LETTER

An Optimization Strategy for CFDMiner: An Algorithm of Discovering Constant Conditional Functional Dependencies

Jinling ZHOU[†a)], **Xingchun DIAO**[††], **Jianjun CAO**[††], ***Nonmembers*, *and* Zhisong PAN**[†b)], ***Member***

SUMMARY Compared to the traditional functional dependency (FD), the extended conditional functional dependency (CFD) has shown greater potential for detecting and repairing inconsistent data. CFDMiner is a widely used algorithm for mining constant-CFDs. But the search space of CFDMiner is too large, and there is still room for efficiency improvement. In this paper, an efficient pruning strategy is proposed to optimize the algorithm by reducing the search space. Both theoretical analysis and experiments have proved the optimized algorithm can produce the consistent results as the original CFDMiner.

key words: *Data Quality, conditional functional dependency, free itemset, closed itemset, frequent itemset*

1. Introduction

Traditional functional dependency (FD) is proposed to guarantee the data consistency in business information systems. However, FD is not enough to fully reflect the consistency in data, and its form is too limited to express various dependency rules [1]. In recent years, the conditional functional dependency (CFD) extends from FD by adding the constant patterns reflecting the semantics in data, which has shown greater potential for detecting and repairing inconsistent data [2], [3].

FDs or CFDs are usually set up by domain experts through manual work. However, such an artificial approach cannot meet the demands of the Data Quality Management due to the increase of database scale and the improvement of real-time requirements. Algorithms for auto-discovering dependency rules from data are essential to check the data consistency.

CFDMiner is proposed by Wenfei Fan et al, which is the most popular algorithm for discovering constant CFDs. CFDMiner is more efficient than other similar algorithms [1]. Because it discovers constant CFDs from the *free* itemsets and *closed* itemsets, which are two classes of specific *frequent* itemsets. Even so, there is still room for efficiency improvement of CFDMiner. Past studies focus on just generating effective candidate *free* and *closed* itemsets to reduce the search time for CFDs [4]. However, this method is limited to improve the efficiency of CFDMiner, for even generating all *free* and *closed* itemsets is fast enough in some efficient algorithms, such as GcGrowth and so on [5]–[7].

In this paper, a more efficient optimization strategy (pruning on *free* itemsets) are proposed to reduce the search space of CFDMiner and improve its computational efficiency. Firstly, it is proved in theory that the consistent results can be generated after reasonable pruning of CFDMiner. Experiments show that the optimized algorithm has a smaller search space and less search time.

2. Discovering Constant CFDs

Consider a relation R over a set of attributes, denoted by $\mathbf{Attr}(R) = \{A_1, A_2, \cdots A_m\}$. For each attribute $A_i \in \mathbf{Attr}(R)$, $i = 1, 2, \cdots m$, we use $\mathbf{Dom}(A_i)$ to denote its domain. Let $t[A_i]$ be the projection of the tuple t on attribute A_i.

Definition 1 (CFDs) A *conditional functional dependency* (CFD) φ over R is a pair $(X \to A, t_p)$ [8], where (1) X is a set of attributes in $\mathbf{Attr}(R)$, and A is a single attribute in $\mathbf{Attr}(R)$, (2) $X \to A$ is a standard *functional dependency* (FD), referred to as the FD embedded in φ, and (3) t_p is a pattern tuple with attributes in X and A, where for each B in $X \cup \{A\}$, $t_p[B]$ is either a constant value in $\mathbf{Dom}(B)$, or an unnamed variable '_' that draws values from $\mathbf{Dom}(B)$.

We denote X as LHS(φ) and A as RHS(φ). The X and A attributes in a pattern tuple is separated with '‖'. Given an instance I over a relation R, a CFD φ is satisfied by the instance I, denoted by $I \vDash \varphi$.

Definition 2 (CCFDs) A CFD is a *constant conditional functional dependency* (CCFD) if its pattern tuple t_p consists of constants only.

Examples Here are some CFDs that hold in Table 1.

