
$DC_{(k)} = aprioriGen2(DL_{(k-1)}, NDL_{(k-1)})$
for all Y in $DC_{(k)}$ *do*
compute support s of Y;
if $s \geq minSup$ *then*
insert Y into $DL_{(k)}$
end Procedure

Fig. 4. Sub procedure derivable frequents of DDN algorithm

It is important to mention that in each iteration of the DDN algorithm, the $DL_{(k)}$ list is identical in each site. That is, deduced frequent derivable itemsets in all sites are the same. In the beginning iterations of DDN, there are few derivable frequent itemsets, but as the algorithm continues to the latter iteration, the number of derivable frequent itemsets is increased. This trend continues until there is no non-derivable frequent itemsets. As the number of derivable frequent itemsets increases, the derivability concept delivers greater efficiency to the distributed algorithm. However, the number and length of derivable frequent itemsets are the property of the distributed database.

5. EXPERIMENTAL EVALUATIONS

We have implemented all programs in C++ using Visual C++ 6 compiler. The implementations have been tested on a workstation for which Windows XP is running on every node. This workstation consists of eight 1.2GHz Pentium IV PCs with 256 MB of main memory, interconnected via a 10M/100M hub. Parallel message passing software MPICH 2 is used here. To empirically evaluate the proposed DDN algorithm rather than the FDM, several tests are performed on the datasets summarized in the following table. These real life datasets have different properties that help us to evaluate our new DARM algorithm in different situations. Our aim is to compare the DDN and the FDM in terms of communication and computation. Communication and computation are measured with various numbers of nodes and various minimum support values. In every experiment the original dataset is horizontally divided into a number of fragments, each of which resides on a node.

Table 1. Datasets characteristics

Dataset	Number of transactions	Number of items	Average transaction size
Accident	340 184	572	45
Pumsb	490 46	2 112	74
Connect-4	67 557	130	43

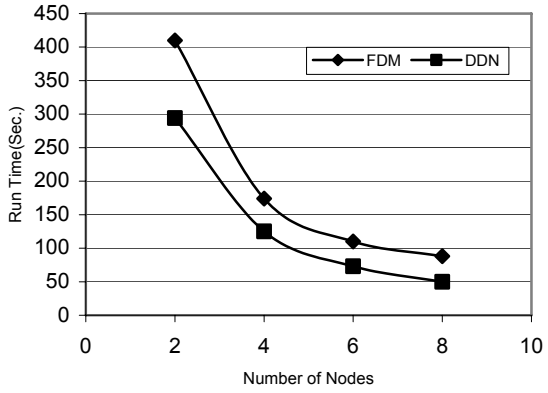
We conduct a set of experiments to show the efficiency of DDN when rules up to depth 3 are used. The following figures show the performance comparison between the DDN and the FDM. The DDN first finds all non-derivable frequent itemsets by using rules up to depth 3 and then mines remaining frequent sets by using them. Using rules up to limited depth 3 makes the approach more efficient, since computing all deduction rules takes more time than evaluating the rules up to only depth 3. In the DDN approach each site finds non-derivable frequent itemsets by negotiation with other sites, while finding the other frequent itemsets (i.e., derivable frequent itemsets) independently. Figures 5 and 6 show the run time and message size of the two algorithms, FDM and DDN, on different data sets with respect to the number of sites when minimum support is fixed.

Both FDM and DDN produce all frequent itemsets. As shown in Figure 5, DDN takes less time than FDM. This difference is due to the fact that the FDM is a direct approach that needs communication and synchronization with other sites to produce all frequent itemsets, while the DDN approach is an indirect approach that needs communication and synchronization only for non-derivable frequent itemsets that are produced by rules up to depth 3.

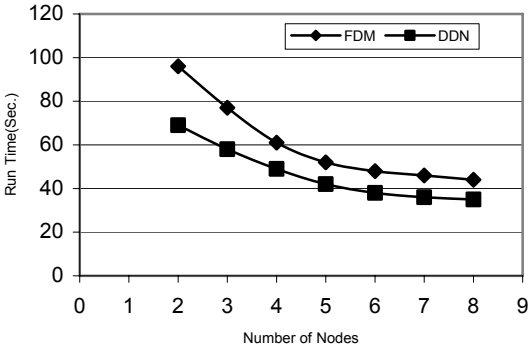
Since the number of this type of non-derivable frequent itemsets can be quite small in dense datasets rather than all frequent itemsets, a large number of frequent itemsets (i.e., remaining frequent itemsets that are derivable) are generated without local database scan, thus as shown in Figure 5, DDN is more efficient than FDM.

Figure 5 also shows that these results are true about the accident, the Pumsb and the connect-4 datasets which have different characteristics. Minimal support is set to 0.7, 0.5 and 0.8 for accident, pumsb and connect-4 respectively. The figure clearly shows that the performance gap between the two approaches is tangible. This experiment shows that performance achievement using the DDN approach rather than the direct approach of FDM is different in various datasets.

