A Framework for Mining Functional Dependencies from Large Distributed Databases

Feiyue Ye[1] Jixue Liu[2] Jin Qian[1] Xiaofeng Xue[1]
[1] College of Computer Engineering,
Jiangsu Teachers University of Technology, Changzhou, China
yfy@jstu.edu.cn
[2] School of Computer and Info. Sci., University of South Australia
Jixue.liu@unisa.edu.au

***Abstract*—Discovering functional dependencies (FDs) from existing databases is important to knowledge discovery, machine learning and data quality assessment. A number of algorithms has been proposed in the literature for FD discovery. However these algorithms are designed to work with centralized databases. When they are applied to distributed databases, communication cost of transporting data from different sites makes the algorithms not efficient. In this paper, We analyze the characteristics of mining functional dependencies from large distributed database , and we propose an distributed mining framework for discovery FDs from distribute large databases. We develop a theorem that can prune candidate FDs effectively and extend the partition based approach for distributed databases.**

Keyword: Mining functional dependencies; Functional dependencies; Data mining; Knowledge discovery

I. INTRODUCTION

Functional Dependencies (FDs) capture semantic constraints within data. An FD between two sets of attributes (X, Y) is denoted by $X \rightarrow Y$ and it holds in (also said is satisfied by) a relation r if the values of the latter set Y are functionally determined by the values of the former X [1]. More specifically, for any two tuples t_1 and t_2 in r, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. For example, in Table I(a), $A \rightarrow B$ is satisfied by r_1. With $X \rightarrow Y$, X is called the left hand side while Y is called the right hand side.

Functional dependencies are a key concept in relational theory and the foundation for designing relational databases [2], [3]. They also play critical roles in database management to warrant quality in databases [4] and in reverse engineering to capture data semantics in relational data sources [5]. When FDs are discovered from data of existing databases, they indicate the knowledge embedded in the data and can be used to verify semantic constraints employed in the database design and used to measure data quality. As such, the discovery of functional dependencies from data has been extensively studied [5], [6], [7], [8], [9].

Consequently existing algorithms aim to reduce the complexity of the problem in typical cases. These algorithms include the partition based methods TANE [6]and FD_MINE [9], the negative cover and similar methods, the free-set method and the sampling method. Among these algorithms, the partition method is simple and has relative good efficiency.

The existing methods, including the partition based methods, are designed for centralized databases. When they are applied directly to large distributed databases which are widely in use, the data fragments on different locations (called sites hereafter) have to be transported to one site for checking. The reason for this is that an FD that is checked satisfied on all sites alone may be violated when the data of these sites are put together. Consider Table I where the table $r(A, B)$ has fragments r_1 and r_2 distributed in two sites. The FD $A \rightarrow B$ holds on each of the sites. However, if the data of the two sites is brought together (Part (c)), the FD is violated.

Table I
A DISTRIBUTED DATABASE

(a) r_1 at Site1

TID	A	B
1	$a1$	b1
2	$a1$	b1
3	$a2$	b2

(b) r_2 at Site2

TID	A	B
1	a2	b1
2	a3	b2

(c) $r_1 + r_2$

TID	A	B
1	a1	b1
2	a1	b1
3	a2	b2
4	a2	b1
5	a3	b2

The example shows that checking FDs against data of different sites is not possible if the data is not put together. However, if data is transported together, the communication cost in both time and money adds huge complexity to the already hard problem especially when the size of the distributed database is large.

The time complexity of the FD discovery algorithm depends on the communications speed among the sites, the properties of data distributed, the number of attributes m, the number of tuples N and the degree (in the number of non-local values) of correlation among the attributes in the distributed database. If an existing algorithm is used to mine the FDs in a distributed database, we first need to send the tuples of all sites to the central computer, then FDs are mined by the existing algorithm. In this paper, firstly,

978-0-7695-4225-6/10 $26.00 © 2010 IEEE
DOI 10.1109/AICI.2010.262

we discuss how to reduce the time to check candidate for mining FDs from distributed database, and then, we propose a distributed framework for mining functional dependencies in large distributed databases.

