International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Algorithms for Conditional Functional Dependencies

P. Sobha Rani
M.Tech.
Pydah Engineering College, A.P., India.

A. Kamala Priya
Asst. Prof., Dept. of CSE,
Pydah Engineering College, A.P., India.

P. Suresh Babu
Assoc. Prof., Dept. of CSE,
Kaushik Engineering College, A.P., India.

***Abstract* — Functional dependencies (FDs) are fundamental constraints that define the relation between attributes in a database. Key relationships are special kinds of functional dependencies, and FDs provide a mechanism for database normalization during the design process. Conditional Functional Dependencies (CFDs) extend standard functional dependencies (FDs) by enforcing patterns of semantically related constants. CFDs have been proven more effective than FDs in detecting and repairing inconsistencies (dirtiness) of Data. To make CFD methods to be effective in practice, it is necessary to discover CFDs from the sample data. This paper presents an overview of discovering conditional functional dependencies using algorithms like CFDMiner, CTANE, Apriori-like and close-like.**

***Keywords* — Apriori, Conditional functional dependency, Functional dependency, Powerset.**

I. INTRODUCTION

Functional dependencies (FDs) are fundamental constraints that define the relation between attributes in a database. Key relationships are special kinds of functional dependencies, and FDs provide a mechanism for database normalization during the design process. In its most basic form, a functional dependency over a relation instance r of a relation schema R is an expression $X \rightarrow Y$, where X, Y $\subseteq$ R. R is said to satisfy the dependency $X \rightarrow Y$ if all tuples with any fixed value of the attributes X share a single value of the attributes Y . This captures the idea that the values in Y depend on the values in X.

Conditional functional dependencies (CFDs) [1][7] were recently introduced for data cleaning. They extend standard functional dependencies (FDs) by enforcing patterns of semantically related constants. CFDs have been proven more effective than FDs in detecting and repairing inconsistencies (dirtiness) of data [1][2], and are expected to be adopted by data-cleaning tools that currently employ standard FDs. However, for CFD-based cleaning methods to be effective in practice, it is necessary to have techniques in place that can automatically discover or learn CFDs from sample data, to be used as data cleaning rules. Indeed, it is often unrealistic to rely solely on human experts to design CFDs via an expensive and long manual process.

Discovery of dependencies existing in an instance of a relation received considerable interest as it allowed automatic database analysis. Knowledge discovery and data mining [3], database management [4], reverse engineering [5] and query optimization [6] are among the main applications benefiting from efficient dependencies discovery algorithms. New dependencies on instances of relations where defined. The discovery problem in CFDs, however, highly nontrivial. It is already hard for traditional FDs since, among other things, a canonical cover of FDs discovered from a relation r is inherently exponential in the arity of the schema of r, i.e., the number of attributes in R. Since CFD discovery subsumes FD discovery, the exponential complexity carries over to CFD discovery. Moreover, CFD discovery requires mining of semantic patterns with constants, a challenge that was not encountered when discovering FDs

A CFD is defined as follows: Let R be a relation schema defined over a set of attributes Attr(R). The domain of each attribute $A \in$ Attr(R) is denoted by Dom(A). A CFD φ defined on R is a pair (X $\rightarrow$ Y, Tp), where (1) X $\rightarrow$ Y is a standard FD, referred to as the FD embedded in φ ; and (2) Tp is a tableau with attributes in R, referred to as the pattern tableau of φ , where for each $A \in$ Attr(R) and each pattern tuple tp $\in$ Tp, tp[A] is either:

- a constant 'a' in Dom(A),
- an unamed variable $\top$ that draws values from Dom(A),
- or an empty variable $\perp$ which indicates that the attribute does not contribute to the pattern (i.e. $A \notin X \cup Y$).

