Effective Pruning for the Discovery of Conditional Functional Dependencies

Jiuyong Li, Jiuxue Liu
School of Computer and Information Science, University of South Australia, Australia
Hannu Toivonen
Department of Computer Science, University of Helsinki, Finland
Jianming Yong
School of Information Systems, University of Southern Queensland

Abstract

Conditional Functional Dependencies (CFDs) have been proposed as a new type of semantic rules extended from traditional functional dependencies. They have shown great potential for detecting and repairing inconsistent data. Constant CFDs are 100% confidence association rules. The theoretical search space for the minimal set of CFDs is the set of minimal generators and their closures in data. This search space has been used in the currently most efficient constant CFD discovery algorithm. In this paper, we propose pruning criteria to further prune the theoretic search space, and design a fast algorithm for constant CFD discovery. We evaluate the proposed algorithm on a number of medium to large real world data sets. The proposed algorithm is faster than the currently most efficient constant CFD discovery algorithm, and has linear time performance in the size of a data set.

Keywords: Functional dependencies, conditional functional dependencies, association rules, closed patterns.

1 Introduction

Poor data quality has been a major problem in many organisations. Erroneous and inconsistent data has costed US business hundreds of billions of dollars because of poor business decisions resulting from the poor data quality [1]. Recently, conditional functional dependencies (CFDs) have shown great potential for detecting and repairing inconsistent data in relational data sets [2, 3]. For example, the following CFD was discovered in US airline traffic data in our experiment. "((Origin State Name, Distance Group → Destination State Name), Hawaii, 1 || Hawaii). It means that if the Origin State is Hawaii and the Distance Group is in 1 (e.g. less than 1000 km), then the Destination must be Hawaii. Normally, there is no functional dependency (FD) between attributes (Origin State, Distance Group) and attribute Destination State. In other words, two flights departing from the same state and flying within the same distance group (e.g. less than 1000 km), they may fly to different destination states. However, when the original state is Hawaii and the distance group is 1, the destination state is dependent on the values of two attributes. In this example, the destination state has to be Hawaii. This is a conditional functional dependency where the value dependency only holds for some specific attribute values (not for all values).

Conditional Functional Dependencies are designed for the detection and repairing of inconsistencies of data. For example, suppose that the above CFD has been found in January data. In February, we found a record with the following information (Origin State Name = Hawaii, Distance Group=1, Destination State Name = Washington). We have very strong reason to suspect an erroneous value in the record. Or if a CFD can be formed when very few records are disregarded, then these records may contain erroneous values. It is important for an organisation to have a complete set of integrity constraints that reflect the organisation's policies and domain semantics to improve and maintain the data quality of the organisation. CFD discovery helps an organisation to build a set of such constraints.

CFDs can also be used in database design to enforce the semantical constraints for maintaining high quality databases. However, most CFDs are to be obtained from databases since they could not be identified as functional dependencies at the database design stage. Firstly, they represent local constraints that domain experts are not aware of. Secondly, some CFDs are formed in daily practices and hence they can only be revealed from data. Therefore, the discovery of CFDs is the first and very important step for applying CFDs to data quality enforcement.

The study on the discovery of CFDs has just started. A few algorithms have been published in the last few years. An algorithm using an attribute lattice to generate candidate embedded FDs is published in [1]. A greedy approximation algorithm is proposed in [4] to compute a close-to-optimal tableau for a CFD when the embedded FD is given. Three algorithms called CTANE, CFDMiner, and FastCFD were proposed in [5]. The rule discovery based method, CFDMiner, has been shown to be several orders of magnitude faster than two other functional dependency based methods. One major reason is that the rule discovery based method makes use of the frequency and closure pruning strategies developed in rule discovery [6, 7].

Constant CFD discovery is a special case of association rule discovery [5]. In the light of rule discovery research, a theoretical search space for the minimal set of CFDs is the set of minimal generators and their closures in data [8, 9, 7]. This search space has been used by CFDMiner [5], the currently most efficient constant CFD discovery algorithm. In this paper, we will show that the discovery of minimal set of CFDs does not need all minimal generators and closures. We will propose new criteria to further prune this search space. We will then propose a fast algorithm for the discovery of minimal set of CFDs. The algorithm is evaluated on some real world data sets and has been shown to be faster than CFDMiner [5].

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

Table 1: An example data set from [5]. Each row specifies phone details of a customer. CC stands for country code, AC for area code, PN for (phone number), NM for (name), STR for street, CT for city, and ZIP for zip code.

2 Conditional Function Dependencies and CFD discovery

Assume a table R over a set of attributes $\{A_1, A_2, \ldots, A_m\}$. Let dom$(A)$ be a set of values (or tuples) of attribute (or attribute set) A. Let $t_i[A]$ be the projection of tuples t_i on attribute (or set) A. In this paper, a capital letter stands for a set or an attribute, and a lower case letter for an attribute value. We call a tuple over the set of all attributes $\{A_1, A_2, \ldots, A_m\}$ a record.

Definition 1 (Functional dependency (FD))**.** *An FD over R is represented as $X \rightarrow Z$, where (1) X is a set of attributes and Z is a single attribute and $Z \not\subset X$, and (2) $\forall i, j$, if $t_i[X] = t_j[X]$, then $t_i[Z] = t_j[Z]$.*

For example, the following two FDs hold in Table 1.

$$\begin{aligned} f1: &\quad [CC, AC] \rightarrow CT \\ f2: &\quad [CC, AC, PN] \rightarrow STR \end{aligned}$$

$f1$ requires that two customers with the same country and area codes also have the same city code; $f2$ requires that two customers with the same country and area codes and the same phone number also have the same street address.

We only need to consider a single attribute on the right hand side (RHS) since $X \rightarrow YZ$ is equivalent to $X \rightarrow Y$ and $X \rightarrow Z$.

Definition 2 (Conditional functional dependency (CFD) [5])**.** *A CFD ϕ over R is a pair $(X \rightarrow Z, t_p)$, where (1) $X \rightarrow Z$ is a standard FD, referred to as* *the FD embedded in ϕ. (2) t_p is a pattern tuple with attributes in X and Z. For each attribute set $B \subset X \cup Z$, $t_p[B]$ is either a constant in* dom(B) *or an uninstantiated (or unnamed) variable '_' that stands for any value of* dom(B). *Tuple t in R matches the pattern tuple t_p in ϕ if for any attribute set $B \subset X \cup Z$, $t[B] = t_p[B]$ or $t_p[B] =$ '_' (t agrees with t_p on all instantiated (or named) attributes of t_p.) Relation R satisfies ϕ if $\forall$ i, j, $t_i[X] = t_j[X]$ and both match $t_p[X]$, then $t_i[Z] = t_j[Z]$ and both match $t_p[Z]$. Let* $\text{LHS}(t_p) = t_p[X]$ *and* $\text{RHS}(t_p) = t_p[Z]$.

The following are some CFDs that hold in Table 1. || is the divider of LHS (Left Hand Side) and RHS (Right Hand Side).

$$\begin{aligned}
\phi_0 : &\quad ([CC, ZIP] \rightarrow STR, (44, \_||\_))\\
\phi_1 : &\quad ([CC, AC] \rightarrow CT, (01, 908||MH))\\
\phi_2 : &\quad ([CC, AC] \rightarrow CT, (44, 131||EDI))\\
\phi_3 : &\quad ([CC, AC] \rightarrow CT, (01, 212||NYC))
\end{aligned}$$

CFDs specify 1) the specific cases of an FD in a data set, and/or 2) some conditions where FD holds in parts of a data set. For example, CFDs ϕ_1, ϕ_2, ϕ_3 specify special cases of FD f_1. FD $[CC, ZIP] \rightarrow STR$ does not hold in data set 1, but holds for the part of the data set where $CC = 44$. This has been summarised as a CFD ϕ_0. An FD can be considered as a special case of CFD when the tuple pattern contains only the unnamed variable. For example, FD $f1 : [CC, AC] \rightarrow CT$ can be represented as a CFD $f1 : ([CC, AC] \rightarrow CT, (\_, \_||\_))$.

Definition 3 (Support of CFDs)**.** *The support of a CFD is the fraction of records in a data set that satisfies the CFD.*

For example, the support of ϕ_1 is 3/8 since three tuples in the example data set satisfy ϕ_1. Given a minimum support requirement, a CFD is *frequent* if its support is at least as large as the minimum support.

The major purpose of CFDs is for data quality enforcement. CFDs should be applicable for future cases. Many low support CFDs do not generalize and they are not of interest for data quality enforcement. For example, we may have CFDs ((ID → Name), 10011110 || John), and ((ID → Name), 10011111 || Smith). They explain the current data set but have no use for future data quality enforcement. Furthermore, many low support CFDs may be results of random matches, and they have no use for quality enforcement. Therefore, we are interested in frequent CFDs.

Definition 4 (Constant CFDs [5])**.** *A CFD is a constant CFD if its pattern tuple t_p consists of constants only. Specifically, for a constant CFD $(X \rightarrow Z, t_p)$, $t_p[Z]$ is a constant and for all $B \in X$, $t_p[B]$ is a constant.*

For example, ϕ_1, ϕ_2, and ϕ_3 are constant CFDs.

If a CFD is not constant, it is *variable* [2]. A variable CFD represents a set of constant CFDs in finite domains. For example, the variable CFD $(|CC, ZIP| \rightarrow STR, (44, \_||\_))$ in Table 1 can be specialized as $\{(|CC, ZIP| \rightarrow STR, (44, 908||\text{Port PI}))$, and $(|CC, ZIP| \rightarrow STR, (44, 131||\text{High St}))\}$.