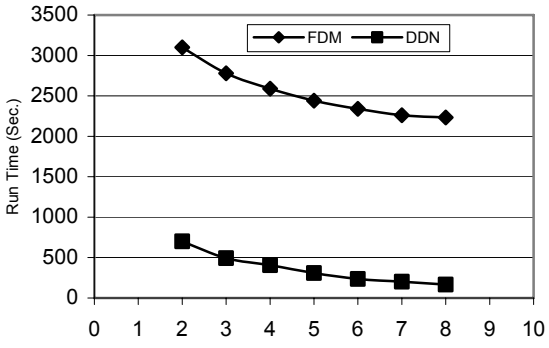
The total size of the message transmitted among participating sites using the two algorithms is also measured experimentally. Here, message size is measured based on the number of communication units which are exchanged among sites.



(a) accident

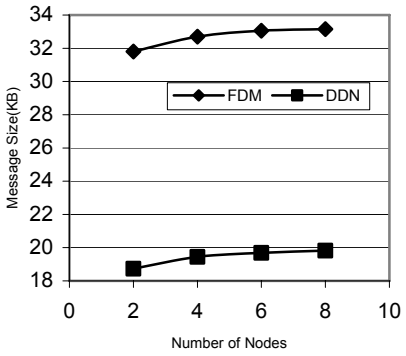


(b) pumsb

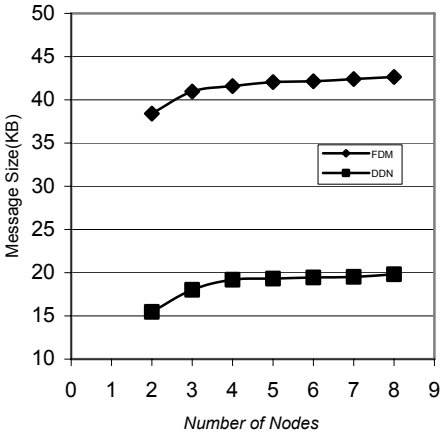


(c) connect-4

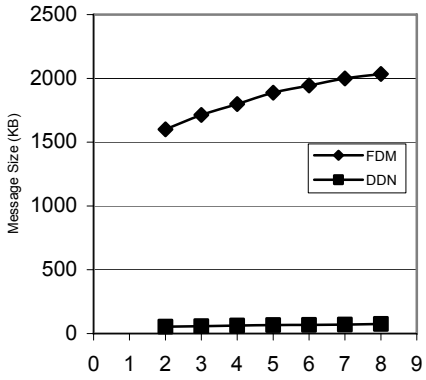
Fig. 5. Run Time comparison of DDN and the FDM algorithms for different number of sites



(a) accident



(b) pumsb



(c) connect-4

Fig. 6. Message size comparison of DDN and the FDM algorithms for different number of sites

Figure 6 shows the amount of communication that is transmitted among sites with respect to the number of sites about different datasets. As can be seen in this figure, total message size transmitted by DDN is significantly less than the FDM algorithm. Since DDN independently obtains the support of derivable frequent itemsets, it does not need to exchange support counts for this type of itemset. This is the reason for the superiority of DDN rather than FDM in terms of communication amount. Figure 6 similarly

shows that the above result is true for accident, pumsb and connect-4 datasets. The figure obviously shows a considerable communication gap between DDN and FDM. In addition, to have better performance on datasets we also need less communication in the DDN.

Figures 5 and 6 also show that the performance achievement of DDN in terms of message size is more than the performance achievement of the algorithm in terms of run time. This is because of the computation overhead of evaluating deduction rules. Although the DDN algorithm does not need a distributed database scan for derivable itemsets, the algorithm needs to evaluate deduction rules up to depth 3 instead. This evaluation imposes considerable computation to the DDN algorithm. In contradiction, DDN expends no extra communication to obtain derivable frequent itemsets. In fact, when in a distributed system the main cost is computation or a network bandwidth limitation, the DDN algorithm is more suitable.

6. CONCLUSION

In this study an efficient approach called DDN to mine all frequent itemsets in the distributed environment is proposed. The DDN exploits the derivability of itemsets for efficient mining of all frequent itemsets in distributed databases. In the DDN, derivability of itemsets is utilized within the well-known FDM algorithm. In fact, other DARM algorithms can benefit from the derivability of itemsets in order to achieve better response time and less communication.

In the DDN, non-derivable frequent itemsets are mined in a distributed fashion and derivable frequent itemsets are mined locally and independently at every site. Therefore, in the DDN communication and I/O the cost of the distributed data mining is reduced significantly. Experimental evaluation on different horizontally partitioned real-life datasets show the superiority of the DDN approach rather than previously proposed method in terms of communication and computation time.

Empirical evaluations also show that communication improvement of the DDN is more than its run time improvement rather than the well-known FDM algorithm. Therefore, our new algorithm is more useful in environments with communication bottleneck and bandwidth limitations.

7. FUTURE WORK

As mentioned in section 4, Calders et al. [11] suggested using deduction rules up to depth 3 or 4 for centralized databases. However, further experiments are required in distributed environments to decide the depth of distributed deduction rules in order to reach a suitable tradeoff between the run time and communication costs.