$\varphi_0 : ([CC, ZIP] \to STR, (44, \_ \parallel \_))$

Table 1 An example instance of the customer relation from [8]. NM stands for name, PN for phone number, CC for country code, AC for area code, STR for street, CT for city, and ZIP for zip code.

	NM	PN	CC	AC	STR	CT	ZIP
t_1	Mike	11111	01	908	Tree Ave	MH	07974
t_2	Rick	11111	01	908	Tree Ave	MH	07974
t_3	Joe	22222	01	212	5th Ave	NYC	01202
t_4	Jim	22222	01	908	Elm Str	MH	07974
t_5	Ben	33333	44	131	High St	EDI	EH4
t_6	Ian	44444	44	131	High St	EDI	EH4
t_7	Ian	44444	44	908	Port PI	MH	W1B
t_8	Sean	22222	01	131	3rd Str	UN	01202

Manuscript received August 3, 2015.
Manuscript revised October 12, 2015.
Manuscript publicized November 6, 2015.

†The authors are with College of Command Information Systems, PLA University of Science and Technology, Nanjing, 210007, China.
††The authors are with PLA University of Science and Technology, Nanjing, 210007, China.
a) E-mail: kinringchow@yahoo.com
b) E-mail: hotpzs@hotmail.com

DOI: 10.1587/transinf.2015EDL8170

$\varphi_1 : ([CC, AC] \rightarrow CT, (01, 908 \parallel \text{MH}))$
$\varphi_2 : ([CC, AC] \rightarrow CT, (44, 131 \parallel \text{EDI}))$
$\varphi_3 : ([CC, AC] \rightarrow CT, (44, 131 \parallel \text{NYC}))$

CFDs specify the specific cases of an FD in a dataset or some conditions where FD holds in parts of a dataset. $\varphi_1 \sim \varphi_3$ are CCFDs, and φ_0 is a *variable* CFD (VCFD).

This paper mainly focus on CCFDs. For a more detailed discussion of CFDs, refer to [2], [8], [9].

Definition 3 (Non-trivial, non-redundant, k-frequent CCFDs [9]) For a CCFD: $\varphi = (X \rightarrow A, t_p)$: (1) if $A \notin X$, then φ is non-trivial, otherwise it is trivial; (2) if whenever $I \vDash \varphi$, and $I \nvDash (Y \rightarrow A, (t_p[Y] \parallel t_p[A]))$ for any proper subset $Y \subset X$, then φ is non-redundant, or it is redundant; (3) All the tuples matching the CCFD φ in I constitute a set, denoted by $\text{supp}(\varphi, I)$. If the number of tuples in the set $|\text{supp}(\varphi, I)| \geqslant k$, then φ is a k-frequent CCFD.

Definition 4 (Minimal set of CCFDs) A set of CCFDs Σ is said to be minimal if $\forall\varphi \in \Sigma$ and φ is a non-trivial, non-redundant, k-frequent CCFD.

Definition 5 (Canonical cover of CCFDs) If Σ is a minimal set of CCFDs and Σ covers all the k-frequent CCFDs in I, then Σ is the (k-frequent) canonical cover of CCFDs.

Definition 6 (The discovery of CCFDs) The discovery of CCFDs is to discover the (k-frequent) canonical cover of CCFDs in an instance I.

The first and most popular algorithm for discovering CCFDs, CFDMiner is shown in **Algorithm** 1, refer to [2], [8] and [9] for more details. It discovers CCFDs based on the cover of *free* and *closed* itemsets, which are two specific kinds of *frequent* itemsets. The definition of *free* and *closed* itemsets will be given in the next section, for they are also very important concepts for the following optimization strategy.

3. Pruning Stategy and Optimized CFDMiner

Free itemsets and *closed* itemsets are two important concepts for CFDMiner and our optimization (pruning) strategy. To make it easier to understand CFDMiner and follow the upcoming optimization, definitions of *free* and *closed* itemsets are given.