In this paper we only consider constant CFDs for the following three reasons. First, constant CFDs represent general CFDs at the value level and are necessary to the satisfaction check of general CFDs. Second, all variable CFDs can be upgraded from constant CFDs for finite domains [2]. Third, in data quality control or assessment, constant CFDs will be used for consistency checking [1]

Definition 5 (The relationship of constant CFDs)**.** *Let t_p and t_q be two tuple patterns of constant CFDs. t_p is more general than t_q if* $\text{LHS}(t_p) \subset \text{LHS}(t_q)$ *and* $\text{RHS}(t_p) = \text{RHS}(t_q)$. *Equally, t_q is more specific than t_p. This relationship is denoted as $t_p \ll t_q$.*

For example, $(|CC, ZIP| \rightarrow STR, (44, 131||\text{High St}))$ is more general than $(|CC, ZIP, \text{Phone Number}| \rightarrow STR, (44, 131, 33333||\text{High St}))$. The second CFD is implied by the first one and hence is redundant. Whenever a tuple satisfies the second CFD, it satisfies the first CFD too. Furthermore, the second CFD explains at most the same number of tuples as the first CFD. In work [5], a non-redundant CFD is called left reduced CFD.

Definition 6. *[Minimal set of constant CFDs] A set of CFDs is said to be minimal if no tuple pattern in the set is more specific than another one. The* *minimal set of CFDs includes non-redundant CFDs only.*

For example, let a set of CFDs be $\{(|CC, ZIP| \rightarrow STR, (44, 131||\text{High St})), (|CC, ZIP, \text{Phone Number}| \rightarrow STR, (44, 131, 33333 \mid\mid \text{High St})), (|CC, ZIP, \text{Phone Number}| \rightarrow STR, (44, 131, 44444 \mid\mid \text{High St})), (|CC, ZIP| \rightarrow STR, (44, 908 \mid\mid \text{Port PI}))\}$. The minimal set of CFDs is $\{(|CC, ZIP| \rightarrow STR, (44, 131 \mid\mid \text{High St})), (|CC, ZIP| \rightarrow STR, (44, 908 \mid\mid \text{Port PI}))\}$.

In terms of consistency check, the minimal set of CFDs is as good as the complete set of CFDs. An inconsistency in a record is identified when the record contains the LHS of a CFD but does not contain its RHS. Note that a redundant CFD, say r_2, contains extra values in the LHS with respect to a more general CFD, say r_1, in the minimal set. All inconsistencies discovered by r_2 are discovered by r_1 too. Therefore, the set excluding r_2 does not lose any capability for consistency check.

Definition 7 (The problem of constant CFD discovery)**.** *CFD discovery is to discover the minimal set of frequent constant CFDs.*

In the following discussions, CFDs are constant CFDs since we do not consider variable CFDs.

3 A theoretical search space for constant CFDs

Constant CFD discovery has a close relationship with association rule discovery. We firstly discuss their relationship, and then discuss the search space for constant CFDs.

We firstly introduce notions in association rule discovery. A tuple can be equivalently represented as a set of attribute value pairs, called a *pattern*. Given a scheme (Name, Postcode, Phone Number), a tuple is represented as (John Smith, 077144, 12345678). The order is important since values have to match attributes. However, the tuple can be represented as a set (Name = John Smith, Postcode = 077144, Phone Number= 12345678). In this representation, the order is insignificant since values are associated with their attributes. The tuple representation has been used in FD discovery, and the set representation has been used in rule discovery. A pattern is a sub (or super) pattern of another pattern if its set of attribute value pairs is a sub (or super) set of the set of attribute value pairs of another pattern. We use set representation in the following discussions.

Definition 8 (Association rule [10])**.** *Let p be a pattern and z be an attribute value pair where $z \notin p$. $p \rightarrow z$ is called an implication. The support of p in relation R is the fraction of records in R containing p, the support of z the fraction of records in R containing z, and the support of $p \rightarrow z$ or pz (a shorthand for $p \cup z$) the fraction of records containing both p and z in R. The confidence of $p \rightarrow z$ is the ratio of the support of pz to the support of p. Implication $p \rightarrow z$ is an association rule if its support s and confidence c are at least as large as the minimum support and confidence, denoted as an (s, c) association rule. Confidence and support will be denoted as* conf *and* supp.

For example, (CC = 01, AC = 908) → CT = MH is an association rule with the support of 3/8 and confidence of 1. We say that it is a (0.375, 1) association rule.

Observation 1. *The pattern tuple t_p of a constant CFD $(X \rightarrow Z, t_p)$ is equivalent to an $(\epsilon, 1)$ association rule where ϵ is a small positive number.*

The above observation has been made in [5]. We provide a proof in the following for the self-containment of this paper.

Proof. The pattern tuple t_p occurs at least once in the data set, and hence has some support $\epsilon > 0$. If any tuple in the data set contains $LHS(t_p)$, it contains $RHS(t_p)$. Therefore, $\text{supp}(LHS(t_p)) = \text{supp}(LHS(t_p) \cup RHS(t_p))$ and $\text{conf}(LHS(t_p) \rightarrow RHS(t_p)) = 1$. Rule $LHS(t_p) \rightarrow RHS(t_p)$ is an $(\epsilon, 1)$ association rule.

Consider now an $(\epsilon, 1)$ association rule $p \rightarrow z$. Since it has support $\epsilon > 0$, it occurred at least once in the data set. Since the confidence is 1, any tuple containing p must contain z too. Therefore, $p \rightarrow z$ is a pattern tuple. □

The confidence of a CFD is the same as that of its corresponding rule because of their equivalence.

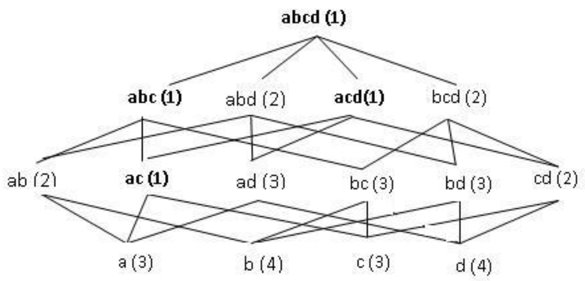

Figure 1: The search lattice of Example 1. The support count is listed with a pattern. One group of generators and their closure are highlighted in bold.

For example, $\phi_2 : ([CC, AC] \rightarrow CT, (44, 131||EDI))$ is a (0.25, 1) association rule, $(CC = 44, AC = 131) \rightarrow CT = EDI$.

The next question is how to discover $(\epsilon, 1)$ association rules. This discovery is closely related to closure operator [8, 11].

Consider a data set R and a pattern p. The closure of p is its longest super pattern $p_c \supset p$ such that $\text{supp}(p_c) = \text{supp}(p)$. p_c is also called the closed pattern of p and p is called a generator of p_c. In other words, a closure is the pattern which does not have a super pattern with the same support. If p_m is the shortest pattern that maps to the closure p_c, p_m is called a minimal generator. In other words, the minimal generator is a pattern which does not have a sub pattern with the same support. We give an example to illustrate the concepts.

Example 1. *Consider the following data set. For simplicity, we only list attribute values that we are interested in.*

A	B	C	D
	b	c	d
a			d
a	b		d
a	b	c	d
	b	c	

The search lattice for all patterns in the table is illustrated in Figure 1. The support is listed with each pattern. Patterns ac, abc, acd have the same closure as abcd. abc, acd and ac are its generators and ac is the minimal generator. In the data set, other minimal generators and their closures are $\{(a, ad), (c, bc), (ab, abd), (cd, bcd)\}$.

We will use minimal generators and closures for efficient CFD discovery. Firstly, we will discuss the search space for CFD discovery.

Definition 9 (Search space for constant CFDs)**.** *Given a data set, all attribute values and their combinations form a lattice, which is the complete search space. The search space is the set of nodes in the lattice to be traversed in order to discover a set of constant CFDs.*

Normally, the size of the lattice is very large even for a data set with only a few attributes. For example, consider a data set of 16 attributes, each of which contains 4 values. When we search for CFDs with up to 6 attribute values in the LHSs and 1 attribute value in each RHS, the size of the lattice (the search space without pruning) is 2^{42} (see the last paragraph of Section 5.4 for details.). A large search space makes the discovery of CFDs challenging.

Forward pruning can be used to make the search space smaller by predicting nodes that do not contain CFDs and by removing these nodes from the search space. The infrequent nodes can be pruned following the Apriori principle for frequent pattern mining [6]. All descendant nodes (patterns) of infrequent ones are infrequent and hence are pruned. The pruned search space is significantly smaller than the unpruned search space.

The search for a minimal generators and their closures is another well defined search task [9, 11] and it is closely related to the non-redundant rule (CFD) discovery [5]. We do not use all information in the search space for determining CFDs but only closures and their minimal generators. For example, nodes a, b, d, ab, ad, bd, and abd in Figure 1 are nodes in the search space, but only nodes a, ad, ab and abd are used for determining CFDs. The four nodes form a search space for CFDs in this example. We will provide detailed discussions in the following.

The following result has been discussed in previous work [7, 8, 9]. A generator and its closure will form a 100% confidence association rule (CFD). For example, generator ab and its closure abd form a 100% confidence rule $ab \rightarrow d$ because of $\text{supp}(ab) = \text{supp}(abd)$ in Figure 1.