II. How to reduce the time to check candidate of FDs to mine FDs from distributed database

We first observe how the number of tuples affects on the time spending. Tables II, III and IV show the execution time in seconds for mining FDs for different numbers of tuples (the numbers increase downward), different numbers of attributes (the numbers increase rightwards) and different correlated data (c is the percentage of values of a column at a site that also appear in the same column of another site). The experiments are from synthetic data. The results are from literature [10]. The algorithm for mining FD is FUN [7]. We see that the execution time increases along as the tuple numbers and attribute numbers increase. With respect to affection of attribute numbers researcher have put forwards many methods, for pruning methods [6], [9] for the candidate. we do not discuss it in this paper.

With respect to affection of tuple we see that the increment of the execution time is over multiple the increment of the tuple number$\triangle T$. We suppose the increment of the execution time $\triangle t$ is k times the increment of the tuple number , that is, $\triangle t = k\triangle T$ (we see $k > 2$ from Table II , table III and table IV). By this token, it is important to reduce the search space of tuple. so how to reduce the tuble numbers checked is important.

Table II
Execution times in seconds for correlated data($c = 30\%$)

Tuple	10	20	30	40
5000	0.040	0.220	0.761	2.543
20000	0.340	1.462	3.265	7.560
50000	1.161	4.907	11.546	21.390
100000	2.563	11.035	25.606	46.156

Table III
Execution times in seconds for correlated data($c = 50\%$)

Tuple	10	20	30	40
5000	0.080	0.630	2.894	18.586
20000	0.610	2.773	8.241	25.116
50000	2.052	9.032	22.642	53.697
100000	4.736	20.739	50.903	60.050

Table V shows the experiment results from [9] on fifteen datasets from the UCI Machine Learning Repository (2005), where column 3 represents the number of FDs checked, column 4 represents the number of FDs found satisfied [9], column 5 represents the percent of FDs found over the number of FDs checked. We see that the average percent of FDs found over checked is 2.69%. so if the data in each site is checked respectively the candidate numbers checked can be reduced largely so that the tuples checked are reduced.

Table IV
Execution times in seconds for correlated data($c = 70\%$)

Tuple	10	20	30	40
5000	0.130	1.510	8.341	50.352
20000	0.791	4.025	14.440	61.718
50000	2.663	12.077	33.398	96.318
100000	6.879	30.553	73.966	205.357

Table V
The percent of FDs found

Dataset name	# of attribute	# of FDs checked	# of FDs found	% of found /checked
Abalone	8	594	60	10.10
Balance-sale	5	70	1	1.43
Breast-cancer	10	5095	3	0.59
Bridge	13	15397	61	0.40
Cancer-Wisconsin	10	4562	19	0.42
Chess	7	434	1	0.23
Crx	16	47919	494	1.03
Echocardiogram	13	2676	536	20.03
Glass	10	405	27	6.67
Hepatitis	20	1161108	7381	0.64
Import-85	26	2996737	3971	0.13
Iris	5	70	4	5.71
Led	8	477	11	2.3
Nursery	9	2286	1	0.04
Pendigits	17	223143	27501	12.32
Average	11.8	99132	2671	2.69

III. The Distributed mining Framework for Mining Functional Dependencies in Distributed Database

Definition III.1. [6] Let $r(U)$ be a relation over the set of attributes U and $X, Y \subseteq U$. A functional dependency (FD) is a constraint denoted by $X \rightarrow Y$. The FD $X \rightarrow Y$ is satisfied by $r(U)$ if every two tuples t_i, $t_j \in r(U)$, $t_i[X] = t_j[X]$ and $t_i[Y] = t_j[Y]$. In an FD $X \rightarrow Y$, we refer to X as the antecedent and Y as the consequent.

