International Journal of Advanced Information Science and Technology (IJAIST) *ISSN: 2319:2682*
Vol.4, No.12, December 2015 DOI:10.15693/ijaist/2015.v4i12.12-15

Comparative Study of Functional Dependency Generation Algorithms

W.C.Uduwela
Lecturer(prob)/Dep Mathematics and Computer Science, Faculty of Natural Sciences, The Open University of Sri Lanka

P.G.Wijayarathna
Senior Lecturer/Dep of Industrial Management, Faculty of Science, University of Kelaniya, Sri Lanka

Abstract **- Relational database model is the most common database model use of current Information Systems. The basis of its design process is Functional Dependencies. Various researches have been carried out to develop algorithms to discover the hidden Functional Dependencies in the existing data sets. The findings help for database designers in various ways: to database design verifications, to database management, to reverse engineering, and to query optimization. Therefore, it is important to find the most efficient algorithm since there are few. Four popular functional dependency algorithms (TANE, FD_Mine, Fast_FD and Dep_Miner) were selected to suggest the most efficient algorithm. Algorithms were analyzed based the data published in the literature and by implementing our own version of FD_Mine and Fast_FD algorithms for the missing data. According to the analysis we were able to conclude that the performance of TANE and FD_Mine are good for a large number of records while the performance of FastFD is good for a large number of attribute sets.**

Index Terms: Functional Dependencies, Relational Database, Relational schema, Functional Dependency Algorithms

I. INTRODUCTION

Relational database model is the most common database model spread over in commercial applications of information systems. Its efficiency fully depends on the relational schema (database schema): it describes the categorizations of the data and the relationships among them. The basis of the relation schema design process is Functional Dependencies (FDs): it describes relationships between attributes of the database relations. Further, an FD uniquely determines the value of an attribute with the values of some other attributes [1]. For an example in a student database student_name and student_address are determined by student_id. Formally a functional dependency can be denoted as $X \rightarrow Y$ in a relational schema R, where $X, Y \subseteq R$, is satisfied by r(R), if for all pairs of tuples ti, tj$\in$ r(U), if ti[X] = tj[X] then ti[Y]=tj[Y].

Typically, Functional Dependencies are obtained from the semantic model of the application domain [2,3], but various researches have been carried out to develop algorithms to discover hidden FDs in the existing data sets. These approaches, especially for knowledge discovery and data mining purposes [1]. Not only that, but also these help in various ways: to verify database design [4], to database management, to reverse engineering and, to query optimization [1]. Therefore, database designers, especially non technical people and novel database designers can get the help of these algorithms to make the correction in the existing relational schema as it is difficult to develop the correct relational schema at the beginning. In this paper, we do a comprehensive study of four popular functional dependency algorithms (TANE, FD_Mine, FAST_FD and Dep_Miner) to suggest the efficient algorithm among them, as I couldn't find a solution from the literature.

The paper has organized as follows. Section 2 reviews the existing approaches used in FD discovery. Section 3 describes the methodology adapted to analyze the selected approaches. Section 4 depicts the outcome and Section 5 concludes the paper.

II. FD DISCOVERY METHOD AND BASIC CONCEPTS OF THEM

Researchers motivate to find the efficient solutions and algorithms to discover the functional dependencies automatically from the datasets at the very beginning of 1980s [4]. These findings can be grouped as either top-down approaches or bottom-up approaches [4], but some researchers categorized them as either breadth-first search approaches or depth-first search approaches, respectively [5]. The rest of this paper considers the top-down and bottom-up categorization for its analysis.

Top-down approaches start by generating candidate FDs level-by-level, from short left hand side (lhs) to long lhs; it is like an attribute lattice (Refer Figure 01 for its illustration). Then it checks the satisfaction of the candidate FDs for satisfaction against the relation or its partitions [4]. TANE and FD_Mine [4, 5] are famous algorithms in this category and Table 01 describes the strategies they have used.