Observation 2. *All $(\epsilon, 1)$ rules can be derived from generators and their corresponding closures.*

Proof. Let p be a generator and p_c be its closure. $\text{conf}(p \rightarrow p_c \backslash p) = 1$ where $\backslash$ is the set difference operator. $p \rightarrow p_c \backslash p$ is an $(\epsilon, 1)$ rule.

If p_c is not the closure for generator q, then $\text{supp}(q) \neq \text{supp}(p_c)$ and there is no possibility to form an $(\epsilon, 1)$ rule between them. □

For example, $(\epsilon, 1)$ association rules in Example 1 include $abc \rightarrow d$, $acd \rightarrow b$, and $ac \rightarrow bd$ (equivalently, $ac \rightarrow b$ and $ac \rightarrow d$). Therefore, a search space for constant CFDs is the set of generators and closures. For simplicity of presentation, we use rules to represent CFDs in this and following sections.

However, we also note the redundant rules between a non-minimal generator and its closure. For example, rule $acd \rightarrow b$ is a redundant rule with respect to rule $ac \rightarrow b$. As a result, the search space for a minimal set of CFDs is further refined as the following.

Observation 3. *The search space for the minimal set of CFDs is the set of minimal generators and their closures.*

Example 2. *Following Example 1, many closed pattern (and minimal generator) mining algorithms, such as [12, 13, 7, 14, 15, 16] can find the set of minimal generators and their closures in the data set. All minimal generators, their closures and CFDs are listed in the following table.*

minimal generators	*closures*	*CFDs*
a	ad	$a \rightarrow d$
c	bc	$c \rightarrow b$
ab	abd	$ab \rightarrow d$
cd	bcd	$cd \rightarrow b$
ac	$abcd$	$ac \rightarrow b$, $ac \rightarrow d$

Based on Definition 6, the final minimal set of constant CFDs includes $a \rightarrow d$ and $c \rightarrow b$.

Previous work [5] makes use of this search space to generate the minimal set of CFDs. All five closures are to be searched to generate the minimal set of CFDs. However, we will demonstrate that this search space can be further pruned. In the above example, we do not search for the closures in the last three rows, i.e. abd, bcd and $abcd$, and we do not miss any CFDs in the the minimal set of CFDs.

4 Further pruning the theoretical search space

In this section, we will show that it is not necessary to use all minimal generators and their closures for the discovery of the minimal set of constant CFDs. CFDs from some minimal generators will not be in the minimal set of CFDs and can be pruned. For easy understanding, we should show how closures abd, bcd and $abcd$ in Example 2 are pruned step by step by presenting pruning criteria. We then give an example to show how the criteria work together to reduce the search space.

The objective of pruning is to remove nodes from the search space. The removal of one node effectively removes all its descendant nodes from the search space in a branch and bound search. Therefore, we should be sure that no eligible CFDs will be derived from a node or any of its descendant nodes before we remove the node. Formally, a node in the search lattice is prunable based on two conditions. Firstly, there are no CFDs that can be produced from the node to be included in the minimal set of CFDs. Secondly, there are no CFDs that can be produced from all descendant nodes of the node to be included in the minimal set of CFDs.

We first present notations for CFD candidates in the search lattice. Let pair $(abd, \{a, b, d\})$ stand for the node abd in the lattice of Figure 1 and the set of RHSs of potential CFDs in the node and its descendant nodes. In node $(abd, \{a, b, d\})$, three potential CFDs are $bd \rightarrow a$, $ad \rightarrow b$, and $ab \rightarrow d$. Formally, a CFD candidate is represented as a pair (pattern, RHS set). The first part pattern indicates a node in the search lattice (when we refer to the search lattice, it is easy to understand a candidate as a node. However, in the algorithm, we do not use the lattice as the search space. Instead, we directly deal with patterns. Therefore, we use a pattern rather than a node here.). The RHS set is a set of RHSs of potential CFDs in the node and its descendant nodes. The removal of one attribute value pair x in the RHS set indicates the removal all potential CFDs in the node and all its descendant nodes with the LHS of x. For example, $(abd, \{a, d\})$ indicates two CFDs $bd \rightarrow a$ and $ab \rightarrow d$. Since value b is removed from the RHS set of node abd, the value b will be absent from RHS sets of all descendant nodes of node abc too. For example, the candidate in node $abcd$ will be $(abcd, \{a, c, d\})$. There will be no CFD $acd \rightarrow b$.

When we reference the candidates in the search lattice, we call them descendant nodes and ancestor nodes. When we do not reference them in the search lattice, we call them sub candidates and super candidates. Formally, (P_1, T_1) is a sub candidate of (P_2, T_2) if $P_1 \subset P_2$. Equivalently, (P_2, T_2) is a super candidate of (P_1, T_1).

The ideas of pruning are outlined as the following. We firstly prune attribute value pairs in the RHS set of a node. Using candidate $(abd, \{a, b, d\})$ as an example, if b is pruned from the RHS set, the candidate becomes $(abd, \{a, d\})$. This means that candidate CFD $ad \rightarrow b$ and all its more specific candidate CFDs are not in the minimal set of CFDs. In the following discussions, we use $adQ \rightarrow b$ for any Q to represent all more specific CFDs of $ad \rightarrow b$. Secondly, we consider pruning the candidate when all attribute values in its RHS set are pruned, for example, $(abd, \emptyset)$. Once a candidate is pruned, all its super candidates will not be generated. We need to be sure that no CFDs will be in the minimal set of CTDs from all super candidates of a pruned candidate. For example, we prune candidate $(abd, \emptyset)$ only if we know that CFDs from candidate $(abdX, Y)$ for any X and $Y \subseteq X$ will not be in the minimal set of CFDs. We present two RHS set pruning criteria, and two candidate pruning criteria in the following.

Criterion 1 (RHS set pruning 1). *Assume a CFD candidate (Pz, T). If there is a sub candidate (P, T_1) such that $\text{supp}(P) = \text{supp}(Pz)$, then z will not be in the RHS sets of all super candidates of (Pz, T) and hence z can be pruned from RHS set T.*

Proof. $P \rightarrow z$ is a CFD because of $\text{supp}(P) = \text{supp}(Pz)$. Since $P \rightarrow z$ is a CFD, all its more specific candidate CFDs will be CFDs too. However, those more specific CFDs should be excluded from the minimal set of constant CFDs by the definition. □

In Example 1, candidate $(abd, \{a, b, d\})$ is pruned to $(abd, \{a, b\})$ since $\text{supp}(ab) = \text{supp}(abd)$. candidate $(bcd, \{b, c, d\})$ is pruned to $(bcd, \{c, d\})$ since $\text{supp}(bcd) = \text{supp}(cd)$.

We then look at another RHS set pruning criterion. Let $\neg z$ be any value that is not z. $\text{supp}(P\neg z)$ means the fraction of records containing P but not z. That is $\text{supp}(P\neg z) = \text{supp}(P) - \text{supp}(Pz)$.

Criterion 2 (RHS set pruning 2). *Assume a CFD candidate (Pxz, T). If there is a sub candidate (Px, T_1) such that $\text{supp}(P\neg z) = \text{supp}(Px\neg z)$, then z will not be in RHS sets of all super candidates of (Pxz, T) and hence z can be removed from RHS set T.*

Proof. The main point is that if $PQx \rightarrow z$ is a CFD, then $PQ \rightarrow z$ must be a CFD. So, $PQx \rightarrow z$ is not in the minimal set of CFDs. We have the following deductions.

$$\text{supp}(P\neg z) = \text{supp}(Px\neg z) \Longrightarrow$$
$$\text{supp}(PQ\neg z) = \text{supp}(PxQ\neg z).$$

$$\begin{aligned}
&\text{conf}(PxQ \rightarrow z) = \text{supp}(PxQz)/\,\text{supp}(PxQ)\\
&= \text{supp}(PxQz)/(\text{supp}(PxQz) + \text{supp}(PxQ\neg z))\\
&= \text{supp}(PxQz)/(\text{supp}(PxQz) + \text{supp}(PQ\neg z))\\
&\leq \text{supp}(PQz)/(\text{supp}(PQz) + \text{supp}(PQ\neg z))\\
&= \text{conf}(PQ \rightarrow z)
\end{aligned}$$

Therefore, $PQx \rightarrow z$ will not be in the minimal set of CFDs since its confidence is at most the same as that of $PQ \rightarrow z$. □

Following the example before, candidate $(abd, \{a, b\})$ is pruned to $(abd, \{a\})$ since $\text{supp}(a\neg b) = \text{supp}(ad\neg b)$. Candidate $(abd, \{a\})$ is further pruned to $(abd.\emptyset)$ since $\text{supp}(d\neg a) = \text{supp}(bd\neg a)$. In the same way, candidate $(bcd, \{c, d\})$ is pruned to $(bcd, \emptyset)$ because of $\text{supp}(b\neg c) = \text{supp}(bd\neg c)$ and $\text{supp}(c\neg d) = \text{supp}(bc\neg d)$.

Now there is not a single potential CFD in candidate $(abd, \emptyset)$ since its RHS set is an empty set. This satisfies condition 1 stated in the second paragraph of this section. However, we are unable to prune candidate $(abd, \emptyset)$ since the second condition is not satisfied. For example, if candidate $(abd, \emptyset)$ is pruned, then there is no possibility to generate candidate $(abcd, \{c\})$ if we use an efficient algorithm based on forward pruning. In this case, CFD $abd \rightarrow c$ is potentially lost. A candidate has to be kept even if its RHS set is empty.