Definition 7 (Itemsets and support) An itemset is a pair (X, t_p), where $X \subseteq \textbf{Attr}(R)$ and t_p is a constant pattern over X. Given an instance I of R, we use notation of *supports*, and denote by $\text{supp}\big((X, t_p), I\big)$ the support of (X, t_p) in I, i.e., the set of tuples in I that matches t_p on the X-attributes.

The concept of "itemset" here is in keeping with the "*frequent* itemsets" in "Association Rules Mining". In fact, the CCFDs are a kind of special association rules (with 100% confidence).

Definition 8 (*Free* and *closed* itemsets) Given (X, t_p) and (Y, s_p), we say that (Y, s_p) is more general than (X, t_p) ((X, t_p) is more specific than (Y, s_p)), denoted by $(X, t_p) \prec (Y, s_p)$, or $(Y, s_p) \succ (X, t_p)$, if $Y \subset X$ and $s_p = t_p[Y]$. Obviously, $\text{supp}\big((Y, s_p), I\big) \supseteq \text{supp}\big((X, t_p), I\big)$. (1) (X, t_p) is called *free* if $\nexists(Y, s_p) \succ (X, t_p)$ and $\text{supp}\big((Y, s_p), I\big) = \text{supp}\big((X, t_p), I\big)$; (2) (X, t_p) is called *closed* if $\nexists(Z, u_p) \prec (X, t_p)$ and $\text{supp}\big((Z, u_p), I\big) = \text{supp}\big((X, t_p), I\big)$; (3) If there exists a *free* itemset (Y, s_p) and a *closed* itemset (Z, u_p), and $(Y, s_p) \succ (Z, u_p)$, $\text{supp}\big((Y, s_p), I\big) = \text{supp}\big((Z, u_p), I\big)$, then (Z, u_p) is the unique *closed* itemset that extends (Y, s_p), denoted by $\text{clo}(Y, t_p) = (Z, u_p)$.

Free itemset is sometimes called "generator" and *closed* itemset is called "closure" in other publications [7], [10]. CFDMiner discovers constant CFDs not directly from the data, but from the cover of the *free* and *closed* itemsets.

Algorithm 1 CFDMiner

Input: An Instance I of R and a natural number $k \geqslant 1$.
Output: A canonical cover of k-frequent CCFDs..

Compute a mapping C2F that associates with each k-frequent closed itemset in I its set of k-frequent free itemsets (using GcGrowth [5]);
for all k-frequent closed itemset (Y, s_p) in I **do**
  Let L be the list of all free itemset in C2F;
  for all $(X, t_p) \in L$ **do**
    Initialize $\text{RHS}(X, t_p) = (Y \setminus X, s_p[Y \setminus X])$
  end for
  for all $(X, t_p) \in L$ do
    for all $(X', t_p[X']) \in L$ such that $X' \subset X$ **do**
      $\text{RHS}(X, t_p) = \text{RHS}(X, t_p) \setminus \text{RHS}\ (X', t_p[X'])$
    end for
    if$(\text{RHS}(X, t_p) \neq \emptyset)$
      Output $(X \rightarrow A, t_p[X] \parallel a)$ for all $(A, a) \in \text{RHS}(X, t_p)$
    end if
  end for
end for

Before the effective pruning strategy is given to optimize CFDMiner, some lemmas should be proved in advance to show that the strategy will not change the CCFDs output.

Lemma 1 For a *free* itemset (X, t_p) and the *closed* itemset $\text{clo}(X, t_p) = (Y, s_p)$, if there exists a super set $X' \supseteq X$, (X', t'_p), then its *closed* itemset $\text{clo}(X', t'_p) = (Y', s'_p)$, $s'_p[Y' \setminus X'] \supseteq s_p[Y \setminus X]$.

Proof. Assume $s'_p[Y' \setminus X'] \not\supseteq s_p[Y \setminus X]$, we distinguish between 2 cases: (1) $\exists A \in Y \setminus X$ and $A \notin Y' \setminus X'$, $s_p[A] \notin s'_p[Y']$; (2) $\exists A \in Y \setminus X$ and $A \in Y' \setminus X'$, $s_p[A] \neq s'_p[A]$.