On the other hand, the bottom-up approaches, start with comparing tuples to get either agree-sets or difference-sets. Then it generates candidate FDs and check them against the agree-sets or difference-sets for satisfaction. FAST_FD and Dep_Miner [4, 5] are famous algorithms in this category and Table 01 describes the strategies they used.

The following concepts and terms are needed to understand the concepts in the above mentioned algorithms.

Basic Concepts of Functional Dependencies

Let F be a set of FDs over a dataset D and X be a candidate over D;

- Minimal FD — A FD, $X \rightarrow A \subset F$ is minimal, if A is not functionally dependent on any proper subset of X, i.e. if $Y \rightarrow A$ does not hold in F for any $Y \subset X$.
- Closure of FD — Closure of candidate X is denoted Closure (X) or X+, with respect to F, is defined as {Y | X →Y can be deduced from F by Armstrong's axioms}.
- Non Trivial FD — A FD, $X \rightarrow A \subset F$ is non-trivial if $A \notin X$.
- Nontrivial closure — Non trivial closure of candidate X denoted Closure'(X) with respect to F, is defined as Closure'(X) = Closure(X) – X.
- Candidate set — It is a combination of the attributes over the dataset.
- Partition of attributes (Equivalence class partition) — Partition of attribute A can be denoted as A(D) = {{t1, t2, t3, t4, t7}, {t5, t6}}. The values of tuples t1, t2, t3, t4, and t7 on attribute A are all the same, they are assigned to the same group. Likewise, as the values of t5 and t6 are the same, they are assigned into another group.
- Cardinality of a partition — Which is the number of groups in partitions, Ex :- According to the above example cardinality of A is, $|\pi A| = 2$.
- Minimal cover — It is the simplified set of FDs, that is equivalent to F. This means that they have the same closure of F+ as F and its no further reduction

Other concepts used in the algorithms

- Agree Set — Let t_i and t_j be tuples and X an attribute set. he tuples t_i and t_j agree on X if $t_i[X] = t_j[X]$.
- Disagree sets — If t_1 and t_2 do not appear together in some stripped partition, then $t_1$and t_2 disagree on every attribute. Such tuples that disagree form a set and such sets are said to be disagree sets.
- Maximal Set — A maximal set is an attribute set X which, for some attributes A, is the largest possible set not determining A.

Pruning rules used in TANE and FD_Mine algorithms:
Let X and Y are candidates over a dataset D;

- Rule 1 : If $X \rightarrow Y$ and $Y \rightarrow X$ hold, then X and Y are said to be equivalent candidates, denoted as $X \leftrightarrow Y$, then candidate Y can be deleted.
- Rule 2 : If X is a key, then any superset XY of X does not need to be checked.
- Rule 3: If Closure'(X) and Closure'(Y) are the nontrivial closures of attributes X and Y, respectively, then $XY \rightarrow$ Closure'(X) U Closure'(Y) does not need to be checked.
- Rule 4 : Let $X_1X_2\ldots X_k \rightarrow X_{k+1}$ be a k-level FD. If any subsets $X_{i(1)}X_{i(2)}\ldots Xi_{(k-1)}$ of $X_1X_2\ldots X_k$ satisfies $X_{i(1)}X_{i(2)}\ldots X_{i(k-1)} \rightarrow X_{i(k)}$, then $X_1X_2\ldots X_k \rightarrow X_{k+1}$ does not need to be checked.