Now we discuss criteria for candidate pruning. $(S, \emptyset)$ means that there is not a CFD in candidate (node) S. However, in order to prune it from the search space we need to make sure that there is not a potential CFD in all its super candidates (descendant nodes) in the search space. The removal of a node in the search space means the removal of all its descendant nodes from the search space.

Criterion 3 (Candidate pruning 1)**.** *Candidate* $(S, \emptyset)$ *is prunable if there is subset* $U \subset S$ *such that* $\text{supp}(S) = \text{supp}(U)$*.*

Proof. Let $S = Pz$ where z is an attribute value pair and $z \notin P$. Since all values in the RHS set of pattern S are pruned by Criteria 1 and 2, we know that there are not CFDs like $PQ \rightarrow z$ for any Q. However, the candidate is not prunable because there may be CFDs like $SQ \rightarrow x$ for any Q where x is an attribute value pair and $x \notin SQ$. Now, we will show that CFDs like $SQ \rightarrow x$ are impossible in the minimal set of CFDs.

Since $\text{supp}(S) = \text{supp}(U)$ and $U \subset S$, for a CFD $SQ \rightarrow x$, there must be another CFD $UQ \rightarrow x$. Therefore, $SQ \rightarrow x$ will not be in the minimal set of CFDs □

Following the example before, candidate $(abd, \emptyset)$ is prunable since $\text{supp}(ab) = \text{supp}(abd)$. Candidate $(abcd, c)$ is pruned too because it is a super candidate of $(abd, \emptyset)$. Candidate $(bcd, \emptyset)$ is prunable since $\text{supp}(cd) = \text{supp}(bcd)$.

Up to now, three closures abd, bcd and $abcd$ in Example 2 have been pruned since they are unnecessary for the discovery of the minimal set of CFDs.

We will present another criterion for candidate pruning, which is related to the CFD candidate generation. We will discuss how candidates are generated before we are able to present the pruning criterion.

For forward pruning, the Apriori candidate generation [6] is an effective approach. It takes two steps to generate candidates: combination and pruning. In the combination process, prefix sets are used. For example, abc, abd, and abe are three sets with the prefix of ab. The last single values, c, d, and e make them distinct. Candidates are generated from the combination of two sets with the same prefix. Three candidates are generated as $abcd$, $abce$ and $abde$. In the combination process, each candidate only makes use of two subsets in the previous level. The existence of a candidate needs the existence of all its subsets in the previous level. The pruning process will prune a newly generated candidate whose any subset does not exist. For example, if bcd does not exist, then candidate $abcd$ should be pruned.

CFD candidate generation is more complicated than the Apriori candidate generation because of the RHS set. Let us look at two candidates, $(abc, \emptyset)$ and $(abd, \{a, b, d\})$. We firstly combine the two to a new candidate $(abcd, T)$ using the Apriori candidate generation to generate the pattern part of the new candidate. The next step is to determine the RHS set T. Let us firstly set $T = \{a, b, c, d\}$. Then we prune attribute value pairs from the RHS set $\{a, b, c, d\}$. Candidate $(abc, \emptyset)$ means that none of $ab \rightarrow c$, $ac \rightarrow b$, $bc \rightarrow a$ and their more specific CFDs will be in the minimal set of CFDs. Therefore, only $abc \rightarrow d$ and its more specific CFDs are potentially in the minimal set of CFDs, and the new candidate should be like $(abcd, d)$. How do we achieve this? Firstly, we expand the RHS set of each CFD candidate. For $(abc, \emptyset)$, the RHS set is expanded to $\emptyset \cup d$ since d is new to the candidate. For $(abd, \{a, b, d\})$, the RHS set is expanded to $abd \cup c$ since c is new to the candidate. Secondly, the RHS set of the new candidate takes the intersection of two expanded RHS sets. That is $T = \{d\} \cap \{a, b, c, d\}$. The new candidate is $(abcd, \{d\})$. This candidate is further pruned by its other sub candidates, such as $(abd, \{a, b\})$ and $(bcd, \{b, c, d\})$. In the pruning stage, if any sub candidate of $(abcd, \{d\})$ does not exist, candidate $(abcd, \{d\})$ should be pruned. Let us assume that two remaining sub candidates $(abd, \{a, b\})$ and $(bcd, \{b, c, d\})$ exist. Then, we further prune the RHS set of $(abcd, \{d\})$ by the remaining sub candidates. We use sub candidate $(abd, \{a, b\})$ as an example. The RHS set of $(abd, \{a, b\})$ is expanded to abc since c is new to it. The RHS set of new generated candidate $(abcd, \{d\})$ intersects with the expanded RHS set abc, and the result is the RHS set of the new generated candidate set. The new generated candidate set becomes $(abcd, \emptyset)$.

Let us look at an example to motivate the following pruning criterion. Let $(abc, \emptyset)$, $(abd, \{a, b\})$, $(abe, \{a, b\})$ be all candidates prefixed by ab in the level 3 candidate set. We have to keep candidate $(abc, \emptyset)$ since we do not know if candidates $(abcd, \{d\})$, $(abce, \{e\})$ or $(abcde, \{d, e\})$ exist. By combining the first two candidates, we have $(abcd, \emptyset)$. At this stage, we have to keep $(abcd, \emptyset)$ since we do not know if candidate $(abcde, \{e\})$ exists. By combining the first and the third candidate, we have $(abce, \emptyset)$. until now, we know that $(abcde, \{e\})$ does not exist and hence $(abc, \emptyset)$ can be pruned. The following criterion will enable us to prune $(abc, \emptyset)$ earlier.

Criterion 4 (Candidate pruning 2)**.** *Consider a CFD candidate* $(Sp, \emptyset)$. *If for every candidate* (Sq, T_q) *with the same prefix* S *there is* $q \notin T_q$*, then candidate* $(Sp, \emptyset)$ *is prunable.*

Proof. We first examine the next level candidates. All values in Sp will not be in the RHS set of new candidate (Spq, T) because of $(Sp, \emptyset)$. However, q is possible because of the expansion process in the candidate combination. q will be eliminated eventually in the intersection operation since $q \notin T_q$. We obtain $(Spq, \emptyset)$ in the next level candidate. Moreover, we obtain $(SQp, \emptyset)$ for any $Q \neq \emptyset$ in a recursive way. Therefore, no CFD can be generated from the candidate $(SQp, \emptyset)$, and $(Sp, \emptyset)$ is prunable. □

Let us re-examine the motivating example before the Criterion. Attribute values c, d, and e make candidates $(abc, \emptyset)$, $(abd, \{a, b\})$ and $(abe, \{a, b\})$ distinct. Since d is not in the RHS set of $(abd, \{a, b\})$ and e is not in the RHS set $(abe, \{a, b\})$. Based on the criterion, $(abc, \emptyset)$ is prunable and we do not search for $(abcd, \emptyset)$, $(abce, \emptyset)$ and $(abcde, \emptyset)$.

We now use another example to show the criteria are able to prune the search space without generating all minimal generators and their closures.

Example 3. *Consider the following data set. For easy illustration, we only list attribute values that we are interested in.*

A	*B*	*C*	*D*	*E*
	b	*c*	*d*	*e*
a		*c*	*d*	*e*
a	*b*		*d*	*e*
a	*b*	*c*		*e*
a	*b*	*c*	*d*	

There is not a single CFD in the table, but this has only been found out after the following closures (minimal generators) have been examined. No closed pattern mining methods can reduce the following search space.

all closures (or minimal generators); [support]
a, b, c, d, e; [4/5]
ab, ac, ad, ae, bc, bd, be, cd, ce, de; [3/5]
abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde; [2/5]
abcd, abce, abde, acde, bcde; [1/5]

All level 3 candidates are pruned by Criteria 2 and 4 and no level 4 candidates are examined. We use candidate $(abc, \{a, b, c\})$ *as an example for pruning. Since* $\mathrm{supp}(a\neg c) = \mathrm{supp}(ab\neg c)$*, value* c *is removed from the RHS set and the candidate becomes* $(abc, \{a, b\})$*. Similarly, values* a *and* b *are removed from the RHS set because of* $\mathrm{supp}(b\neg a) = \mathrm{supp}(bc\neg a)$ *and* $\mathrm{supp}(a\neg b) = \mathrm{supp}(ac\neg b)$*. The candidate becomes* $(abc, \emptyset)$*. In the same way, we have candidates* $(abd, \emptyset)$ *and* $(abe, \emptyset)$*. Based on Criterion 4, candidate* $(abc, \emptyset)$ *is pruned. All level 3 candidates are pruned in the same way. No level 4 candidates are generated and examined.*

In this section, we have demonstrated that it is unnecessary to use all minimal generators and their closures to generate the minimal set of CFDs. The search space for the minimal set of CFDs can be pruned by the four proposed Criteria. In the next section, we will present an algorithm that makes use of the four criteria.

We have shown that the search space by using the four pruning criteria is smaller than that for the closed patterns and minimal generators. This work follows the idea of rule based pruning (in contrast to pattern based pruning used in closed pattern discovery), and Criterion 2 is a typical optimality pruning criterion [17, 18]. The major distinction is that RHSs in the optimal rule discovery are fixed to the class attribute and RHSs of CFDs in this paper are not fixed and they can be any attribute values. When the RHSs are not fixed, we will need to predict whether any attribute values can be RHSs of all descendant nodes of the current node (or whether the RHS set of the current node is permanently empty.). This constitutes the major complexity in the above discussions and also in the implementation in the next Section.

