Trends in Applied Sciences Research

ISSN 1819-3579

Academic Journals Inc.

www.academicjournals.com

Trends in Applied Sciences Research 7 (4): 285-302, 2012
ISSN 1819-3579 / DOI: 10.3923/tasr.2012.285.302

CFD-Mine: An Efficient Algorithm For Discovering Functional and Conditional Functional Dependencies

[1]Musbah M. Aqel, [2]Nidal F. Shilbayeh and [3]Mohammed S. Hakawati
[1]Faculty of Science and Information Technology, AlZarqa University, Zarqa, Jordan
[2]Faculty of Computer and Information Technology, University of Tabuk, Tabuk, Saudi Arabia
[3]Faculty of Information Technology, Middle East University, Amman, Jordan

Corresponding Author: Nidal F. Shilbayeh, Faculty of Computer and Information Technology, University of Tabuk, Tabuk, Saudi Arabia

ABSTRACT

Dirty data is a serious problem that affects many enterprises across all aspects of their business ranging from operational efficiency to revenue protection. In this study, we present a new algorithm called CFD-Mine that efficiently discovers all possible rules and determines the minimum set of these rules using functional Dependencies (FDs) and Conditional Functional Dependencies (CFDs) for detecting inconsistencies in data. The algorithm is based on the level-wise search algorithm that extends TANE, a well-known algorithm for discovering FDs. CFD-Mine searches for the minimum CFDs among the data values and prunes redundant candidates. The conducted experiments show that CFD-Mine is scalable and applicable to work efficiently when the data sets are large.

Key words: Data cleaning, functional dependency, conditional functional dependency, data quality, data inconsistency

INTRODUCTION

The presence of errors and inconsistencies in data dramatically reduce the value of data, making it worthless, or even harmful. In 2000, statistics estimated that data quality costs US businesses $600 billion annually (English, 2000). It is also estimated that data cleaning, a labor-intensive and complex process, accounts for 30 to 80% of the development time and budget in most data warehouse projects (Shilakes and Tylman, 1998). Studies conducted by many researchers forecast that more than 50 percent of data warehouse projects will have limited success, or will be outright failures, as a result of the lack of attention to data quality issues (Sagiroglu and Ozturan, 2006; Sahai *et al.*, 2005; Wei *et al.*, 2007; Al-Nabhan *et al.*, 2006). As a result of the previous statistics and studies, there is an increasing demand for data cleaning tools that automatically detect and effectively remove inconsistencies and errors from the data.

Dirty data often arises due to changes in use and perception of the data and violation of integrity constraints (or lake of such constraints) (Chiang and Miller, 2008). Inconsistencies and errors in a database often emerge as violations of integrity constraints (Arenas *et al.*, 2003; Rahm and Do, 2000; Renuga and Sadasivam, 2009).

Constraint-based data cleaning has mostly focused on two concepts, introduced by Arenas *et al.* (2003), a repair and a consistent query answer. A repair is to find another database that is consistent and differs minimally from the original database. A consistent query answer is to find an answer to a given query in every repair of the original database, without editing the data.

The limitations in traditional dependencies lead the researchers in data cleaning to revive actions by considering extensions of FDs and INDs (Inclusion Dependencies), referred to as Conditional Functional Dependencies (CFDs) and Conditional Inclusion Dependencies (CINDs) respectively. The extended functional dependencies additionally specifying patterns of semantically related values; these patterns impose conditions on what part of the relation(s) the dependencies are to hold and which combinations of values should occur together (Fan *et al.*, 2009).

In this study, we present a new approach that uses recent advances in constraint-based data cleaning and the CFDs rules to repair the dataset based on two main phases:

- Discovering rules, to find the rules that the relation depends on
- Repairing inconsistencies, by identify tuples that have some error in some of its fields (violate the discovered rules)

The main contributions of this research paper are the following:

- We developed an algorithm for discovering a minimal set of Conditional Functional dependencies (CFDs) and a minimal set of Functional Dependencies (FDs). Even though, the underlying ideas are not new, but this is the first attempt to develop a concrete algorithm for discovering both rules from a large datasets
- We used two new pruning techniques in our algorithm to reduce the number CFDs to be checked hence improving its performance. The first technique merges the similar CFDs for discovering a few and more accurate rules, while the second technique finds the minimal set of CFDs based on the intersect partitions between the candidates that formed the rules
- We implemented an application for discovering the CFDs and FDs from any dataset located anywhere and with any extension. The application generates the partitions for the attribute set and then generates the rules. The application also allow us to change the accuracy of the discovered CFD rules and filter the discovered rules after generate them
- The results of experiments we conducted on syntactical datasets shows that our algorithm is scalable and can be used to discover FDs and CFDs in a large datasets

PRELIMINARIES

Relation schema definition: A relation schema R is a finite set of attributes. The domain of an attribute A, denoted by Dom (A) is the set of all possible values of A, a tuple t over a relation schema $R = \{A_1 \ldots \ldots A_m\}$ is a member of the Cartesian product Dom $(A_1) \times \ldots \times$ Dom (A_m), a relation r over R is a finite set of tuples over R. The cardinality of a set X of tuples is denoted by $|X|$.

If $X \subseteq R$ is an attribute set and t a tuple over R, we denote by t (X) the restriction of t to X. The projection of a relation r over R onto X is defined by $\pi_x (r) = \{t(x) \mid t \in x\}$. A database schema R is a finite set of relation schemas R_i, A database *d* over R is a set of relations r_i over each $R_i \in R$.

Functional dependency definition: Let r (R) be a relation and X, Y $\in$ R. A Functional Dependency (FD) is a constraint, denoted by $X \rightarrow Y$. The FD $X \rightarrow Y$ is satisfied by r(R) if every two tuples t_i, $t_j \in r(R)$ that have $t_i(X) = t_j(X)$ also have $t_i(Y) = t_j(Y)$. In an FD $X \rightarrow Y$, we refer to X as the antecedent and Y as the consequent.