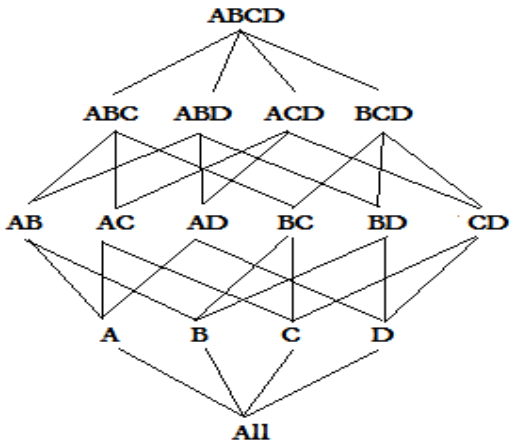

Figure 01. Example for the attribute lattice

Table 01. Summary of each tool and their approaches

Name of the Tool	Summary of the approach
[1] TANE: 1999 It discovers all minimal non trivial FDs.	It searches the FDs in level wise manner in the lattice while reducing the search space using pruning rules (Rule 2 and Rule 4). Results from the previous level are used for the later levels. To find the FDs, it represents the attribute sets as equivalence class partitions of the set of tuples and compares the cardinality of partitions. If the value of lhs is equal to rhs then there is an FD.
[8] Dep_Miner: 2000 It discovers all minimal non trivial FDs and the real world Armstrong relations.	This is based on the agree set. From agree set maximal sets are derived, and from maximal sets, all minimal non-trivial FDs are generated. Lhs of the FDs are generated from the complement of the maximal set.
[6] FastFD: 2001 It discovers all	First, it identifies the difference set (This is can be derived based on the

minimal non trivial FDs	agree set). Then it reduces to the set of minimal candidates by removing the superset of the difference set. The minimum cover of the difference set gives the minimal FDs.
[2,3] FD_Mine: 2002 It discovers all minimal non trivial FDs	The approach is same as TANE, but it uses the all four pruning rules to minimize the searching space of the lattice.

III. COMPARATIVE STUDY AND DISCUSSION

We reviewed the above stated FD algorithm approaches and compare their performances based on its pros and cons and the time taken to explore the FDs. E-mail survey was conducted for the analysis. As the response rate is not at the considerable level, tools were analyzed based the data published in the literature and implemented our own version of FD_Mine and FastFD algorithms for the missing data.

The article [6], had published a study on a comparison on running time between TANE, FastFD and Dep_Miner to explore the FDs. They had implemented their own versions of Dep_Miner and FastFD, while they had used the version available on the web for TANE. The experiment had conducted on the relations extracted from the UCI Machine Learning repository which is in online [7]. It says the running time meets or exceeds the Dep_Miner's performance by the FastFD performance for large data set and FsatFD becomes increasingly faster than Dep_Miner for large attribute set too. That means FastFD's performance is better than the Dep_Miner's performance for all types of datasets. However, the comparison between TANE and FastFD is not straight forward. FastFD had taken a long time than TANE when the data set is large, but TANEs performance is poorer than FastFD when the number of attributes getting larger.

A comparison between TANE and FD_Mine had done in the article published [3]. Although, they had used the same approach to develop both algorithms, the techniques used in FD_Mine reduce the search space by using four pruning rules described above. The analysis says that FD_Mine performance is better than TANE in execution time as well as the number of FDs to be checked (has to check lesser number of FDs than TANE). In their experiment they had fixed the number of tuples (records) into 100,000 and the number of attributes ranges from 10 to 60 for each experiment. It showed that the gap between execution times of the FD_Mine and TANE is strongly enlarged when the number of attributes increases.

We could not find the comparison between FastFD and FD_Mine in the literature; therefore our own versions of FD_Mine and FastFD were implemented using C# for the comparison. Algorithms were tested on few dataset using a personal computer installed with Windows 7 professional operating system, core i7 processor and 8GB RAM. Data sets with a large number of attributes and large number of record sets were used for the analysis. Table 02 shows the analysis of the result.

Further, the thesis [5] says that the output of the algorithm described in this paper is same for the same dataset. The summary of the analysis of TANE, Dep_Miner, FastFD and FD_Mine as presented in the Table 03.