5 Algorithm of CFD discovery

In this section, we present an algorithm for fast CFD discovery using the pruning criteria presented in the previous section. These criteria can be used in conjunction with frequency pruning [6, 9, 19]. The closure pruning [9] is implied by Criteria 1 and 3 and need not be employed separately.

5.1 Candidate representation

We store attribute value pairs in lexicographic order. We do not use attribute information explicitly. However, no node in the search lattice will be formed by attribute value pairs from the same attribute. This will be enforced in the candidate generation of this algorithm. For example, if a_1, a_2 are two values from the same attribute. $\text{supp}(a_1, a_2) = 0$, and pattern (a_1, a_2) is pruned in the second level of candidate generation, and all its super patterns will not appear in the search lattice.

Using notations in the previous section, a candidate is a pair (pattern, RHS set), denoted by (P, T). The RHS set T is a set of attribute value pairs that are possible RHSs of CFDs. RHS set T is a sub or equal set of P and candidate (P, T) represents a number of potential CFDs. For example, candidate $(abc, \{a, b, c\})$ indicates three potential CFDs $ab \to c$, $ac \to b$ and $bc \to a$, and candidate $(abc, \{a, b\})$ indicates two potential CFDs $ac \to b$ and $bc \to a$. The removal of attribute value pairs from RHS set T is determined by Criteria 1 and 2 as discussed in the previous section. Candidate $(P, \emptyset)$ is a legal candidate, and it can be pruned only when it satisfies Criteria 3 or 4.

5.2 Candidate generator

The algorithm is based on the branch and bound search. For easy understanding and comparison, we present Candidate generator in a similar way as the Apriori candidate generation [6], the most famous branch and bound algorithm for rule discovery. We call a candidate l-candidate if its pattern contains l attribute value pairs. An l-candidate set includes all l-candidates. In the following discussions, we assume that attribute value pairs in a pattern are stored in a lexicographic order to avoid generating duplicate candidates like $(abc, \{a\})$ and $(cba, \{a\})$. $S_{l+1} \backslash S_l$ means the set difference of S_{l+1} and S_l.

Function 1. *Candidate generator*

```
1:  for each pair of candidates $(S_{l-1}p, T_p)$ and $(S_{l-1}q, T_q)$ in $l$-candidate set do
2:    insert candidate $(S_{l+1}, T)$ where $S_{l+1} = S_{l-1}pq$ and $T = (T_pq) \cap (T_qp)$ in the $(l+1)$-candidate set
3:    for each $S_l \subset S_{l+1}$ except $S_{l-1}p$ and $S_{l-1}q$ do
4:      if candidate $(S_l, T_l)$ does not exist then
5:        remove candidate $(S_{l+1}, T)$ and move to the next pair of candidates
6:      else
7:        $T = T \cap (T_l z)$ where $z = S_{l+1} \backslash S_l$
8:      end if
9:      if $T = \emptyset$ and $(S_{l+1}, \emptyset)$ satisfies Criterion 3 or 4 then
10:       remove candidate $(S_{l+1}, \emptyset)$ and move to the next pair of candidates
11:     end if
12:   end for
13: end for
```

The function takes two steps for candidate generation: combination and pruning. Lines 1 and 2 are for combination, and lines 4 to 10 are for pruning. We firstly illustrate the combination. Suppose that we have two candidates $(abc, \{a\})$ and $(abd, \{a, d\})$. They have the same prefix ab, and the pattern of the new candidate is $abcd$ by joining patterns of two candidates. For the RHS set of the new candidate, both RHS sets are firstly expanded as $\{a, \mathbf{d}\}$ and $\{a, d, \mathbf{c}\}$ since d and c are new to candidates $(abc, \{a\})$ and $(abd, \{a, d\})$ respectively. Then the

RHS set of the new candidate takes the intersection of both expanded RHS sets. The new candidate is $(abcd, \{a, d\})$. The intersection of RHS sets here and on line 7 is to ensure that removed potential RHSs from the RHS set of a candidate never appear on the RHS sets of its super candidates. The correctness is guaranteed by pruning Criteria 1 and 2.

Secondly we show the pruning process. Suppose that other 3-candidates are: $(acd, \{c, d\})$ and $(bcd, \{b\})$. For the new candidate $(abcd, \{a, d\})$, we need to make sure that all its sub candidates are in the 3-candidate set. In this example, they are and hence candidate $(abcd, \{a, d\})$ is kept. RHS set expansion and intersection are applied to every sub candidate. With sub candidate $(acd, \{c, d\})$, the RHS set of $(abcd, \{a, d\})$ becomes d. With sub candidate $(bcd, \{b\})$, the RHS set becomes $\emptyset$. Now, the new candidate becomes $(abcd, \emptyset)$. The removal of this candidate will be determined by the satisfaction of Criteria 3 or 4.

5.3 Pruning and CFD testing

In this subsection, we present another pruning process after counting the support of candidates in addition to the one in candidate generation. This is a key to the efficiency of the algorithm. In the following algorithm, ϵ is the minimum support, and $\{S\backslash z\}$ means the set difference between S and $\{z\}$.

Function 2. *Prune and test $(l + 1)$ candidate set $(l + 1)$ is the new level after supports are counted.*

```
1:  for each candidate $(S, T)$ in $(l+1)$-candidate set do
2:    if $\mathrm{supp}(S) \le \epsilon$ then
3:      remove candidate $(S, T)$ and move to the next candidate
4:    end if
5:    for each $z \in T$ do
6:      if $\mathrm{supp}(S\backslash z) = \mathrm{supp}(S)$ {Criterion 1} then
7:        add $(S\backslash z) \to z$ to the CFD set, remove $z$ from $T$ and label candidate $(S, T)$ restricted
8:      else
9:        if there is an $x \in (S' = S\backslash z)$ such that $\mathrm{supp}((S'\backslash x)\neg z) = \mathrm{supp}(S'\neg z)$ then
10:         remove $z$ from $T$ {Criterion 2}
11:       end if
12:     end if
13:   end for
14: end for
15: for each candidate $(S, \emptyset)$ in $(l+1)$-candidate set do
16:   if $(S_{l+1}, \emptyset)$ satisfies Criterion 3 or 4 then
17:     remove candidate $(S_{l+1}, \emptyset)$
18:   end if
19: end for
```

A candidate is pruned from two aspects, the infrequency of the pattern and the satisfaction of Criterion 3 or 4. On line 2 a candidate with infrequent pattern is removed. From lines 6 to 10, we limit attribute value pairs in the RHS set of a candidate based on Criteria 1 and 2. On lines 16 and 17, we consider removing a candidate based on Criterion 3 or 4.

We introduce a concept *restricted candidate* to easily test the satisfaction of Criterion 3.

Definition 10. *Candidate $\{Px, T\}$ is restricted if there is a candidate (P, T_1) such that* $\mathrm{supp}(Px) = \mathrm{supp}(P)$.

To test if candidate $\{Px, T\}$ satisfies Criterion 3, we need to test if its support equals to the support of any of its sub candidate. When the length of Px is long, there are many such sub candidates, and the test will affect efficiency. In this algorithm, we only test this support equality with its immediate sub candidates whose patterns have one attribute value pair less. If their supports are equal, we label the candidate as restricted. All super candidates of a restricted candidate are restricted too. This is because $\mathrm{supp}(Px) = \mathrm{supp}(P) \Rightarrow \mathrm{supp}(PQx) = \mathrm{supp}(PQ)$ for any Q. This means the restricted status is inheritable from any of its sub candidates. Furthermore, the RHS set of a restricted candidate is not expandable and only prunable based on Criterion 2. A restricted candidate is prunable when its RHS set is empty.

We use an example to show how function Pruning and Testing works.

Example 4. *Given the following data set. We list one attribute value in each attribute for easy illustration.*

A	B	C	D	E
	b	c		e
a	b	c		e
a		c	d	e
a	b		d	e
	b	c	d	

The candidate generation and pruning are illustrated in Figure 2. For brevity, we do not draw all edges. All candidates are grouped by the same prefixed pattern for the easy observation of candidate generation and termination. Candidates in dash-lined boxes are restricted. In level 3, restricted candidates are pruned by Criterion 3, and others are pruned by Criterion 4.

In the second level, only one RHS value is pruned. Candidate $(ae, \{a\})$ is restricted since $\text{supp}(ae) = \text{supp}(a)$. CFD $a \rightarrow e$ is generated and e is removed from the RHS set of the candidate following Criterion 1. All its super candidates are restricted too.

In the third level of candidate tree. We use candidates $(abc, \emptyset)$, $(abe, \emptyset)$ and $(ade, \emptyset)$ to illustrate how candidates are generated and pruned.

Candidate $(abc, \{a, b, c\})$ is generated initially with all possible RHSs since each of its sub candidate has complete RHS set. Since $\text{supp}(a\neg c) = \text{supp}(ab\neg c)$, c is removed from the RHS set. Since $\text{supp}(a\neg b) = \text{supp}(ac\neg b)$, b is removed from the RHS set. Similarly, a is removed from the RHS set because of $\text{supp}(b\neg a) = \text{supp}(bc\neg a)$. Therefore, the candidate set becomes $(abc, \emptyset)$ after the pruning. Candidate $(abc, \emptyset)$ is eventually pruned by Criterion 4 because candidates $(abd, \emptyset)$ and $(abe, \emptyset)$ do not have RHS values d and e respectively. Candidates $(abd, \emptyset)$, $(acd, \emptyset)$, $(bcd, \emptyset)$, $(bce, \emptyset)$, and $(cde, \emptyset)$ are generated and pruned in the same way.