Levelwise algorithm: The idea of the levelwise algorithm is to start from the most general attributes and try to generate and evaluate more and more specific attribute set (Pramada *et al.*, 2012; Qi and Wei, 2008; Sharma *et al.*, 2007; Thakur *et al.*, 2007). The semi-lattice in Fig. 1 shows

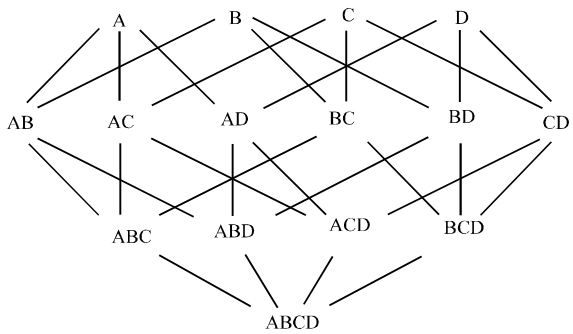

Fig. 1: Lattice for 4 attributes

the search space of an exhaustive algorithm for finding rules for four attributes and all possible nonempty combinations of the four attributes (A, B, C and D) (Mannila and Toivonen, 1997).

Conditional functional dependency definition: CFD extend FD by incorporating a pattern tuple of semantically related data values.

For each attribute A in a schema R, we denote its associated domain as Dom (A) which is either infinite (e.g., string; real) or finite (e.g., Boolean; date). A CFD f on R is a pair (R: X → Y, Tp).

Where:

- X and Y are sets of attributes in attr (R)
- X → Y is a standard FD, referred to as the FD embedded in φ
- Tp is a tableau with attributes in X and Y, referred to as the pattern tableau of φ, where for each A in X → Y and each tuple t ϵ Tp, t(A) is either a constant 'a' in dom(A), or an unnamed variable '_' that draws values from Dom(A) (Fan *et al.*, 2009; Wang *et al.*, 2011)

Medina and Nourine (Raoul and Lhouari, 2008), present the idea of decomposition the relation into a small relations (X-complete horizontal decomposition) denote by RX(r) the set of all X-complete fragment relations of r. More formally: RX (r) = {r' → r | r is X-complete}.

When stating that FD holds on the entire of relation, the CFD is a FD hold on a sub relation of R, but to find a hybrid idea between them, let's consider the decomposition of the relation R into small sub relation based on CFD which means that these CFDs holds on a specific sub relation and maybe interleaved with another sub relation.

Table 1 shows a sample records from library instance which contains records about items available in the library and its attributes are Name, Type, Country, Price and Tax and this relation holds on this Functional Dependency:

FD: (Name, Type, Country) → (Price, Tax)

Intuitively, we can recognize the following inconsistencies:

- The entire Harry Potter books sell in France don't have any tax rate, but in t_9 we notice that this tuple has 0.05 tax rates which violate the semantic constraint
- There are two different prices to the same item in same country which means that one of these tuples violates the semantic constraint

Table 1: Library relation instance

	Name	Type	Country	Price	Tax
t_1	Harry Potter	Book	France	10	0.00
t_2	Terminator	DVD	USA	40	0.08
t_3	Harry Potter	Book	France	10	0.00
t_4	Armani Suit	Clothing	UK	500	0.05
t_5	Armani Slacks	Clothing	UK	250	0.00
t_6	Star Wars	DVD	UK	25	0.00
t_7	Terminator	DVD	USA	25	0.08
t_8	Prada Shoes	Clothing	France	500	0.05
t_9	Harry Potter	Book	France	10	0.05
t_{10}	Harry Potter	Book	France	10	0.00
t_{11}	Prada Shoes	Clothing	France	200	0.05

As we noticed in the previous two cases, these tuples which violate the semantic constraints but don't violate the FD constraint, i.e., this Functional Dependency does not help us to find the tuples that violate the sales rules in the library.

Let's define a new type of rules to help us avoid these violations:

φ_1: ([Name, Type="Book", Country="France") → [Price, Tax = 0])
φ_2: ([Name, Type, Country="USA") → [Price, Tax])

This type of constraint is called Conditional Functional Dependency which is FD but has some constant values to help users in data cleaning phases.

The first CFD (φ_1) prevents the user in the library system from violating this rule: if the book sells in France, then no tax rate is added. While as the second CFD (φ_2) means that in the USA country, the name and the type of items define the price and the tax for them which prevent the same item to have two different prices. These rules do not violate the FD that the relation holds in, but added some consistency and accuracy to the relation.

RELATED WORK

As the discovering of Functional dependencies take a lot of work from the researchers of the database and data cleaning system, the approaches for discovering the FD are varied and have different options and pruning phases.

As tight relation exists between FDs and CFDs, we can think that FDs discovery approaches can apply to discover CFDs too. The authors (Fan *et al.*, 2009), divide the discovering of CFD into three methods .The first, referred to as CFDMiner, is based on techniques for mining closed item sets and is used to discover constant CFDs, namely, CFDs with constant patterns only. The other two algorithms are developed for discovering general CFDs.

The first algorithm, referred to as CTANE, is a levelwise algorithm that extends TANE, a well-known algorithm for mining FDs. The other, referred to as FastCFD, is based on the depth first approach used in FastFD, a method for discovering FDs.

It leverages closed-item set mining to reduce search space. The authors demonstrate the following. (a) CFDMiner can be multiple orders of magnitude faster than CTANE and FastCFD for constant CFD discovery. (b) CTANE works well when a given sample relation is large, but it does not scale well with the arity of the relation. (c) FastCFD is far more efficient than CTANE when the arity of the relation is large.

CFD-MINE ALGORITHM

CFD-Mine is a level-wise search algorithm for mining the CFD Rules, that each Candidate (element on the lattice) at level k is used to discover the results at level k+1. The approach uses multi pruning phases to filter the discovered rules, finds a set of minimum CFDs which is equivalent to another set of CFDs discovered by different approaches.

CFD-Mine approach performs a level-wise search on the lattice for finding partitions to the candidates and generating the CFDs between adjacent levels. Top-down search in the lattice starts from singleton sets and proceeds downwards searching for bigger sets.

At level one, CFD-Mine starts from Singleton Candidates (i.e., single attributes set available in the relation) and stores them in a variable C_1. At level two, each element in C_1 is used to generate Candidates of the form (x_1x_2) where, $\{x_1,x_2 \in C_1\}$ and $\{x_1 \neq x_2\}$ and stores them in another variable C_2.