By Armstrong rules, an FD of the form $X \rightarrow AB$ is equivalent to the two FDs $X \rightarrow A$ and $X \rightarrow B$. Consequently we consider FDs with one attribute on the right hand side. Such FDs are also more efficient to discover.

We call an FD with k attributes on the left hand side a *k-attr FD*.

Definition III.2. Let D be a database distributed at n sites. Let D^i denote the data at site $i (i = 1, \cdots, n)$. Then $D = D^1 \cup \cdots \cup D^n$. Let $r(U)$ be a relation in D and $r^i(U)$ be the fragment of the relation at site i. Then $r(U) = r^1(U) \cup \cdots \cup r^n(U)$.

Theorem III.1. *Let $r^1, \cdots, r^n$ be all the fragments of*

relation r. If an FD $X \rightarrow Y$ is satisfied by r, then FD $X \rightarrow Y$ is satisfied by r^i $(i = 1, \cdots, n)$.

Proof We use proof by contradiction. According to definition 1 if there exists a site k such that $r^k(U)$ violates $X \rightarrow Y$, that is, $\exists\, p, q(t_p[X] = t_q[X] \wedge t_p[Y] \neq t_q[Y]$, then because $r^k \subseteq r^g$, then $p, q \in r^g(U)([t_p[X] = t_q[X] \wedge t_p[Y] \neq t_q[Y])$, that is, FD $X \rightarrow Y$ is violated by $r^g(U)$.

Definition III.3. Two tuples t and u are equivalent with respect to a given set X of attributes if $t[X] = u[X]$, called as an equivalent class. A partition of an attribute set X is a set of (disjoint) equivalent classes of X that contain all tuple identifiers of a relation, denoted by π_X. The rank $|\pi_X|$ is the number of equivalence classes in π_X.

Table VI
AN EXAMPLE OF PARTITION

ID	A_1	A_2	A_3
1	a1	b1	c1
2	a1	b1	c2
3	a2	b2	c1
4	a2	b2	c1
5	a2	b3	c2

Theorem III.2. *An FD $X \rightarrow Y$ is satisfied by a relation $r(U)$, if and only if $|\pi_X| = |\pi_{XY}|$[6].*

Example 3. Consider the relation in Table VI. $\pi_{A_1} = \{\{1,2\} : a1, \{3,4\} : a2, \{5\} : a3\}$, $\pi_{A_1 \cup A_2} = \{\{1,2\}, \{3,4\}, \{5\}\}$, then $|\pi_{A_1}| = |\pi_{A_1 \cup A_2}| = 3$, so FD $A_1 \rightarrow A_2$ holds.

Definition III.4. If a FD holds at a site i in a distributed database, then we call the FD a local functional dependencies. If a FD hold at all sites in a distributed database, then we call the FD a global functional dependencies.

To assign FD candidates to each site, we design a special hash function that not only distributes all candidates to each site but also satisfies the requirement of the partition method. That is, it is viable to calculate the partition of m attributes from the partition of $(m-1)$ attributes. Hereby, we design hash function is following.

Assume that there are k possible FD candidates to be checked on all sites. To distribute the calculation load evenly among these sites, the hash function decides which site is the start site for the checking of a candidate. At the same time, this reduces the load of calculating partition from site to site. Let there be n sites in the distributed database system, the candidate XY is consisted of $A_{j1}A_{j2}\cdots A_{jk}$(where $(j1, j2, \cdots, jk) \subseteq (1, 2, \cdots, m)$), we have formula (1). In formula (1), the m is the number of attributes in the distributed database. The n is the number of sites. The int(n/m) converts the quotient of n/m to an integer. The n/m part is balance the load when the number of sites is

$$h(j1\,j2\cdots jk) = \begin{cases} \text{int}(\frac{n}{m} \times j1) \bmod n & when \quad m \leq n \\ j1 \bmod n & when \quad m > n \end{cases} \tag{1}$$

larger than the number of attributes such that all.

Definition III.5. If the hash value of the values for the attribute set $A_{j1}A_{j2}\cdots A_{jm}$ is i, then site i is the *start site* of the attribute set $A_{j1}A_{j2}\cdots A_{jm}$.