As some experiments in the previous section show, our new algorithm suffers from the scalability problem. Therefore, for future work we plan to add message optimization, presented on ODAM [4] on DDN, in order to achieve more scalability. On the other hand, Muhonen et al. recently proposed closed non-derivable frequent itemset representation which is more compact than both closed and non-derivable patterns [25]. We intend to use this type of representation to further reduction of communication and enhance run time in the distributed setting.

REFERENCES

1. Agrawal, R., Imiliniski, T. & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proc. Of the ACM SIGMOD Conference on Management of Data, Washington, D.C.*
2. Agrawal, R. & Shafer, J. (1996). Parallel mining of association rules. *IEEE Transaction on Knowledge and Data Engineering*, Vol. 8, No. 6, pp. 962-969.

3. Agrawal, R. & Srikant, R. (1994). Fast algorithms for mining association rules. *Proceedings of the VLDB, Santiago de Chile*, pp. 487-499.
4. Ashrafi, M. Z., Taniar, D. & Smith, K. (2004). ODAM: an optimized distributed association rule mining algorithm. *IEEE distributed systems online*, Vol. 05, No. 3.
5. Bodon, F. (2003). A fast apriori implementation. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*.
6. Bodon, F. (2004). Surprising results of trie-based fim algorithms. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04), Vol. 126 of CEUR Workshop Proceedings*, Brighton, UK,.
7. Borgelt, C. (2003). Efficient implementations of apriori and eclat. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), Vol. 90 of CEUR Workshop Proceedings*, Melbourne, Florida, USA.
8. Boulicaut, J. F. et al. (2000). Approximation of frequency queries by means of free-sets. *Proc. PKDD*, pp. 75-85.
9. Brin, S., Motwani, R., Ullman, J. D. & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Vol. 26, No. 2, of *SIGMOD Record*, pages 255–264. ACM Press.
10. Bykowski, A. & Rigotti, C. A (2001). Condensed representation to find frequent patterns. *Proc. PODS*.
11. Calders, T. (2004). Deducing bounds on the support of itemsets. *Database Technologies for Data Mining-Discovering Knowledge with Inductive Queries, Vol. 2682 of LNCS*, pages 214-233. Springer-Verlag.
12. Calders, T. & Goethals, B. (2002). Mining all non derivable frequent itemsets. *Proc. Principles and Practice of Knowledge Discovery in Databases PKDD'02*, Vol. 2431 of *LNAI*, pp. 74-85, Helsinki, FIN, Springer-Verlag.
13. Calders, T. & Goethals, B. (2003). Mining k-free representation of frequent sets. *Proc. Principles and Practice of Knowledge Discovery in Database PKDD'03*, Vol. 2828 of *LNAI*, pp. 71-82, Cavtat-Dubrovnik, HR, Springer-Verlag.
14. Calders, T., Rigotti, C. & Boulicautet, J. F. (2006). A survey on condensed representation for frequent sets. Constraint-Based Mining; Springer; Vol. 3848.
15. Cheung, D. W. et al. (1996). A fast distributed algorithm for mining association rules. *Proc. Parallel and Distributed Information Systems*, IEEE CS Press, pp. 31-42.
16. Geurts, K., Wets, G., Brijs, T. & Vanhoof, K. (2003). Profiling high frequency accident locations using association rules. *Proc. of the 82nd Annual Transportation Research Board*, p. 18.
17. Han, J., Pie, J., Yin, Y. & Mao, R. (2003). Mining frequent pattern without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*.
18. Kryszkiewicz, M. (2001). Concise representation of frequent patterns based on disjunction free generators. In *Proc. ICDM*, pp. 305-312.
19. Naghibzadeh, M. (1998). Modeling and performance evaluation of distributed system with coordinator. *Iranian Journal of Science and Technology, Transaction B: Engineering*, Vol. 22, No. B3, pp 317-328.
20. Park, J. S., Chen, M. S. & Yu, P. S. (1995). An effective hash based algorithm for mining association rules. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, Vol. 24, No. 2, of SIGMOD Record, pp. 175-186. ACM Press.
21. Pasquier, N. et al. (1999). Discovering frequent closed itemsets for association rules. *Proc. ICDT*, pp. 398-416.
22. Schuster, A. & Wolf, R. (2001). Communication-efficient distributed mining of association rules. *Proc. ACM SIGMOD International Conference on Management of Data*, ACM Press, pp. 473-484.

23. Schuster, A., Wolf, R. & Trock, D. (2005). A high-performance distributed algorithm for mining association rules. *Knowledge And Information Systems (KAIS) Journal*, Vol. 7, No. 4.
24. Toivonen, H. (1996). Sampling large databases for association rules. T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, editors, *Proceedings 22nd International Conference on Very Large Data Bases*, pp. 134–145. Morgan Kaufmann.
25. Muhonen, J. & Toivonen, H. (2006). Closed non-derivable itemsets. *The 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 06)*, 601-608, Berlin, Germany.