After finding all the Candidates in both levels (one and two) and stores them in C_1 and C_2, respectively, all the FDs available between these two levels are discovered and stored in variable called F. Also, All the CFDs of the following form are discovered and stored in a variable CF:

$$\varphi : (q = xi, p = \emptyset) \rightarrow (vi)$$

Where, xi is a single value from C_1 and vi is a single value from C_1 added to xi to represent a Candidate in C_2.

For instance, if there is a relationship between (B, AB), then the form of CFD is:

$$\varphi : (q = B, p = \emptyset) \rightarrow (A)$$

At level three, the Candidate set available in C_2 is used to generate the third level of Candidates and stores them in C_3. After that, all FDs available between level two and level three are discovered and added to the previous FDs stored in variable F and all CFDs of the following form is generated and added again to the previous CFDs stored in CF:

$$\varphi : (q = xi, p) \rightarrow (vi)$$

For instance, to find a relation between elements on the edge (AB, ABC), the forms of CFDs are one of the following:

φ: (q = A, p = B) →(C), or
φ: (q = B, p = A) →(C), or
φ: ([q = B, A), p = ø) →(C)

Figure 2 shows CFD-Mine pseudo code which is in Object Oriented Programming (OOP) Language present the main class and this class calls five different methods (procedures) which are: SingletoCalculatePartition, CalculatePartition, AprioriGen, ObtainCFDs and PartitionMinimalCover.

In the following Sub-Sections we explain each of the above procedures and we illustrate how CFD-Mine algorithm works.

Generate next level candidates: The AprioriGen procedure is used to generate all possible Candidates at level k from the Candidates at level k-1. Figure 3 shows the AprioriGen procedure.

```
CFD-MINE Algorithm (r (U))
Input: A relation r (U) over U = {v1...vm}
Output: A set of FDs and CFDs over r (U).
{
Initialize variables step:
1.      CF = ø;
2.      C1 = U;
3.      SingletonCalculatePartition (C1, r (U));
Iteration step:
4.      while |Ck| > 0 do
5.      {
6.      k = k + 1;
7.      AprioriGen (Ck-1);
8.      CalculatePartition (Ck, r (U));
9.      CF ∪ ObtainCFDs (Ck-1, Ck);
10.     MinimalCover (CF);
11.     }
12.     return (CF);
}
Where,
CF      : variable to store all CFDs discovered
during the algorithm progress.
C1      : variable to store singleton Candidates
attribute in relation.
Ck      : variable to store all results comes from
calling the sub algorithms.
k = 1   : variable present the level; where the
algorithm works on.
```

Fig. 2: CFD-Mine pseudo code

```
AprioriGen (Ck-1)
{
1.      Ck = ø;
2.      for each {y, z}⊆ Ck-1, y≠z do
3.          x = y ∪ z;
4.          if for each A∈x, x\{A}∈CK-1  then
5.              Ck = Ck ∪{x};
6.      return Ck;
}
Where,
z, y    : Attribute set at level k.
x       : The new value at level k+1.
```

Fig. 3: AprioriGen procedure

Computing partitions: Partitions definition: Two tuples t and u are equivalent with respect to a given set X of attributes, if t (A) = u (A) for all A in X . Any attribute set X partitions the tuples of the relation into equivalence classes. We denote the equivalence class of a tuple t ϵ r with respect to a given set X ϵR by (t) X, i.e., (t) X = {u ϵ r | t (A) = u (A) for all A ϵ X}. The set ΠX = {(t) X | t ϵ r} of equivalence classes is a partition of r under X.

Π_X is a collection of disjoint sets (equivalence classes) of tuples and each set has a unique value for the attribute set X and the union of sets equals to the relation r. The rank |Π| (aka cardinality) of a partition Π is the number of equivalence classes in Π.

Stripped partition definition: is a partition with equivalent classes of one element size and this element is removed, for instance, the Stripped Partition of the Name Attribute set is Π_{Name} = {{1, 3, 9, 10}, {2, 7}, {8, 11}}, we will not mention it because we will use it as a default partitions, The benefit of using stripped partition is to reduce the comparison space in finding CFDs.

Singleton calculate partition: procedure shown in Fig. 4 is used to find all equal tuples in the attribute set that have the same value in the domain for the single attribute set only.

The partitions are not computed from scratch (Lattice) for each attribute set. Instead, Fig. 5 shows the CalculatePartition procedure that is used to move through the lattice and compute a partition as a product of the two previously computed partitions (in the previous level). The product of the two partitions Π' and Π'', denoted by Π' × Π'', is the least partitions that refines both Π' and Π''.

We compute the partitions Π_X, for each X ϵ R, directly from the database if the value of X = 1. If X = 2 the partitions Π_X are computed as a product of partitions with respect to the two subsets of X. Any two different subsets of size |X|-1 will do which is convenient for the level-wise algorithm since only the partitions from the previous level are needed.

```
SingletonCalculatePartition (C1, r (U))
{
1.      i , j = ø;
2.      for each t[i ] ∈ dom(attr[A]) do
3.      for j=0 to Table. length
4.      If (value [i] = value [j])
5.      t[j]=t[j] ∪ i ;
6.      break;
7.      return t[j];
}
Where,
t[i]: The index number of the tuples.
attr[a]: The attribute set in the level.
value : Value domain available in the tuple of
attribute, Dom (attr [A]).
t[j]: Array of lists to store the partitions.
```

Fig. 4: SingletonCalculatePartition algorithm

```
CalculatePartition (Ck, r (U))
{
1.    n, m = ø;
2.    for each (t[y], t[z])⊆ Ck ,t[y]≠ t[z] do
3.      for each part[n] ∈ t[y]
4.          for each part[m] ∈ t[z]
5.              t[x]=(part[n]-part[m])∪(part[m]-part[n]);
6.                          break;
7.    return t[x];
}
Where,
n, m      : Numeric values present the index to the
partitions to each Candidate.
part      : Array of tuples, present the partitions.
```

Fig. 5: CalculatePartition algorithm

Searching for rules: Now, we use the partitions found for each attribute set in each level stored in C_k to find the CFDs Rules.