Candidate $(abe, \{a\})$ is initially generated with one RHS value inherited from candidate $(ae, \{a\})$ since they are restricted. The RHS value a is then pruned because of $\text{supp}(e\neg a) = \text{supp}(be\neg a)$. Candidate $(abe, \emptyset)$ is pruned since it satisfies Criterion 3. Candidate $(ace, \emptyset)$ is generated and pruned in the same way.

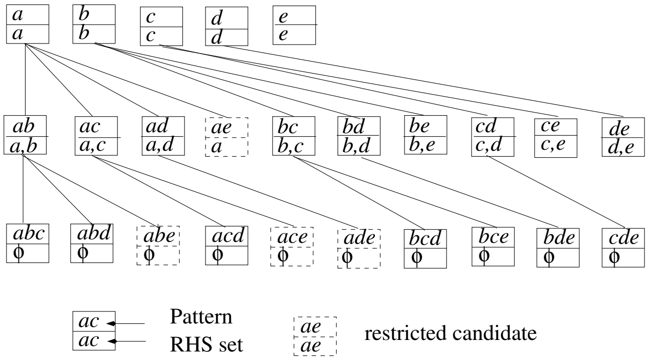

Figure 2: All candidates in Example 4

Candidate $(ade, \{a\})$ is initially generated with one RHS value inherited from candidate $(ae, \{a\})$ since they are restricted. Since $\text{supp}(de) = \text{supp}(ade)$, CFD $de \rightarrow a$ is generated and a is removed from the RHS set of the candidate following Criterion 1. Candidate $(abe, \emptyset)$ is pruned since it satisfies Criterion 3.

No candidate is left. The program returns CFDs $\{a \rightarrow e, de \rightarrow a\}$ and terminates.

5.4 Algorithm

Now we are able to present our CFD discovery algorithm in Algorithm 1. Two main functions have been discussed in the previous subsections.

Algorithm 1 Fast Algorithm for CFD Discovery (FACD)

Input: data set D and the minimum support ϵ.
Output: The minimal set of CFDs.

```
set CFD set $F = \emptyset$
count the support of 1 and 2 patterns
build 1 and 2-candidate sets
prune and test 1 and 2-candidate sets
add CFDs to $F$
generate 3-candidate set
while new candidate set is not empty do
    count support of patterns for new candidates
    prune and test the new candidate set
    add CFDs to $F$
    generate next level candidate set
end while
return
```

Algorithm 1 is self explanatory. Level 1 and 2 candidates are counted by an array. We still build and prune level 1 and 2 candidates to produce the

RHS sets and restricted status for the following candidate generation. After that, candidate generation and pruning are applied to each level of candidates before and after the support count. CFD selection has been conducted at each level too. The returned CFD set is the minimal set of CFDs.

Theorem 1. *Algorithm 1 generates the minimal set of CFDs correctly*

Proof. Firstly, without any pruning, the first two lines of Candidate-generator will generate the complete set of patterns and hence all candidate CFDs will be examined.

Secondly, all pruning of candidate CFDs is based on Criteria 1, 2, 3, and 4, which guarantee that the pruned CFDs will not be in the minimal set of CFDs.

Thirdly, once a CFD is formed, Criterion 1 will remove the candidates for producing any of its more specific CFDs, and hence the output CFDs is the minimal set of CFDs.

Therefore, Algorithm 1 generates the minimal set of CFDs correctly. □

The time complexity of Algorithm 1 is mainly determined by the size of search space. Let N be the size of the search space, and n be the size of a data set. The complexity of the algorithm is $O(Nn)$. It is difficult to estimate N after the pruning. Let m be the number of attributes, and p be the average domain size of attributes. Let us search for CFDs with up to l values in their LHSs. Note that this does not mean that the longest LHS of discovered CFDs is l. It is possible that all candidates have been searched, but no single CFD is found. In the worst case, $N_{\max} = \sum_{i=1}^{l+1}(C(m,i)p^i)$ where $C(m,i)$ denotes the number of combinations of i attributes from m attributes and p^i indicates the number of value combinations within i attributes. In the worst case, the complete search space has been searched. $N_{\max} \approx (mp)^{l+1}$ when $l \ll m$. In the best case, $N_{\min} = \sum_{i=1}^{2}(C(m,i)p^i) \approx (mp)^2$. In the best case, it becomes clear that all other candidates do not contain CFDs after level 2 candidates have been searched. For example, let $m = 16$, $l = 6$ and $p = 4$. $N_{\max} = 2^{42}$ and $N_{\min} = 2^{12}$. The difference between $N_{\max}$ and $N_{\min}$ is huge. Generally speaking, the search space can be any size in between. There is not a good estimation of N because the effectiveness of pruning depends on the value distribution in a data set. In practice, N is much closer to $N_{\min}$ than $N_{\max}$. Let $N = \beta N_{\max}$ where β is a very small number and is determined by the value distribution of a data set. The complexity is still high for data sets of many attributes. In sum, the algorithm scales well with the data size, but does not scale well with the number of attributes. This is common for association and non-redundant rule discovery algorithms [6, 9, 11, 19]. We will assess its efficiency and scalability in experiments.

6 Interesting CFDs

We normally find many CFDs from a medium data set. Are they all interesting?

A CFD $(X \to Z, x||z)$ summarises a fact that when value (tuple) x occurs in attribute X of a record then value z must occur in attribute Z of the same record. When we ignore the database scheme, this means that the set of records containing x is a subset (or an equal set) of the set of records containing z. We define the covering set of a tuple as all records containing the tuple. We denote this relationship as $\mathrm{cov}(x) \subseteq \mathrm{cov}(z)$. A key question is if $\mathrm{cov}(x) \subseteq \mathrm{cov}(z)$ occurs just by chance. For example, let the size of a data set be 1000, and let z occur in the data set 500 times. We assume that x only occurs twice. The chance of $\mathrm{cov}(x) \subseteq \mathrm{cov}(z)$ is $1/2{*}1/2 = 0.25$. This means that even if CFD $(X \to Z, x||z)$ has been discovered, it likely occurs just by chance and has nothing to do with a database constraint.

Fundamentally, we do not wish to have a CFD $(X \to Z, x||z)$, with x and z being independent. Chi square test can be used to test if x and z are independent. Let the observation from a data set be summarised as follows.

	z	$\neg z$
x	$\mathrm{supp}(xz)$	$\mathrm{supp}(x\neg z)$
$\neg x$	$\mathrm{supp}(\neg xz)$	$\mathrm{supp}(\neg x\neg z)$

Here $\neg x$ and $\neg z$ mean that x and z do not occur in a record. $\mathrm{supp}(xz)$ means the proportion of records containing both x and z. $\mathrm{supp}(\neg x\neg z)$ means the proportion of records containing neither x or z. $\mathrm{supp}(x\neg z)$ means the proportion of records containing x but not z, and $\mathrm{supp}(\neg xz)$ means the proportion of records containing z but not x.

If x and z are independent, the expected supports in cells are listed in the following table.

	z	$\neg z$
x	$\mathrm{supp}(x) * \mathrm{supp}(z)$	$\mathrm{supp}(x) * \mathrm{supp}(\neg z)$
$\neg x$	$\mathrm{supp}(\neg x) * \mathrm{supp}(z)$	$\mathrm{supp}(\neg x) * \mathrm{supp}(\neg z)$

Here $\mathrm{supp}(\neg x)$ and $\mathrm{supp}(\neg z)$ mean the proportion of records that do not contain x and z respectively.

Chi square value indicates the variance of the observed supports to the expected supports.

$$\chi^2 = \sum_i^4 (\mathrm{supp}_i^o - \mathrm{supp}_i^e)^2 / \mathrm{supp}_i^e$$

where i is a cell in a contingency table, and supp_i^o and supp_i^e are observed and expected supports in the corresponding cell.

The chi square values can be mapped to statistical significance, in this case with two degrees of freedom. For instance, $\chi^2 > 5.99$ indicates a p-value of 0.05. However, since we conduct a huge number of such tests, many associations are also caused by chance. This is a result of multiple comparisons [20]. Because of this multiple testing, the nominal p-values should not be interpreted as correct statistical significance. However, the p-values or the chi square values are useful as a tool to rank patterns

In our previous example, x occurred only twice, and both times with z, which in turn occurred in 500 out of 1000 tuples. $\chi^2 = 0.33$. This indicates that they are actually independent. Therefore, the CFD is not interesting.

Chi square test should not be used for small sample sizes because it is only an approximation of the true distribution [22]. In our case this is not a problem, since a frequency threshold will prune rare CFDs. If needed, Fisher's exact test [22] can be used for small sample sizes to compute exact nominal p-values.

7 Experiments

7.1 Efficiency and scalability

We have conducted experiments on the data sets as described in Table 2.

Data sets	Size	#Attributes
US Airline	520417	23
Mushroom	8124	23
Census Income	299285	21
Audiology	200	70

Table 2: A description of data sets

The US Airline data set has been downloaded from (http://www.transtats.bts.gov/). We have downloaded one month data and removed rows with the majority of missing values. We have also removed irrelevant and redundant attributes and we keep 23 attributes. The dimension of US airline data set looks not high, but it has 15242 attribute values. This means that each attribute has on average 662 values. Such a large number of attribute value pairs is challenging for a CFD discovery algorithm.