II. CFD MINER ALGORITHM

CFD Miner algorithm is used to discover constant conditional functional dependencies. For a given instance r and threshold k, the algorithm finds a canonical cover of k- frequent minimal constant CFDs of the form ($X \rightarrow A$, (tp $\parallel$ a)). Our algorithm is based on the connection between leftreduced constant CFDs and *free* and *closed itemsets*. An itemset is a pair (X, tp), where X $\subseteq$ attr(R) and tp is a constant pattern over X. An itemset (X, tp) is called *closed* in r if there is no itemset (Y, sp) such that (Y, sp) $\prec$ (X, tp) with supp(Y,sp, r) = supp(X, tp, r). Similarly, an itemset (X, tp) is called *free* in r if there exists no itemset (Y, sp) such that (X, tp) $\prec$ (Y, sp) for which supp(Y, sp, r) = supp(X, tp, r). CFDMiner algorithm works as follows:

(1) For each k-frequent closed itemset (X, tp) we add its free sets, as given by C2F, to a hash table H.

(2) For each closed itemset (X, tp), we associate with each of its free itemsets (Y, sp) the itemset RHS(Y, sp) = (X \ Y, tp[X \ Y]). That is, we associate with each free set the candidate RHS attributes in their corresponding constant CFDs.

During this process, an ordered list L of all k-frequent free itemsets is constructed as well. Itemsets in this list are sorted in ascending order with respect to their sizes.

(3) For each free itemset (Y, sp) in the list L, CFDMiner does the following:

International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 6, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

(a) For each subset $Y^1 \subseteq$ Y such that (Y^1, sp[Y^1]) ∈ L, it replaces RHS(Y, sp) with RHS(Y, sp) ∩ RHS(Y^1, sp[Y^1]). It implies that only those elements in RHS(Y, sp) can lead to a left-reduced constant CFD that are not already included in some RHS(Y^1, sp[Y^1]) of one of its sub-itemsets. It is important to remark that the subset checking can be done efficiently by leveraging the hash-table H.
(b) After all subsets of (Y, sp) are checked, CFDMiner outputs k-frequent constant CFDs (Y → A, (sp || a)) for all (A, a) ∈ RHS(Y, sp).

III. CTANE ALGORITHM

CTANE is a level wise algorithm for discovering minimal, k-frequent CFDs. It is an extension of algorithm TANE[8] for discovering FDs. CTANE mines CFDs by traversing pattern lattice L in a level wise way. To find all valid minimal non-trivial dependencies in a level wise manner, CTANE searches the lattice in a levelwise manner. A level Ll is the collection of attribute sets of size l. CTANE starts with L1={{A}|A ∈ R} and computes L2 from L1, L3 from L2 and so on, according to the information obtained during the algorithm. The algorithm works as follows:
Input: relation r over schema R, FDs
Output: minimal non-trivial functional dependencies that hold in r
1 L0 := {Ø}
2 C(Ø):=R
3 L1 :={{ A } | A ∈ R}
4 l:=1
5 While (Ll ≠ Ø)
6 PRUNE(Ll)
7 Ll+1:=GENERATE_NEXT_LEVEL(Ll)
8 l := l+1

The procedure PRUNE(Ll) prunes the search space by deleting sets from Ll. The procedure GENERATE_NEXT_LEVEL(Ll) forms the next level from the current level. The specification of GENERATE_ NEXT_LEVEL is Ll+1:={ X | |X|=l+1 and for all Y with Y ∈ X and |Y|=l we have Y ∈ Ll}
The procedure PRUNE(L1) is as follows:
Procedure PRUNE(L1)
1 for each X ∈ L1 do
2 if C(X) = Ø do
3 delete X from L1
4 if X is a (super)key do
5 for each A ∈ C(X)\X do
6 if A ∈ $\bigcap_{B \in X}$ C(XU {A} \ {B}) then
7 outputX—>A
8 delete X from L1.
The procedure GENERATE_NEXT_LEVEL(Ll) is as follows:
Procedure GENERATE_NEXT_LEVEL(L1)
1 L_{l+1} :=Ø
2 for each K E PREFIXBLOCKS(Ll) do
3 foreach{Y,Z} ⊆ K,Y ≠ Zdo
4 X:=YUZ
5 if for all A ∈ X, X\ {A} ∈ L1 then
6 L_{l+1} := L_{l+1} U {X}
7 return L_{l+1}

IV. NAÏVE APRIORI- LIKE ALGORITHM

Naïve apriori-like algorithm is used to discover conditional functional dependencies in the powerset of attributes. Let R be a relation schema defined over a set of attributes Attr(R). The domain of each attribute A ∈ Attr(R) is denoted by Dom(A). For an instance r of R, a tuple t ∈ r and X a sequence of attributes, we use t[X] to denote the projection of t onto X.