The ObtainCFDs procedure shown in Fig. 6 is used to generate the CFDs as in the following steps:

Step 1: ObtainCFDs procedure receives two complete levels and compares each element at level C_{k-1} with each related element at level C_k and check the candidate element at level C_{k-1} that is a portion of the candidate element at level C_k, (i.e., $C_{k-1} \subseteq C_k$). For instance, in Fig. 1 the element (AC) is a portion of the element (ABC) at the next level

Step 2: If the partitions of these elements are exactly equal (i.e., identical) then there is a FD between them (nontrivial FD). For instance, in the previous Table the partition after remove stripped partitions for Name and (Name, Type) candidates are:

$$\Pi_{name} = \{1, 3, 9, 10\}, \{2, 7\} \{8, 11\}$$
$$\Pi_{name}, \text{Type} = \{1, 3, 9, 10\}, \{2, 7\}, \{8, 11\}$$

Which mean there exists a FD between them, FD: Name → Type

Step 3: If the partitions of these elements are not exactly equal (i.e., there is at least one partition shared) then maybe there is a CFD exists. The IntersectPartitions procedure shown in Fig. 7 is used to find the same equivalence classes (Intersect Partitions) that are equal in both elements

Step 4: CreateCFD procedure shown in Fig. 8, receives two Candidates and the shared partitions between them and produces CFD rule in this manner; it is try to finds an element in the LHS candidate that contains the same Shared partition found by IntersectPartitions procedure; if it is found, then it is a condition partition or constant, if it is not then the element is variable and its values are from the domain of its attribute. This operation is repeated for the RHS. This step reduces the number of CFDs discovered by giving a direct CFD after merging a lot of CFDs Rules

Pruning the discovered CFDs: Our approach contains several pruning phases that are used to reduce the number of CFDs to be checked, these phases have good effects in improving the performance of the algorithm.

```
ObtainCFDs (Ck, Ck-1)
{
1.      F= ø;
2.      for each x in Ck -1
3.          for each vi ∈ U – x+
4.          if (|Πx| = |Πxvi|)
5.            F =F ∪ (FD : [x →   vi ])
6.          else
7.            Ωx= IntersectPartitions (x, xvi);
8.            if (Ωx = ø)
9.                break;
10.            else
11.               CreateCFD (Ωx, x, vi);
12.         return Ωx;
}
Where,
x+          : The closure of the element x, i.e. the
element that x contains it.
Ωx          : Variable to store the share partition
between two elements in two levels.
```

Fig. 6: ObtainCFDs procedure

```
IntersectPartitions (x, xvi)
{
1.   n, m = ø;
2.   for each part[n] ∈ x
3.     for each part[m] ∈ xvi
4.         if((part[n] = part[m] )&&(part size ≥r))
5.             Ωx = Ωx ∪ part[m] ;
6.         break;
7.   return Ωx.
}
Where,
n, m        :  Numeric values present the index to
the partitions in each Candidate.
part        : Array of tuples, which represent the
partitions.
part_size: Number of tuples in each group.
r           : Threshold value.
```

Fig. 7: Intersect Partitions procedure

```
CreateCFD (Ωx, x, vi)
{
1.for each h ∈  x do
2.if Ωx ⊆ h then
3.    q = q  ∪ h;
4.else
5.    p = p ∪ h;
6.      if Ωx ⊆ vi then
7.        CF=CF ∪ ω= [q = value of dom (x), p] → [vi
= value of dom (vi)];
8.      else
9.        CF=CF ∪ ω= [q = value of dom (x), p] → [vi];
10.         return CF;
}

Where,
q : variable to store the Conditional Attributes.
p : variable to store the Variable Attributes.
```

Fig. 8: CreateCFD procedure

Stripped partitions: As we mentioned early, the partition with one element size is removed from the search space for the following reasons:

- Reduce the search space for finding the CFDs
- Prevent CFD comes from single tuple to appear in the final rules and this means that there is no CFD Rule that has constant values in all of its attributes of the rules which means that the static rules are pruned

Merging similar CFDs: In our approach, we added the idea of merging CFDs rules based on the similarities between their attributes. This idea is presented in the third inference rules in (Fan *et al.*, 2009). The idea of merging rules based on finding the attributes have the same value and make it Condition.

Minimal cover for CFDs: Before exploring the modified algorithm for finding the minimum CFD, we will study the inference axioms for CFD given in Fig. 9 and their relations with CFD-Mine.

FD1: Extends Armstrong's axioms of Reflexivity, because CFD-Mine algorithm finds the CFD between two adjacent levels and because it discovers the nontrivial CFD, then this inference rule doesn't have any effect in our algorithm.

Now, as an application of consistency and implication analyses of CFDs, we present a modified algorithm for computing a minimal cover MCF of a set CF of CFDs based on the Intersect Partitions between the candidates which produce the Rules that can be reduced and eliminated from the CF set.

The cover MCF is equivalent to CF but does not contain redundancies and thus is often smaller than $\sum$. Since the costs of checking and repairing CFDs are dominated by the size of the CFDs to be checked along with the size of the relational data, a non-redundant and smaller MCF typically leads to less validating and repairing costs. Thus finding a minimal cover of input CFDs serves as an optimization strategy for data cleaning.

A minimal cover MCF of a set S of CFDs is a set of CFDs such that:

- Each CFD in MCF is of the form $(R: X \rightarrow A, tp)$ as mentioned earlier
- MCF = CF
- No proper subset of MCF implies MCF and
- For each $\varphi = (R: X \rightarrow A, tp)$ in MCF, There exists no $\varphi = (R: X \rightarrow A, tp\ (X \cup A))$ in MCF such that $X \subset X$. Intuitively, MCF contains no redundant CFDs, attributes or patterns

Now, if we applied the inference axioms for finding the minimal cover set of CFDs, we will see that the FD1, doesn't have any effect on CFD-Mine, because already the finds the nontrivial CFDs between adjacent levels.

About the second axiom FD2, maybe there is an equivalence set of attributes between the discovered CFDs and as proposed by Yao *et al.* (2002) the equivalence rules are removed and then the second one will be removed from the set of the discovered CFD rules.

While as the third axiom FD3, will remove any variable value from the LHS of the CFD rules; this will make the discovered rules have only constant attributes on the RHS.

FD1: If $A \in X$, then $(R : X \rightarrow A,\ t_p)$, where $t_p[A_L] = t_p[A_R] =$ 'a' for some 'a' $\in \mathrm{dom}(A)$, or both are equal to a '_'.