For case (1), we proceed as follows: Since $X' \supseteq X$ and $\text{clo}(X', t'_p) = (Y', s'_p)$, then we have $\text{supp}(Y', s'_p) = \text{supp}(X', t'_p) \subseteq \text{supp}(X, t_p) = \text{supp}(Y, s_p)$. That is, $\text{supp}(Y', s'_p) \subseteq \text{supp}(Y, s_p)$, then for $\forall t \in \text{supp}(Y', s'_p)$, $t \in \text{supp}(Y, s_p)$. Since $\exists A \in Y \setminus X$ and $A \notin Y' \setminus X'$, $s_p[A] \notin s'_p[Y']$, then for $\forall t \in \text{supp}(Y', s'_p)$, $t \in \text{supp}((Y' \cup A), (s'_p, s_p[A]))$. That is, $\text{supp}(Y', s'_p) \subseteq \text{supp}((Y' \cup A), (s'_p, s_p[A]))$.

Always $\text{supp}(Y', s'_p) \supseteq \text{supp}((Y' \cup A), (s'_p, s_p[A]))$, then we have $\text{supp}(Y', s'_p) = \text{supp}((Y' \cup A), (s'_p, s_p[A])) = \text{supp}(X', t'_p)$. $((Y' \cup A), (s'_p, s_p[A]))$ is a super set of (Y', s'_p) having the same support as (X', t'_p), this contradicts (the definition of *closed* itemset) "(Y', s'_p) is the unique *closed* itemset of (X', t'_p)". So case (1) is invalid.

For case (2), we proceed as follows: Since $X' \supseteq X$ and $\text{clo}(X', t'_p) = (Y', s'_p)$, then we have $\text{supp}(Y', s'_p) = \text{supp}(X', t'_p) \subseteq \text{supp}(X, t_p) = \text{supp}(Y, s_p)$. That is, $\text{supp}(Y', s'_p) \subseteq \text{supp}(Y, s_p)$, then for $\forall t \in \text{supp}(Y', s'_p)$, $t \in \text{supp}(Y, s_p)$. But $s_p[A] \neq s'_p[A]$, then $t_{s_p}[A] \neq t_{s'_p}[A]$.

Either side of the inequality is the same tuple over different patterns. So case (2) will not happen.

As a conclusion, Lemma 1 is proved to be correct. □

Definition 9 ($-p^{level}$ subset) A *free* itemset $(Y, t_p[Y]), Y \subset X$ is called a $-p^{level}$ subset of $(X, t_p[X]), |X| = n$ if the number of attributes $|Y| = n - p, (0 < p < n)$.

All the $-p^{level}$ subsets of $(X, t_p[X])$ is denoted by $\text{sub}^{-p}(X, t_p[X])$. These subsets can be sorted, or processed in the order they appear. The j^{th} subset is denoted by $\text{sub}^{-p}(X, t_p[X])_j, j \in \{1, 2, \cdots, C_n^{n-p}\}$.

For example, all -2^{level} subsets of (a, b, c, d) : $\text{sub}^{-2}(a, b, c, d) = \{(a, b)(a, c)(a, d)(b, c)(b, d)(c, d)\}$; the 2nd and 3rd subset in the -2^{level} subsets: $\text{sub}^{-2}(a, b, c, d)_2 = (a, c)$, $\text{sub}^{-2}(a, b, c, d)_3 = (a, d)$. Obviously, the union set of -1^{level} subset for all sets in sub^{-k} is just $\text{sub}^{-(k+1)}$, that is:

$$\text{sub}^{-(k+1)}(X, t_p) = \bigcup_{j=1}^{C_n^{n-k}} \text{sub}^{-1}(\text{sub}^{-k}(X, t_p)_j) \qquad (1)$$

For example, $\text{sub}^{-3}(a, b, c, d) = \bigcup_{j=1}^{C_4^2} \text{sub}^{-1}(\text{sub}^{-2}(a, b, c, d)_j) = \{(a)(b)(c)(d)\}$.