Census income, Mushroom and Audiology data sets have been downloaded from the UCI Machine Learning data repository [21]. The Census Income data set is merged with the training and test data sets. Numerical attributes and irrelevant attributes have been removed. The Audiology data set has 70 attributes, and we use it to test the scalability of the algorithm with the number of attributes. The Audiology data set is a small data set, but its dimension is higher than other data sets. We use it to test the scalability of the algorithms.

There are functional dependencies in the US Airline data set. There are many CFDs in the US airline data set too, e.g. 28955 with the minimum support of 0.001. There are not functional dependencies in other data sets, however, there are many CFDs in these data sets.

To demonstrate the efficiency of the proposed algorithm, we compare the proposed algorithm with the currently most efficient CFD discovery algorithm CFDMiner [5]. The core component of CFDMiner is to find minimal generators and their closures. There are many algorithms for closed pattern mining [16]. Each one has its own strengths and weaknesses. For example, frequent pattern tree based methods [12, 13, 7] are normally time efficient but may not handle high dimensional data because of their large memory consumption. Branch and bound search based methods [14, 15] are normally memory efficient but may not be time efficient for small data sets since they scan a data set many times. To have a

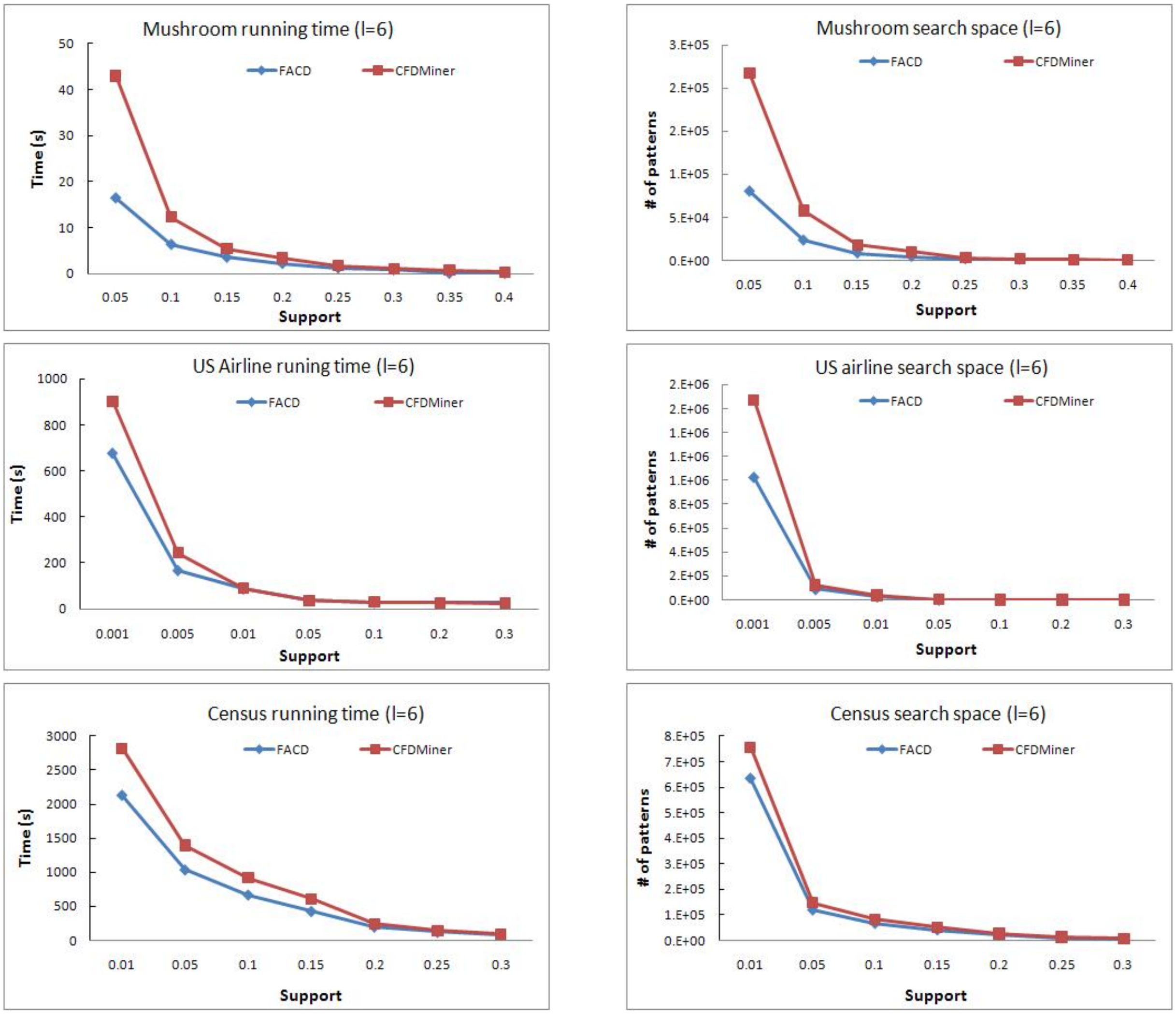

Figure 3: The search space and running time for CFD discoveries by FACD and CFDMiner

fair comparison, we have implemented a branch and bound algorithm, A-CLOSE [15], for minimal generator discovery, as the core for CFDMiner (we could not obtain the original implementation because of potential commercial interest on it.). Our aim is to handle large and very large data sets with the low minimum support. Memory efficiency is very important given the fact that some fast closed pattern mining algorithms do not handle large data data well or could not discover patterns with the low minimum support [16].

Experimental results of FACD in comparison to CFDMiner are listed in Figure 3. FACD is faster than CFDMiner on all three data sets. To understand the results, we list the number of candidates searched in Figure 3. The trend for time efficiency is the same as the trend for the size of search space. The improvement of efficiency is obtained by the pruning search space of the proposed algorithm.

The scalability of the proposed algorithm with the data set size and the number of attributes are listed in Figure 4. FCAD scales well with the size of a data set. So does CFDMiner. We note that some data points deviate from the straight lines in the figure. This is because the size of search space changes with the change of data sets. The overall trends of both algorithms are consistent. FACD does not scale well with the number of attributes. However, FACD performs better than CFDMiner with the increase of

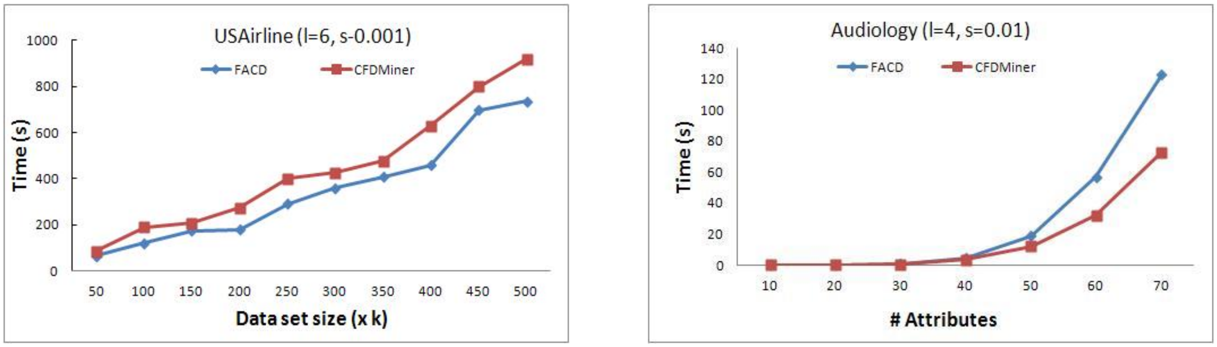

Figure 4: The scalability of the algorithm FACD with the data size and the number of attributes

the number of attributes. All rule and FD discovery algorithms do not scale well with the number of attributes since their time complexity are eventually exponential to the number of attributes [6, 19, 9, 11]. 70 attributes are high for these algorithms.

7.2 Significance of discovered CFDs

To understand the discovered CFDs, we study the support of LHS and RHS with the statistical significance of CFDs. Chi square test [22] is used to test if LHS and RHS are independent. Normally, if a chi square value is less than 2, LHS and RHS are very likely to be independent. The higher a chi square value, the lower possibility of independency between LHS and RHS. To obtain a reliable chi square value estimation, we need to keep the expected count in every cell of a contingency table above 5. We set the minimum support as 0.0031 to ensure that the expected support number of (supp(x, z)) of a CFD tuple pattern $x \rightarrow z$ is more than 5. We removed CFDs with the expected support number of less than 5 in any other cell from the discovered CFDs. We obtained 2091 CFDs from the US airline data set.

The distribution of chi square values of the discovered CFDs is listed in Figure 5. All chi square values are larger than 5. The distribution has two peaks. Many CFDs have very high chi square values (more than 10000 with the highest ones of 520417) and many have chi square values between 5-10 . Not many other CFDs locate in between. We list some typical ones in Table 3.

Most CFDs with very high chi square values are those with the strongest association (one to one mapping from the LHS to the RHS). For example, the first CFD in Table 3 instantiates FD “Destination World Area Code → Destination Name”. There are a number of CFDs with different Destination Cities.