FD2: If (1) $(R : X \rightarrow A_i, t_i)$ such that $t_i[X] = t_j[X]$ for all $i, j \in [1, k]$, (2) $(R : [A_1, \ldots, A_k] \rightarrow B, t_p)$ and moreover, (3) $(t_1[A_1], \ldots, t_k[A_k]) \preceq t_p[A_1, \ldots, A_k]$, then $(R : X \rightarrow B, t'_p)$, where $t'_p[X] = t_1[X]$ and $t'_p[B] = t_p[B]$.

FD3: If $(R : [B, X] \rightarrow A, t_p)$, $t_p[B] =$'_', and $t_p[A]$ is a constant, then $(R : X \rightarrow A, t'_p)$, where $t'_p[X \cup \{A\}] = t_p[X \cup \{A\}]$.

FD4: If (1) $\Sigma \vdash_I (R : [X, B] \rightarrow A, t_i)$ for $i \in [1, k]$, (2) $\mathrm{dom}(B) = \{b_1, \ldots, b_k, b_{k+1}, \ldots, b_m\}$, and $(\Sigma, B = b_l)$ is not consistent except for $l \in [1, k]$, and (3) for $i, j \in [1, k]$, $t_i[X] = t_j[X]$, and $t_i[B] = b_i$, then $\Sigma \vdash_I (R : [X, B] \rightarrow A, t_p)$ where $t_p[B] =$'_' and $t_p[X] = t_1[X]$.

Fig. 9: Inference rules for CFDs

And finally the fourth axiom FD4 will cause merging similar CFDs that have the same RHS and LHS and the idea of the merge illustrated above.

After applying these axioms on a set of CFDs and applying the Minimum Cover algorithm proposed by Fan *et al.* (2009), we will have a minimum set of CFDs.

We have a mixed up all the above axioms and Minimum Cover algorithm, because our approach mainly depends on the intersect partitions between the Candidates; we will use the idea of this partitions for finding the minimum set MCF of CFDs.

Partition minimal cover: Procedure shown in Fig. 10 works as follow: the procedure selects all the CFDs that have the same intersect partitions and then chooses between them the CFDs that have the same RHS and have some intersect LHS attributes between them, after that the algorithm applies the inference axioms which will produce the Minimum set of CFDs.

Illustrative case study: This section contains a real case taken from free database available on the internet to explain how CFD-Mine algorithm works.

A database called Balloon dataset from UCI (Machine Learning Repository) (Asuncion and Newman, 2007). Balloon dataset given in Table 2 consists of a set of trousers with different characteristics and has 5 attributes and 20 tuples.

First of all, AprioriGen algorithm takes the singleton elements (candidates) and SingletonCalculatePartition procedure finds all equivalence classes to the singleton attributes stored in C_1= {Color, Size, Act, Age, inflated} as shown in Table 3.

At level two the AprioriGen uses the candidates available in C_1 to finds the candidates in C_2 and the CalculatePartition finds all equivalence classes for the candidates attribute set stored in C_2 and these steps repeated until the last level in the dataset as shown in Table 4.

Next step is creating the rules and this step follows the four steps shown early, if there is an element in second level such as (Act, Age) and its partitions are:

$$\Pi_{Act,Age} = \{\{1, 6, 11, 16\}, \{2, 7, 12, 17\}, \{3, 8, 13, 18\}, \{4, 5, 9, 10, 14, 15, 19, 20\}\}$$

And one of its related elements in the third level is (Act, Age, Inflated) and its partitions are:

$$\Pi_{Act,Age,Inflated} = \{\{1, 6, 11, 16\}, \{2, 7, 12, 17\}, \{3,8,13,18\}, \{4, 5, 9, 10, 14, 15, 19, 20\}\}$$

```
PartitionMinimalCover (CF)
{
1.      for each i=1 to CF_size
2.          for each j=2 to CF_size
3.           If ((Ωx CFD[i]= Ωx CFD[j])&&
4.              (RHS CFD[i]= RHS CFD[j])&&
5.               (LHS CFD[j] ⊆ LHS CFD[i]))
6.
7.           MCF= MCF  ∪ IR (CFD);
8.return MCF;
}
```

Fig. 10: PartitionMinimalCover procedure

Table 2: Balloon relation instance

t#	Color	Size	Act	Age	Inflated
t_1	Yellow	Small	Stretch	Adult	T
t_2	Yellow	Small	Stretch	Child	T
t_3	Yellow	Small	Dip	Adult	T
t_4	Yellow	Small	Dip	Child	F
t_5	Yellow	Small	Dip	Child	F
t_6	Yellow	Large	Stretch	Adult	T
t_7	Yellow	Large	Stretch	Child	T
t_8	Yellow	Large	Dip	Adult	T
t_9	Yellow	Large	Dip	Child	F
t_{10}	Yellow	Large	Dip	Child	F
t_{11}	Purple	Small	Stretch	Adult	T
t_{12}	Purple	Small	Stretch	Child	T
t_{13}	Purple	Small	Dip	Adult	T
t_{14}	Purple	Small	Dip	Child	F
t_{15}	Purple	Small	Dip	Child	F
t_{16}	Purple	Large	Stretch	Adult	T
t_{17}	Purple	Large	Stretch	Child	T
t_{18}	Purple	Large	Dip	Adult	T
t_{19}	Purple	Large	Dip	Child	F
t_{20}	Purple	Large	Dip	Child	F

Table 3: Partitions of the Candidates in C_1

Attr name	Partitions	Cardinality
Π_{Color} =	{{1,2,3,4,5,6,7,8,9,10}, {11,12,13,14,15,16,17,18,19.20}}.	$\lvert\Pi_{Color}\rvert$ =2
Π_{Size} =	{{1,2,3,4,5,11,12,13,14,15}, {6,7,8,9,10,16,17,18,19.20}}.	$\lvert\Pi_{Size}\rvert$ = 2
Π_{Act} =	{{1,2,6,7,11,12,16,17}, {3,4,5,8,9,10,13,14,15,18,19.20}}.	$\lvert\Pi_{Act}\rvert$ = 2
Π_{Age} =	{{1,3,6,8,11,13,16,18}, {2,4,5,7,9,10,12,14,15,17,19,20}}.	$\lvert\Pi_{Age}\rvert$ = 2
$\Pi_{Inflated}$ =	{{1,2,3,6,7,8,11,12,13,16,17,18}, {4,5,9,10,14,15,19,20}}.	$\lvert\Pi_{Inflated}\rvert$ = 2