Lemma 2 All non-empty proper subsets of a *free* itemset are *free*.

Proof. This lemma has been proved in [7] (Propsition 2). The Lemma shows that if an algorithm (e.g. GcGrowth) can mining all the *free* itemsets in a database, then any non-empty proper subset x of each *free* itemset will appear in the output, for x is also a *free* itemset.

In the following, all "subsets" given in this paper are in terms of non-empty proper subsets.

Algorithm 2 prCFDMiner

Input: An Instance I of R and a natural number $k \geqslant 1$.
Output: A canonical cover of k-frequent CCFDs..

1: Compute a HashMap C2F $\langle (X, t_p), (Y, s_p) \rangle$ that associates with each k-frequent *closed* itemset in I its set of k-frequent *free* itemsets (using algorithm GcGrowth [5]);
2: **for all** k-frequent *closed* itemset (Y, s_p) in I **do**
3: Let L be the list of all *free* itemset in C2F;
4: **for all** $(X, t_p) \in L$ **do**
5: Initialize $\text{RHS}(X, t_p) = (Y \setminus X, s_p[Y \setminus X])$
6: **end for**
7: **for all** $(X, t_p) \in L$ **do**
8: **for all** $\text{sub}^{-1}(X, t_p)_j \subseteq L, j \in \{1, 2, \cdots, C_{|X|}^{|X|-1}\}$ **do** (**Strategy**)
9: $\text{RHS}(X, t_p) = \text{RHS}(X, t_p) \setminus \text{RHS}(\text{sub}^{-1}(X, t_p)_j)$
10: **end for**
11: **if**$(\text{RHS}(X, t_p) \neq \emptyset)$
12: Output $(X \rightarrow A, t_p[X] \parallel a)$ for all $(A, a) \in \text{RHS}(X, t_p)$
13: **end if**
14: **end for**
15: **end for**

Deduction 1 For a *free* itemset $(X, t_p[X])$ and its -1^{level} sub *free* itemsets $\text{sub}^{-1}(X, t_p[X])$, all the corresponding *closed* itemsets to each itemset in the -1^{level} sub *free* itemsets $\text{sub}^{-1}(X, t_p[X])$ constitute a set, denoted by $\text{clo}(\text{sub}^{-1}(X, t_p[X])) = \bigcup_{j=1}^{C_n^{n-1}} \text{clo}(\text{sub}^{-1}(X, t_p[X])_j)$. Thus, we have $\text{clo}(\text{sub}^{-1}(X, t_p[X])) \supseteq \bigcup_{k=1}^{n-1} \text{clo}(\text{sub}^{-k}(X, t_p[X]))$.

Proof. $\because$ According to Lemma 1, $s'_p[Y' \setminus X'] \supseteq s_p[Y \setminus X]$ where $(X', t'_p) \supseteq (X, t_p), \text{clo}(X', t'_p) = (Y', s'_p), \text{clo}(X, t_p) = (Y, s_p)$.
$\therefore \{s'_p[Y' \setminus X'] \cup s'_p[X']\} \supseteq \{s_p[Y \setminus X] \cup s_p[X]\}$, that is, $s'_p[Y'] \supseteq s_p[Y]$.

That's to say, we have $\text{clo}(X', t'_p) \supseteq \text{clo}(X, t_p)$ if (X, t_p) is a subset of the *free* itemset (X', t'_p). Thus,
$\because \forall k \in \{1, 2, \cdots, n-1\}, \text{sub}^{-1}(X, t_p[X]) \supseteq \text{sub}^{-k}(X, t_p[X])$
$\therefore \text{clo}(\text{sub}^{-1}(X, t_p[X])) \supseteq \text{clo}(\text{sub}^{-k}(X, t_p[X]))$.
$\Longrightarrow \text{clo}(\text{sub}^{-1}(X, t_p[X])) \supseteq \{\text{clo}(\text{sub}^{-1}(X, t_p[X])) \cup \text{clo}(\text{sub}^{-2}(X, t_p[X]) \cup \cdots \cup \text{clo}(\text{sub}^{-(n-1)}(X, t_p[X]))\}$. That is, $\text{clo}(\text{sub}^{-1}(X, t_p[X])) \supseteq \{\bigcup_{k=1}^{n-1} \text{clo}(\text{sub}^{-k}(X, t_p[X]))\}$. □

Strategy In Line 8 of CFDMiner, just search the -1^{level} subsets of the *free* itemsets instead of all subsets.