Mining functional dependencies from large distributed databases has a number of challenges. Firstly, the data transmitted among the sites must be minimized. Secondly, the computer resources in all the sites should be evenly utilized. Thirdly, the FD mining process has to be efficient. Fourthly, the algorithm should be efficient even if the data distribution is skewed.

We illuminate the framework of FD mining in distributed databases by using Figure 1 containing three distributed sites as follow. The framework assumes that there is a central computer in the application coordinating the computation when necessary and checking global satisfaction after individual sites are checked satisfied. It also assumes that the sites are ordered what enables an FD to be checked from site to site following the order.

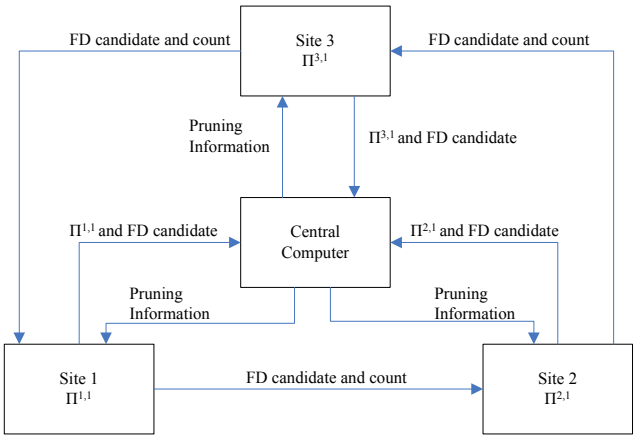

Figure 1. the framework for mining FD from large distributed databases

In the figure, the *FD candidate* denotes an FD that needs to be further checked. the *count* is the number of sites in which the FD has been checked satisfied; $\prod^{1,1}$,$\prod^{2,1}$ and $\prod^{3,1}$ denote the the cluster of single-attribute partition in site 1,2 and 3. The basic procedure for mining functional dependencies is described in the following steps.

Firstly, at each site, the data is partitioned against each attribute by using the partition method in literature [7]. clusters of single-attribute partitions are sent to the central computer.

FD checking starts. At the beginning, all 1-attr FDs are

checked. This is the first round. After the first round is completed, the central computer calculates implied FDs and sends them to all the sites to prune candidate FDs and notifies them to start the second round of checking of 2-attr FDs. This process continues until all possible FDs are checked.

(1) each site generates all candidate 1-attr FDs following the algorithm in *Apriori-Gen* [11] and then picks the FDs of which the site is the *start site* following Formula (1). All the picked FDs are noted by $pFDs$.

(2) at each site, each of $pFDs$ is checked against the fragment of the site. If the checking is successful, the FD is sent to the next site for further checking. If the checking is not successful, the FD does not hold on the global data and the checking of this FD stops permanently.

(3) when a site receives an FD f from its previous site, the site checks the satisfaction of f against its own fragment. If the checking is not successful, the FD does not hold globally. If the checking is successful, there are two cases. The first case is where this site is the final site for f and in this case, f is sent to the central computer for global checking. The second case is where this site is not the final site for f , then f is sent to the next site for further checking.

After all FDs in $pFDs$ are checked, the site sends the central computer a message to indicate the completion. It then waits for the central computer to send the pruned candidates.

(4) the central computer receives an FD f from a site and checks f against the merged partition for single attributes. If the checking is not successful, the FD does not hold globally. Otherwise, the FD holds globally and the FD is stored.

After the central computer receives the completion messages from all the sites and completes checking of all the FDs sent from the sites, the central computer calculates implied FDs from the FDs that hold globally. It then forwards the pruned candidates to each site for them to calculate candidate 2-attr FDs for the second round checking. In the similar manner, n-attr FDs are forwarded to other sites to check (n+1)-attr FDs.