And these two groups of partitions are identical so there is a FD between them and the discovered FD is:

- FD = (Act, Age → Inflated)

Now If we have two candidates (Act, Inflated) and (Act, Age, Inflated) and there is a shared partition (Ox) between them Ox = {{3, 8, 13, 18}, {4, 5, 9, 10, 14, 15, 19, 20}} which is discovered

Table 4: Partitions of the Candidates in C_2

Attr name	Partitions	Cardinality
$\Pi_{Color,Size}$ =	{{1,2,3,4,5},{6,7,8,9,10}, {11,12,13,14,15},{16,17,18,19.20}}.	$\lvert\Pi_{Color,Size}\rvert = 4$
$\Pi_{Color,Act}$ =	{{1,2,6,7},{3,4,5,8,9,10}, {11,12,16,17},{13,14,15,18,19.20}}.	$\lvert\Pi_{Color,Act}\rvert = 4$
$\Pi_{Color,Age}$ =	{{1,3,6,8},{2,4,5,7,9,10}, {11,13,16,18},{12,14,15,17,19,20}}.	$\lvert\Pi_{Color,Age}\rvert = 4$
$\Pi_{Color,Inflated}$ =	{{1,2,3,6,7,8},{5,9,10}, {11,12,13,16,17,18},{14,15,19,20}}.	$\lvert\Pi_{Color,Inflated}\rvert = 4$
$\Pi_{Size,Act}$ =	{{1,2,11,12},{3,4,5,13,14,15}, {6,7,16,17},{8,9,10,18,19,20}}.	$\lvert\Pi_{Size,Act}\rvert = 4$
$\Pi_{Size,Age}$ =	{{1,3,11,13},{2,4,5,12,14,15}, {6,8,16,18},{7,9,10,17,19,20}}.	$\lvert\Pi_{Size,Age}\rvert = 4$
$\Pi_{Size,Inflated}$ =	{{1,2,3,11,12,13},{4,5,14,15}, {6,7,8,16,17,18},{9,10,19,20}}.	$\lvert\Pi_{Size,Inflated}\rvert = 4$
$\Pi_{Act,Age}$ =	{{1,6,11,16},{2,7,12,17},{3,8,13,18}, {4,5,9,10,14,15,19,20}}.	$\lvert\Pi_{Act,Age}\rvert = 4$
$\Pi_{Act,Inflated}$ =	{{1,2,6,7,11,12,16,17},{3,8,13,18}, {4,5,9,10,14,15,19,20}}.	$\lvert\Pi_{Act,Inflated}\rvert = 3$

by IntersectPartitions procedure, Then CreateCFD procedure checks the element in the LHS (Act) and (Inflated) to see which one of them has the same partition from its partitions, in this case (Act) has the same common partition but (Inflated) does not, then the (Act) is the condition portion and the (Inflated) is the variable portion and we repeat this operation on the RHS but we check only the element that is not in the LHS (i.e., Age) and we find that it doesn't have the same partition so it's a variable portion:

- φ: (Act = DIP, Inflated) → (Age)

Suppose CreateCFD procedure creates two CFD rules such as:

- φ1: (Act = "STRETCH", Color = "YALLOW") → (Inflated = "T"), Produced by Ωx = {4, 5, 9, 10} partition
- φ2: (Act = "STRETCH", Color = "PRUPLE") → (Inflated = "T"), produced by Ωx = {14, 15, 19, 20} partition

In our approach you will not see it because we merge it into a single CFD using Merging Similar CFDs pruning algorithm.

- φ: (Act = STRETCH, Color) → (Inflated = T)

EXPERIMENTAL EVALUATION

The utility of the approach: We mentioned earlier that the FD used mainly for schema design purpose and CFD found for cleaning the data relations from erroneous entering, but we can't ignore the important role that the FD approaches have been playing in data cleaning too, so as our approach discovers both minimum set of Conditional Functional Dependencies which may differ from another set of CFD discovered by another approach but they are equivalent and set of Functional Dependences, we can-in future-design a complete system for cleaning the relation based on both FD and CFD .

Almost all of the relations located on the Asuncion and Newman (2007) have inconsistencies; this leads the other approaches for discovering the FD to use what we called Approximate Functional dependency (AFD) which is Functional dependency that almost holds. Our approach sometimes can't find any Functional Dependences in the relation, because the relation has some errors.

Let us think differently. If we apply the discovering of CFD Rules and then modify the data relation according to these Rules, the relation will not have any inconsistencies; this manner will produce a set of real Functional Dependences FD, not Approximate Functional Dependencies (AFD).

So, CFD-Mine function for cleaning data will be finding the most rules agreed on the relation and then modify the spurious tuples which disagree with the discovered rules and help the data entries to insert correct entities which agree with the rules later on.

Accuracy of discovered rules: Golab *et al.* (2008) present a definition to the problem of optimal pattern tableau generation (CFD) based on natural criteria, it might seem that a good tableau should choose patterns to maximize the number of tuples, they think that a good tableau should apply to at least some minimal subset of the data and should allow some of the tuples to cause violation. They present two main variables called support (tuples should match) and confidence (tuples should violate).

Because our approach discovers all possible CFDs, this may seem conflict to what the authors (Golab *et al.*, 2008) come in, but we deal with this criterion in a different manner; we put a variable called threshold r which presents the percentage of the tuples that the discovered CFD rules covered, this value has two main benefits:

- It lets the algorithm produce only the rules that have this value and above
- It reduces the search time for finding the Rules

If the user identifies threshold r, then the application program will filter the discovered rules according to this threshold, all the rules above the value of threshold are support and all the rules under the value of threshold are confidence.

Present algorithm for finding the CFDs based on finding the intersect partitions between the Candidates and each partition has its own cardinality (the number of tuples in that partition) and because we need to increase the speed of the search for finding CFDs, we identify an equation to connect the percentage of the discovered CFDs with the cardinality of the partitions.