Proof. According to the proof for Deduction 1, $\forall (Z, s_p) \in \text{sub}^{-k}(X, t_p[X]), \exists (Z', s'_p) \in \text{sub}^{-1}(X, t_p[X]), s.t. \text{clo}(Z', s'_p) \supseteq \text{clo}(Z, s_p)$, which means the corresponding *closed* itemsets to all the -1^{level} subsets of a *free* itemset will cover all the elements of the *closed* itemsets to all subsets.

Then the remaining question is to make sure that the all -1^{level} subsets for each *free* itemset will appear in the output of GcGrowth (the actual input of CFDMiner). According to Lemma 2, the input of CFDMiner will also cover the all -1^{level} subsets for each *free* itemset. Therefore, it is not necessary to search the whole but just the -1^{level} subsets of each free itemset. □

According to the above strategy, we optimized CFDMiner as the Algorithm 2. The major modification (shaded part) is to search -1^{level} subsets of a *free* itemset instead of all the subsets.

4. Time Complexity Analysis

The search space is largely narrowed, from $n(2^l - 2)$ to nl, where l is the average length of free itemsets and n is the number of *free* itemsets. Therefore, the time complexity ratio of optimized prCFDMiner to original CFDMiner is $l : (2^l - 2)$. Theoretically, the efficiency will improve 100 times when the average length l of free itemsets for a relational database is 10. But in reality, the l will not be a big value, the actual efficiency will improve about 5–6 times.

5. Experiments

Our experiments used 3 real datasets from UCI machine learning repository (http://archive.ics.uci.edu/ml/), namely Adult, Mushroom and Chess. Table 2 lists the parameters of the datasets and the number of pairs for the *free* and *closed* itemsets from the datasets (which is the actual input of the algorithms). 3 numeric attributes in Adult dataset has been removed to adapt to the algorithms.

Experiments have proved that the optimized algorithm can output the consistent CCFDs as the original CFDMiner. The number of CCFDs output is shown in Table 3.

Table 2 The paremeters of datasets and input pairs of itemsets under different support settings

Dataset	Adult	Mushroom	Chess
Arity	12	23	7
Size	32561	8124	28056
The input pairs of *free* and *closed* itemsets			
Support (%)	Adult	Mushroom	Chess
30	72	558	5
10	725	7631	47
5	2360	21160	122
1	24260	103517	1210
0.5	57869	164526	2997

Table 3 The number of output CCFDs under different support settings

Support (%)	Adult	Mushroom	Chess
30	2	95	0
10	7	2774	0
5	27	7510	2
1	288	26029	7
0.5	692	37040	15

Table 4 Comparison of Optimized and Original CFDMiner

on Adult				
Performance	Search Space		Execution Time (ms)	
Support (%)	CFDMiner	prCFDMiner	CFDMiner	prCFDMiner
30	358	165	94	13
10	9362	2439	172	43
5	41130	8819	277	72
1	739858	108786	1697	553
0.5	2235142	277469	4369	1783
on Mushroom				
Performance	Search Space		Execution Time (ms)	
Support (%)	CFDMiner	prCFDMiner	CFDMiner	prCFDMiner
30	5584	1754	163	37
10	140418	29889	575	214
5	466446	88269	1445	613
1	3482502	486990	9539	3854
0.5	6346824	802401	20352	7821
on Chess				
Performance	Search Space		Execution Time (ms)	
Support (%)	CFDMiner	prCFDMiner	CFDMiner	prCFDMiner
30	0	0	75	1
10	14	14	87	8
5	144	144	98	20
1	3664	2650	196	49
0.5	14450	7937	290	96

To show improved performance of the strategy, we test the search space and execution time of original CFDMiner and optimized prCFDMiner on the 3 datasets. The search space refers to the number of the whole non-empty proper subsets for all the free itemsets. Experiment results are shown in Table 4.