Table 02. Time Taken to Find the Functional Dependencies

|r| Number of Raws, |R| Number of Attributes, |F| Number of Functional Dependencies Generated, ∘ - More than 1800 seconds				
|r|	|R|	|F|	FD_Mine (seconds)	FastFD (seconds)
217	05	08	0.061	35.21
217	15	311	267.640	366.641
50	20	3055	∘	228.979
50	26	3159	∘	537.84

Table 03. Summary of the Pros and Cons of the algorithms

Tool	Advantages	Disadvantages
[1] TANE: 1999	More appropriate if the dependencies are relatively small.Applicable to large data set too.	Has poor performance when attributes getting larger. Performance is poorer than FD_Mine
[8] Dep_Miner: 2000	This is the only solution that gives Armstrong's relation along with FDs	Performance is poorer than the Fast_FD
[6] FastFD: 2001	When attributes getting larger performance are better than the Dep_Miner, FD_Mine and TANE.	Performance is poor for large data set.
[2,3] FD_Mine: 2002	Reduces the search space and FDs to be checked than the TANE; therefore performance is better than FastFDalgorithm	Has poor performance when attributes getting larger.

IV. CONCLUSION

Analysis showed that all the algorithms generate the minimal non trivial functional dependencies as the outcome while the thesis [5] says that the outcome of each algorithm are same. Comparison of three algorithms FastFD, Dep_Miner, and TANE are taken from the research paper published [6], while comparison of TANE and FD_Mine also taken from the research paper published [3]. Own versions of FD_Mine and FastFD algorithms were used for their comparison. Since the tests were run in different environment direct comparison is impossible, but the results are, however indicative.

According to the analysis, we can conclude that if there is a large number of attributes the better algorithm is either TANE or FD_Mine. If there is a large number of recordsFastFD performance is better than TANE and

FD_Mine algorithms. Among TANE and FD_Mine, FD_Mine's performance is better than TANE performance.

REFERENCES

[1] H. Yk¨a, J. K¨arkk¨ainen, P. Porkka, and H. Toivonen, "Tane: An efficient algorithm for discovering functional and approximate dependencies," The computer journal, 42(2),pp.100–111, 1999.

[2] H. Yao, H. J Hamilton, and C. J. Butz., "Fd_Mine: functional dependencies in a database using equivalences. In Data Mining", ICDM2003. Proceedings. 2002 IEEE International Conference on, pages729–732. IEEE, 2002.

[3] H. Yao, H. J. Hamilton, "Mining functional dependencies from data," Data Mining and Knowledge Discovery, pp. 197-219, Volume 16 Issue 2, April 2008.

[4] Jixue Liu; Jiuyong Li; Chengfei Liu; Yongfeng Chen, "Discover Dependencies from Data—A Review," in Knowledge and Data Engineering, IEEE Transactions on , vol.24, no.2, pp.251-264, Feb. 2012

[5] K. Sood, "Comparison Of FunctionaLdependency Extraction Methods And An Application Of Depth First Search", M.Sc. Disseration, Department of Computer and Information Science and the Graduate School, University of Oregon, June 2014

[6] W. Catharine, C. Giannella, and E. Robertson, "Fastfds: A heuristic-driven, depth-first Algorithm for mining functional dependencies from relation instances extended abstract.", In Data Warehousing and Knowledge Discovery, pages 101–110. Springer, 2001.

[7] http://archive.ics.uci.edu/ml/

[8] L. St´ephane, J.-MarcPetit, and L. Lakhal., "Efficient discoveryof functional dependencies and Armstrong relations. In Advances in Database Technology EDBT 2000, pages 350–364.Springer, 2000

Authors Profile

W.C.Uduwelareceived the **B.Sc.** degree in Management and Information Technology from the department of Industrial Management, University of Kelaniya in 2008.Currently doing **M.Phil.** research in same department. Her research interest includes software enginnering.

Dr P.G.Wijayarathnareceived Dr. Eng. degree (specializing in Software Engineering) and M.Eng. degree from the University of Electro-Communications, Tokyo, Japan. He graduated from the Faculty of Science, University of Kelaniya with an Honors degree in 1984. His research interest includes Reasoning with spatiotemporal information, 3-D imaging, Specification based auto code/test case generation, Road traffic simulation and Web Engineering.