For instance, if we have a relation with 400 tuples and we need to find only the CFDs that agree only on 20% and above of the relation dataset, so we set the value of the threshold on this equation:

$$g = (\text{The ratio } r \times \text{No. of tuples in relation } t)/100$$

$$g = (20 \times 400)/100 = 80 \text{ (Tuple/Partition)}$$

Now, the value of g means that only partitions that have the cardinality equal 80 and above are added to the create rules phase while the other partitions are removed, so they will not be included in the search space.

Suppose that we want to find only the CFDs that covered 40% of the relation, the value of the threshold r = 40. The number of the tuples in the relation t = 20:

$$g = (r \times t)/100 = (40 \times 20)/100 = 8$$

So, only the partitions that have 8 and above number of tuples in the partition are included in the phase of producing CFDs; if there are rules covering this percentage.

For instance:

- Π_{Act} = {1, 2, 6, 7, 11, 12, 16, 17}, {3, 4, 5, 8, 9, 10, 13, 14, 15, 18, 19, 20}
- $\Pi_{Act, Inflated}$ = {1, 2, 6, 7, 11, 12, 16, 17}, {3, 8, 13, 18}, {4, 5, 9, 10, 14, 15, 19, 20}
- Intersect Partition Ωx = {1,2,6,7,11,12,16,17}
- Cardinality of Ωx = | Ωx | = 8

The discovered CFD is:

- φ: (Act = "STRETCH") → (Inflated=" T")

Now any other tuples between the candidates (Act) and (Act, Inflated) don't agree on one of these rules are Confidence, while as any tuples agree on these rules are Support.

If you need to discover the FDs that the dataset set holds in, you have to set the value of the threshold r = 0, to prevent the algorithm delete any partition with any size, because the mechanism of finding FD as TANE propose is to compare the partition set of two candidate, if its identical then there is a functional dependency between them.

Scalability experiments

Parameters: CFD-MINE was applied on a datasets obtained from the UCI Machine Learning Repository (Asuncion and Newman, 2007). Our experiments were run using a Dual T2350 INTEL Processor 1.86 Ghz (1.86 Ghz) with 3 GB of memory; we used the Adult dataset and Agaricus-lepiota dataset and varied the parameter of interest to test its effect on the discovery running time.

Scalability on the number of tuples: For the purpose of study the behavior of our algorithm when increasing the number of tuples we examined by fixing the number of attribute a = 8 and giving three different values to the threshold r, the values are r = 1, 2 and 3 and starting the number of tuples from t = 1 k to t = 8 k.

When increasing the number of tuples, as shown in Fig. 11 and 12, the algorithm behave semi Linearly, To explain this phenomena, we mentioned that our algorithm mainly divided into two main complexity issues.

The first one is finding the partition of the candidates and because we didn't change the number of candidates in all of the cases (2^9 = 512 candidates), then there is no added time to find it, but the little increase of time is because finding more number of partition to each candidate which mean more time for find the intersect partitions and more time to generate the CFDs rules.

And the second one is the attribute size, because the attribute size of the Adult dataset is larger than the attribute size of the agaricus-lepiota and because we deal with the data as String data type, then when the size of String increased the time for merging and separation and other operations on the string done, the time is also increased. So, the time for the same number of tuples and attribute for Adult data set as shown in Fig. 11 is larger than the time for agaricus-lepiota as shown in the Fig. 12 but the algorithm still behaves linearly.

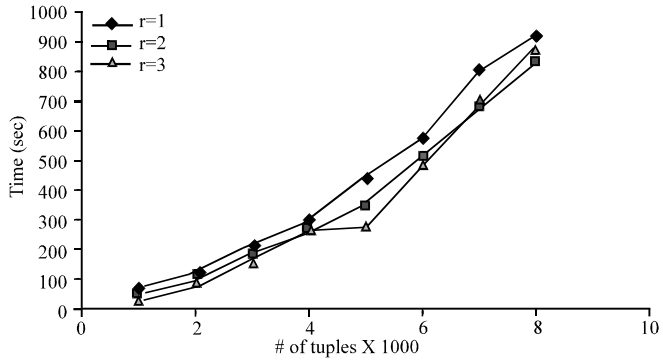

Fig. 11: Scalability per tuples (Adult)

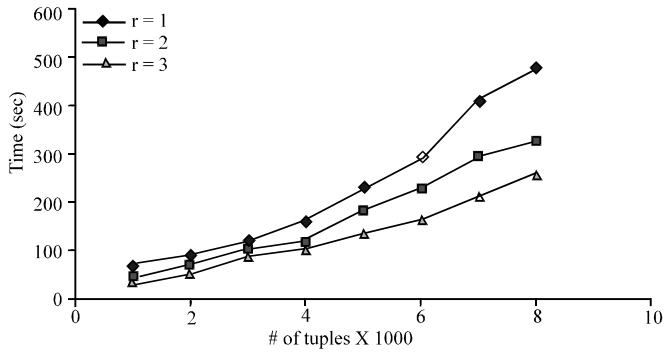

Fig. 12: Scalability per tuples (Agaricus-lepiota)

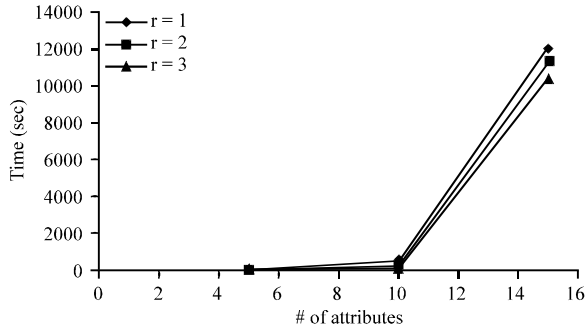

Fig. 13: Scalability per attributes (Adult)

Scalability on the number of attributes: For the purpose of study the behavior of our algorithm when increasing the number of attributes we examined by fixing the number of tuples t = 1 k and giving three different values to the threshold r, the values are r = 1, 2 and 3 and starting the number of attributes from a = 5 to a = 15.