With decreased support going with increasing pairs of *free* and *closed* itemsets, the search space will become larger. In the same support, the optimized algorithm search significantly smaller space than the original CFDMiner.

For example, on Mushroom dataset, the search space of CFDMiner is about 7.1 (3482502/486990) times of prCFDMiner under the support of 1%, and the relative execution time is about 2.5 (9539/3854) times, for average length of *free* itemsets is 4–5 (the theoretical multiple is from $(2^4 - 2)/4 = 3.5$ to $(2^5 - 2)/5 = 6$). Another example, on Chess, the search space is about 1.8 (14450/7937) times under the support of 0.5%, and the relative execution time is about 3.0 (290/96) times, for average length of *free* itemsets is 2-3 (the theoretical multiple is from $(2^2 - 2)/2 = 1$ to $(2^3 - 2)/3 = 2$).

The attributes on Mushroom is more than that on Chess, but the average length of free itemsets mined from datasets will not rise sharply by the increasing of attributes. Thus, the average length will not be a big value, and the actual efficiency on different datasets will not improve much greater times (e.g. above-mentioned 100 times in the theory).

6. Conclusion

In this paper, we have given a more efficient pruning strategy for optimizing CFDMiner, a very popular algorithm of discovering CCFDs. We proved in the theory that the pruning strategy will not influence the output of the original algorithm, and we evaluated the optimized algorithm on real datasets. Experiments show that the proposed optimization has much smaller search space and higher efficiency.

Acknowledgements

The author are grateful to Limsoon Wong and Liu Guimei for supporting the correct version of GcGrowth, which is such a key algorithm for our research.

References

[1] D. Thierno, N. Noel, and P. Jean-Marc, "Discovering (Frequent) Constant Conditional Functional Dependencies," Int. J. Data Mining, Modelling and Management, vol.4, no.3, pp.205–223, 2012.
[2] W. Fan, F. Geerts, X. Jia, et al., "Conditional Functional Dependencies for Captruing Data Inconsistencies," ACM Trans. Database Systems, vol.33, pp.1–48, 2008.
[3] W. Fan, "Dependencies revisited for improving data quality," Proc. of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp.159–170, 2008.
[4] J. Li, J. Liu, H. Toivonen, and J. Yong, "Effective Pruning for the Discovery of Conditional Functional Dependencies," The Computer Journal, vol.56, no.3, pp.378–392, 2013.
[5] H. Li, J. Li, L. Wong, M. Feng, and Y.-P. Tan, "Relative Risk and Odds Ratio: A Data Mining Perspective," Proc. twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '05, pp.368–377, 2005.
[6] T. Calders and B. Goethals, "Non-derivable Itemset Mining," Data Mining and Knowledge Discovery, vol.14, no.1, pp.171–206, 2007.
[7] Jinyan Li, Haiquan Li, Limsoon Wong, and et al., Minimum Description Length Principle: Generators are Preferable to Closed Patterns, Proc. 21st National Conf. on AAAI: 409-414, 2006.
[8] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering Conditional Functional Dependencies," IEEE Transactions on Knowledge & Data Engineering, vol.23, no.5, pp.683–698, 2011.
[9] W. Fan and F. Geerts, "Foundations of Data Quality Management," Synthesis Lectures on Data Management, vol.4, no.5, pp.1–217, 2012.
[10] A. Tran, T. Truong, and B. Le, "Simultaneous Mining of Frequent Closed Itemsets and Their Generators: Foundation and Algorithm," Engineering Applications of Artificial Intelligence, vol.36, pp.64–80, 2014.