Given 'r' a relation, X ⊆ Attr(r) and an attribute A ∉ X. The dependency X → A is a functional dependency of r (i.e. A ∈ θ(X, r)) if and only if ψ (X /→A) = ∅. This means X → A is an FD if and only if the dependency holds in all X-complete fragment relations. If the dependency does not hold in (at least) one of the X-complete fragment relations we thus have a CFD. Given 'r' a relation, X ⊆ Attr(r) and an attribute A ∉ X. The pair (X → A, ψ (X → A)) is a CFD of r if and only if ψ (X → A) ≠ ∅ and (X /→ A) ≠ ∅. The algorithm is as follows:
Algorithm:
Input: r - a relation
Output: Σ the set of all CFDs
1 Σ ← ∅
2 forall X ⊆ Attr(r) do
3 forall A ∉ X do
4 if ψ (X /→A) = ∅
5 Σ <- Σ U {(X → A, Top(X U {A})}
6 else if ψ (X → A) ≠ ∅
7 Σ <- Σ U {(X → A, ψ (X → A))}

V. NAÏVE CLOSE- LIKE ALGORITHM

Naïve close- like algorithm is used to discover conditional functional dependencies based on closed sets lattice [9]. The algorithm is to generate a subset of FDs and CFDs such that the others FDs and CFDs can be derived from this subset. We propose to generate FDs such that their left-part is a minimal generator and CFDs which are pure CFDs.

The minimal generator is defined as: A subset X of attributes of r is said to be a minimal generator of Γ (X, r) if and only if for any Y ⊂ X we have Γ (Y, r) ≠ Γ (X, r). By extension, we also say that X is a minimal generator

of θ(X, r). The algorithm computes all CFDs of the form (X → A, Tp) such that:

• if Tp = {Top(X U {A})} (i.e. it is an FD) then X is a minimal generator of θ (X, r);

• Otherwise X = θ (X, r).

Algorithm:

Input: r - a relation

Output: Σ the set of minimal generators FD and closed CFD

1 Σ ← ∅
2 Gen = {{}}
3 while Gen ≠ ∅ do
4 forall X ∈ Gen do
5 forall A ∉ X do
6 if A ∈ θ (X, r) then
7 Σ <- Σ U {(X → A, Top(X U {A})}
8 else if ψ (X → A) ≠ ∅
9 Σ <- Σ U {(θ (X, r) → A, ψ (X → A))}
10 en ← Compute next minimal generators

VI. Conclusion

Conditional Functional Dependencies (CFDs) are an extension of Functional Dependencies (FDs) by supporting patterns of semantically related constants, and can be used as rules for cleaning relational data. In this paper we present four algorithms for discovering conditional functional dependencies. CFDMiner algorithm is used for mining minimal constant CFDs, a class of CFDs important for both data cleaning and data integration. CTANE algorithm is used for discovering general minimal CFDs based on the levelwise approach. CTANE is an extension to TANE. The algorithm Tane uses partitions to discover FDs. Their approach is similar to the Apriori algorithm in the sense that the search space is the powerset lattice. The algorithm naïve apriori- like algorithm is based on powerset lattice. The algorithm naive close algorithm as an improvement of the TANE algorithm by reducing the search space to the closed set lattice.

References

[1] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *TODS*, vol. 33, no. 2, june 2008

[2] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: Consistency and accuracy," in *VLDB*, 2007.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In VLDB, pages 487–499, 1994.

[4] S. Bell and P. Brockhausen. Discovery of data dependencies in relational databases. Technical Report LS-8 Report-14, University of Dortmund, 1995.

[5] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In ICDE, pages 218–227. IEEE Computer Society, 1996.

[6] G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. ACM Transactions on Database Systems, 17(1):32–64, 1992.

[7] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In ICDE, pages 746–755, 2007.

[8] Y. Huhtala, J. K¨arkk ainen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.

[9] M. J. Zaki and C.-J. Hsiao. Charm: an efficient algorithm for closed itemsets mining. In Proceedings of the Second SIAM International Conference on Data Mining, pages 457–473, 2002.