Some CFDs with very high chi square values do not associate with FDs. For example, the second CFD in Table 3 does not correspond with an FD. However, it explains a number of flights flying between cities in Hawaii. Their characteristic is the short distance, which is too short to fly off Hawaii. These types of constraints are most likely not designed in a database but form in data.

A CFD with a middle level of chi square value is shown as the third CFD in Table 3. This captures a common sense in data — “If a flight arrives earlier than the schedule by 16-30 minutes, it most likely departs on time”. However, this information has not been captured by an FD because of few violations. The third CFD shows a subcase of flights departing at 6:00. There are a number of similar CFDs for different departure times.

Some CFDs can be uninteresting even though they have a chi square value higher than 2. The fourth CFD in Table 3 shows an example. It says that a flight departing between 18:00 and 19:00 and arriving at its Destination of World Area Code 91 between 19:00 and 2:00 has not been diverted. Given that 99.76% of flights have not been diverted, people may not be interested in such a “trivial truth”. A number of CFDs carry the similar semantic meanings.

Figure 5 shows the relationship of chi square values, supports of LHS and RHS. We see that CFDs with highest chi square values locate at the diagonal

1: ((Destination Wac* → Destination Name), 91 || California) supp(LHS) = 60211, supp(RHS) = 60211, $\chi^2 = 520417$ *Wac: World area code
2: ((Origin Sate Name, Distance Group → Destination State Name), Hawaii, 1 || Hawaii) supp(LHS) = 7150, supp(RHS) = 9436, $\chi^2 = 392596$
3: ((CRS Departure Time, Arrival Delay Groups → Departure Delay,), 6:00, -2 || 0) supp(LHS) = 1625, supp(RHS) = 429163, $\chi^2 = 346$
4: ((Destination Wac*, Departure Time Block, Arriving Time Block → Diverted), 91, 18:00-18:59, 19:00-19:59 || 0) supp(LHS) = 2085, supp(RHS) = 519163, $\chi^2 = 5.1$ *Wac: World area code

Table 3: Some typical examples of discovered CFDs

of the plane formed by supports of LHS and RHS. This means that they correspond to CFDs of one to one mapping between LHS and RHS. Other CFDs with high chi square values locate at the corner of low RHS support. CFDs with a highly supported RHS normally do not have high chi square values.

The discovered CFDs will mainly be used for data quality improvement, especially for detecting and fixing value inconsistencies. We have not discussed the application of CFDs in this paper. We refer readers to the following work [2, 3, 23] for the use of discovered CFDs.

8 Conclusions

In this paper, we have studied the problem of discovering the minimal set of constant CFDs that hold in some given data. As in previous work, we take advantage of the observations that constant CFDs essentially are 100% confidence association rules, and that the minimal set of CFDs can be produced from the set of minimal generators and their closures. We proposed new pruning criteria to further reduce the search space, removing unnecessary generators and closures.

We designed an efficient algorithm based on the new pruning criteria and we evaluated it on real data sets. According to the results, the proposed algorithm is faster than the currently most efficient constant CFD discovery algorithm. We also showed how chi square can be used to measure the interestingness of CFDs.

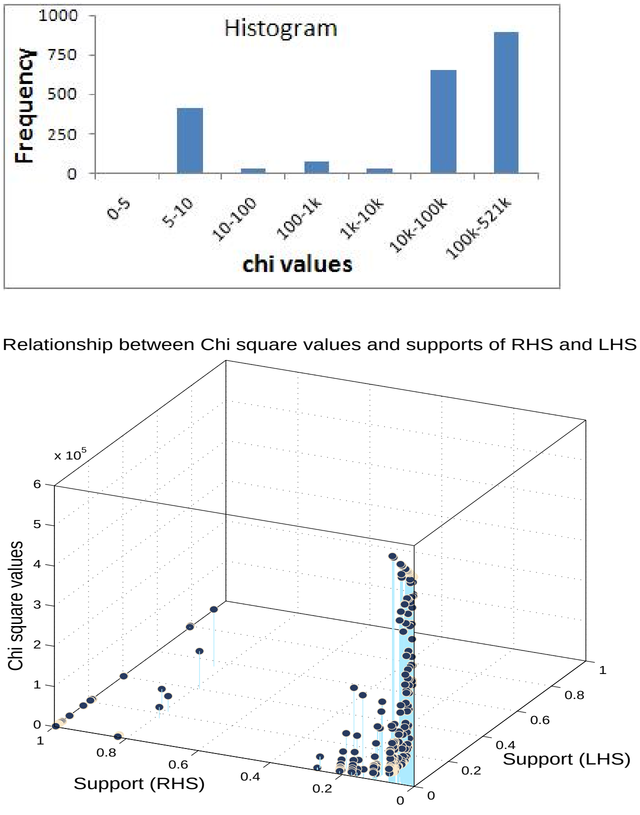

Figure 5: The histogram of chi square values of discovered CFDs and the relationships between chi-square values and support of LHS and RHS

References

[1] Chiang, F. and Miller, R. J. (2008) Discovering data quality rules. *Proceedings of the VLDB Endowment*, **1**, 1166–1177.

[2] Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. (2008) Conditional functional dependencies for capturing data inconsistencies. *ACM Transations on Database Systems*, **33**, 1–48.

[3] Fan, W. (2008) Dependencies revisited for improving data quality. *Proceedings of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, Vancover, Canada, 9-11, June, pp. 159–170. ACM, New York.

[4] Golab, L., Karloff, H., Korn, F., Srivastava, D., and Yu, B. (2008) On generating near-optimal

tableaux for conditional functional dependencies. *Proceedings of the VLDB Endowment*, **1**, 376–390.

[5] Fan, W., Geerts, F., Li, J., and Xiong, M. (2011) Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering*, **23**, 683–698.

[6] Agrawal, R. and Srikant, R. (1994) Fast algorithms for mining association rules. *Proceedings of International Conference on Very Large Data Bases (VLDB)*, Santiago De Chile, Chile, 12-15, September, pp. 487–499. Morgan Kaufmann, San Francisco, CA, USA.

[7] Li, H., Li, J., Wong, L., Feng, M., and Tan, Y.-P. (2005) Relative risk and odds ratio: a data mining perspective. *Proceedings of ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, Baltimore, Maryland, USA, 13-15 June, pp. 368–377. ACM, New York, USA.

[8] Balcázar, J. L. (2010) Redundancy, deduction schemes, and minimum-size bases for association rules. *Logical Methods in Computer Science*, **6(2)**, 1–33.

[9] Zaki, M. J. (2004) Mining non-redundant association rules. *Data Mining and Knowledge Discovery Journal*, **9**, 223–248.

[10] Agrawal, R., Imieliński, T., and Swami, A. (1993) Mining association rules between sets of items in large databases. *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD)*, Washington, D.C., USA, 26-28 May, pp. 207–216. ACM, New York, USA.

[11] Pasquier, N., Taouil, R., Bastide, Y., Stumme, G., and Lakhal, L. (2005) Generating a condensed representation for association rules. *Journal of Intelligent Information Systems*, **24**, 29–60.

[12] Pei, J., Han, J., and Mao, R. (2000) Closet: An efficient algorithm for mining frequent closed itemsets. *Proceedings of ACM SIGMOD International Workshop on Data Mining and Knowledge Discovery (DMKD)*, Dallas, Texa, USA, 14 May, pp. 21–30. ACM, New York, USA.

[13] Wang, J., Han, J., and Pei, J. (2003) Closet+: Searching for the best strategies for mining frequent closed itemsets. *ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Washington, D.C, USA, 24-27 August, pp. 236–245. ACM, New York.

[14] Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999) Efficient mining of association rules using closed itemset lattices. *Journal of Information Systems*, **24**, 25–46.

[15] Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999) Discovering frequent closed itemsets for association rules. *Proceedings of International Conference on Database Theory (ICDT)*, Jerusalem, Israel, 10-12 January, pp. 398–416. Springer-Verlag, London, UK.

[16] Yahia, S. B., Hamrouni, T., and Nguifo, E. M. (2006) Frequent closed itemset based algorithms: a thorough structural and analytical survey. *SIGKDD Exploration Newsletter*, **8**, 93–104.

[17] Li, J. (2006) On optimal rule discovery. *IEEE Transactions on Knowledge and Data Engineering*, **18(4)**, 460 – 471.

[18] Le Bras, Y., Lenca, P., and Lallich, S. (2009) On optimal rule mining: A framework and a necessary and sufficient condition of antimonotonicity. *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Bangkok, Thailand, 27-30 April, pp. 705–712. Springer, Berlin, Germany.

[19] Han, J., Pei, J., Yin, Y., and Mao, R. (2004) Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery Journal*, **8(1)**, 53–87.

[20] Megiddo, N. and Srikant, R. (1998) Discovering predictive association rules. *Proceedings of International Conference on Knowledge Discovery*

and Data Mining (KDD), New York, NY, USA, 27-31 August. pp. 274–278, AAAI Press, Menlo Park, California, USA.

[21] Frank, A. and Asuncion, A. (2010). UCI machine learning repository. http://archive.ics.uci.edu/ml, University of California, Irvine, School of Information and Computer Sciences.

[22] Triola, M. M. and Triola, M. F. (2005) *Biostatistics for the biological and health sciences*, 2nd edition. Addison-Wesley, Boston, USA.

[23] Fan, W., Geerts, F., and Jia, X. (2009) Conditional dependencies: A principled approach to improving data quality. *British National Conference on Databases (BNCOD)*, Birmingham, UK, 7-9 July, pp. 8–20. Springer, Berlin, Germany.