The advantages of the framework are as follows. Firstly, the checking starts from individual sites, the FDs not satisfied at a site can avoid the FDs from being checked in other sites. Secondly, because of the use of the hash function, the checking workload is better distributed among the sites, the computer resource is better utilized.

IV. Analysis

We use hash technology to assign the candidates to corresponding *start site*, thus the candidates which do not hold at *start site* need not checking at other sites, so the search space of tuples is effectively reduced. We suppose that there are P candidates FDs checked and $x\%$ candidate FDs hold at *start site* and they hold at all sites (actually it is quite possible that they hold at less sites, this indicates the number of tuples checked is less than our hypothesis). We again suppose that there are n sites and there are same number of tubles y at each site, then there are $n \times y$ tuples, thus the total number of tuples checked at all sites is $2 \times x\% \times n \times y \times P + P \times y$ if all tuples are processed at a centralized computer using directly existing algorithm, the total tuple number checked is $P \times n \times y$. So when $(2 \times x\% \times n \times y \times P + P \times y) < P \times n \times y$, that is, $x\% < 0.5 \times (1 - 1/n)$, because it is certain for $n \geq 2$ in a distributed database, thus if $x\% < 25\%$ then the search space of tuple of method distributed is less than the method centralized, that is, $t_{c2} > t_{d2} + t_{d3}$. According to the percent of FDs found in Table 5, if $n \geq 2$ then $0.5 \times (1 - 1/n) > 0.25 > 2.69\%$, it indicates that our distributed method has much less tuples involved in the checking. Actually, we can calculate the average percent of tuples not used in checking: $Q = ((P \times n \times y) - (2 \times 2.69\% \times n \times y \times P + P \times y)) \times 100\%/P \times n \times y = (0.9462 - 1/n) \times 100\%$, when $n \geq 2$, $Q \geq 44.62\%$.

V. Conclusion

In this paper, we proposed an framework for mining functional dependencies from large distributed databases. Our method can effectively reduce the number of tuples used in candidate FD checking and the transmitted data, and it can utilize effectively all computer resources in distributed system. To study the performance of our method some detail experiments will be done in future work.

Acknowledgements This work was supported in part by the Jiangsu Teachers University of Technology Science Foundation (No. Kyy06036) and by the Jiangsu Science Foundation of University (No. 08KJD520006).

References

[1] E. Codd, "Further normalization of the data base model," *Technical Report 909, IBM*, pp. 1–18, 1971.

[2] H. Mannila and K. Raiha, "The design of relational databases," *Addison Wesley*, 1994.

[3] M. Levene and G. Loizou, "A guided tour of relational databases and beyond," *Springer-Verlag, London*, 1999.

[4] F. S, F. F, G. S, and Z. E, "Repairing inconsistent xml data with functional dependencies," *Encycl Database Technol Appl Idea Group*, pp. 542–547, 2005.

[5] T. HBK and Z. Y, "Automated elicitation of functional dependencies from source codes of database transactions," *Inform Software Technol*, vol. 46(2), pp. 109–117, 2004.

[6] H. Y, K. J, P. P, and T. H, "Tane: an efficient algorithm for discovering functional and approximate dependencies," *Comput J*, vol. 42(2), pp. 100–111, 1999.

[7] N. N and C. R, "Fun: an efficient algorithm for mining functional and embedded dependencies," *In: Proceedings of the International Conference on Database Theory, London, UK*, pp. 189–203, 2001.

[8] Y. H, H. HJ, and B. CJ, "Fdmine: discovering functional dependencies in a database using equivalences," *In: Proceedings of the 2nd IEEE International Conference on Data Mining*, 2002.

[9] H. Yao, H. J, and Hamilton, "Mining functional dependencies from data," *Data Min Knowl Disc*, vol. 16, pp. 197–219, 2008.

[10] "http://pageperso.lif.univ-mrs.fr/ noel.novelli/datamining/fun/,"

[11] A. R, I. T, and S. AN, "Mining association rules between sets of items in large databases," *In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C.*, pp. 207–216, 1993.