When increasing the number of attributes, as shown Fig. 13 and 14, the algorithm behaves Exponentially, to explain this phenomena, we explain that the time that the algorithm need it to find the partition for dataset with 3 attribute ($2^3 = 8$ candidates) is much less than the time needed to calculate the partitions for dataset with 15 attributes ($2^{15} = 32768$ candidates).

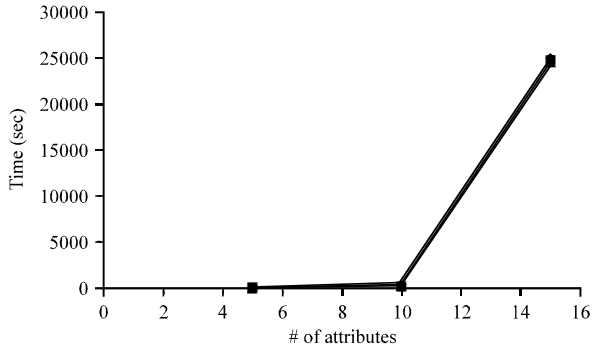

Fig. 14: Scalability per attributes (Agaricus-lepiota)

These results are identical to that of TANE, because the idea of calculating the partition presented in TANE and TANE is one of the most efficient approaches for finding the traditional functional dependencies.

If we assume that we deal directly with existing partitions and we want to only discover the rules the time that will be taken is very small, but the time for finding the partitions is large, so, when increasing the number of attributes the number of candidates increases too.

CONCLUSIONS

In this study we present a new approach to the discovery of both Functional and Conditional Functional Dependencies. The major innovations is a novel way of discovering all possible Rules and determining minimal set on these Rules and use the modified inference Axioms for CFDs, The idea is to maintain information about which rows agree on a set of attributes. Formally, the approach can be described using equivalence classes and partitions. A major advantage of the use of partitions is that it allows efficient discovery of conditional dependencies and traditional dependencies.

The algorithm is based on the levelwise search algorithm that has been used in many data mining applications. It starts from dependencies with a small left-hand side, i.e., from the ones that are not very likely to hold. The algorithm then works towards larger and larger dependencies, until the minimal dependencies that hold are found.

The worst case time complexity of the algorithm with respect to the number of attributes is exponential, but this is inevitable since the number of minimal dependencies can be exponential in the number of attributes. However, if the number of rows increases, the set of dependencies remains the same, the time increases only linearly in the number of rows.

The linearity makes the algorithm especially suitable for relations with large number of rows. Experimental results show that the algorithmic effective in practice and that it makes the discovery of functional and conditional functional dependencies feasible for relations with even hundreds of thousands of rows.

REFERENCES

Al-Nabhan, M., S. Yousef and J. Al-Saraireh, 2006. TCP protocol and red gateway supporting the QoS of multimedia transmission over wireless networks. Inform. Technol. J., 5: 689-697.

Arenas, M., L.E. Bertossi and J. Chomicki, 2003. Consistent query answers in inconsistent databases. Theory Pract. Logic Progra., 3: 393-424.

Asuncion, A. and D. Newman, 2007. UCI machine learning repository of machine learning databases. University of California, School of Information and Computer Science, Irvine, CA. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Chiang, F. and R. Miller, 2008. Discovering data quality rules. Proceedings of the VLDB Endowment, in International Conference on Very Large Data Bases, August, 2008, University of Toronto, Toronto, Canada, pp: 1166-1177.

English, L., 2000. Plain English on data quality: Information quality management: The next frontier// DM review magazine. Prieiga Internet.

Fan, W., F. Geerts, L. Lakshmanan and M. Xiong, 2009. Discovering conditional functional dependencies. Proceedings of the IEEE 25th International Conference on Data Engineering, March 29-April 2, 2009, Shanghai, pp: 1231-1234.

Golab, L., H.J. Karloff, F. Korn, D. Srivastava and B. Yu, 2008. On generating near-optimal tableaux for conditional functional dependencies. PVLDB, 1: 376-390.

Mannila, H. and H. Toivonen, 1997. Levelwise search and borders of theories in knowledge discovery. Data Min. Knowl. Discovery, 1: 241-258.

Pramada, J., J.V. Rao and D.V.S.R. Anil Kumar, 2012. Characterization of boolean valued star and mega lattice functions. Asian J. Algebra, 5: 1-10.

Qi, J.J. and L. Wei, 2008. Attribute reduction in consistent decision formal context. Inform. Technol. J., 7: 170-174.

Rahm, E. and H.H. Do, 2000. Data cleaning: Problems and current approaches IEEE Data Eng. Bull., 23: 1-11.

Raoul, M. and N. Lhouari, 2008. Conditional functional dependencies discovery. Research Report LIMOS/RR-08-13, http://limos.isima.fr/IMG/pdf/RR-08-13.pdf

Renuga, R. and S. Sadasivam, 2009. Data discovery in grid using content based searching technique. Inform. Technol. J., 8: 71-76.

Sagiroglu, O. and M. Ozturan, 2006. Implementation difficulties of hospital information systems. Inform. Technol. J., 5: 892-899.

Sahai, A., M.M. Rahman and R.P. Jaju, 2005. Using meta-analysis for data enrichment-optimal families of estimation strategies. J. Applied Sci., 5: 1575-1581.

Sharma, S., A. Tiwari, S. Sharma and K.R. Pardasani, 2007. Design of algorithm for frequent pattern discovery using lattice approach. Asian J. Inform. Manage., 1: 11-18.

Shilakes, C.C. and J. Tylman, 1998. Enterprise Information Portals. Merrill Lynch Inc., New York, USA.

Thakur, R.S., R.C. Jain and K.R. Pardasani, 2007. Fast algorithm for mining multi-level association rules in large databases. Asian J. Inform. Manage., 1: 19-26.

Wang, X.L., Z. Jing and H.Z. Yang, 2011. Service selection constraint model and optimization algorithm for web service composition. Inform. Technol. J., 10: 1024-1030.

Wei, M., A.H. Sung and M.E. Cather, 2007. FROut: A novel approach to linking large databases. Inform. Technol. J., 6: 37-47.

Yao, H., H.J. Hamilton and C.J. Butz, 2002. FD Mine: Discovering functional dependencies in a database using equivalences. Proceedings of the IEEE International Conference on Data Mining, December 9-12, 2002, University of Saskatchewan, Canda, pp: 729-732-732.