جـامـعـة الـشـرق الأوسـط للدراسـات العـليا
MIDDLE EAST UNIVERSITY FOR GRADUATE STUDIES

# CFD_Mine: Discovery of Conditional Functional Dependencies in Relational Databases.

**A Thesis**
**Submitted in Partial Fulfillment of the**
**Requirements for the Master Degree**

**In**

**Computer Science**

**By**

Mohammed Raghep Mohammed Hakawati

Supervisor

Prof. Musbah Mah'd Aqel

**Computer Science Department**
**Faculty of Information Technology**
**Middle East University for Graduate Studies**
**Amman-Jordan**
**July, 2009**

## جامعة الشرق الأوسط للدراسات العليا

## أقرار تفويض

أنا محمد راغب محمد حكواتي، أفوّض جامعة الشرق الأوسط للدراسات العليا بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

الأسم : محمد راغب محمد حكواتي.

التوقيع :

التاريخ : ١٠ / ١٠ / ٢٠٠٩م

## Middle East University for Graduate Studies
## Authorization Statement

I am *Mohammed Raghep Mohammed Hakawati* ; authorize The Middle East University for Graduate Studies to supply copies of my Thesis to libraries or establishments or individuals upon request.

Name: *Mohammed Raghep Mohammed Hakawati*

Signature:

Date: 10 /10 /2009

Middle East University for Graduate Studies

## Examination Committee Decision

This is to certify that the Thesis entitled "CFD_Mine: Discovery of Conditional Functional Dependencies in relational databases" was successfully defended and approved on August 11 th 2009.

| Examination Committee Members | Signature |
|---|---|
| Prof. Musbah M Aqel<br>Professor, Department of Computer Science<br>(Middle East University for Graduate Studies) | |
| Dr. Jalal Atuom<br>Associate Professor, Department of Computer Science<br>(New York Institute of Technology) | |
| Prof. Nidal Shilbayeh<br>Professor, Department of Computer Science<br>(University) | |
| Prof. Rustom Mamlook<br>Professor, Department of Computer Science<br>(Middle East University for Graduate Studies) | |

# Dedication

This thesis is dedicated to my beloved mother and father, who brought me up from childhood, educated me, and gave me the courage to confront the challenges of life.

To my brother and my right wing, Ahmed.

To the two special girls in my life, leka'a and Ranem, and to all my friends and colleagues.

# Acknowledgment

Prior to acknowledgments, I must glorify Allah the Almighty who gave me courage and patience to carry out this work successfully.

I would be restating the obvious when I say Dr. Musbah Aqel is a great advisor. I consider myself truly fortunate to have him as my advisor and would like to thank him sincerely for the opportunity to work under his guidance and support. I am deeply grateful to him for showing immense patience during the long and frustrating phase of research problem identification. I am also thankful to his constructive criticism during one-on-one meetings as well as discussions at the weekly group meetings. I wish to thank him for giving me the opportunity to work on this topic and for preserving with me throughout the time it took me to complete this research.

Finally, I would like to express my great thanks to my parents, brothers, and sisters for their immense patience, emotional support and engorgement during my entire graduate study.

# Table of Contents

# List of Tables

# List of Figures

# Abstract

Dirty data (i.e. containing inconsistences, conflict and errors) is a serious problem for many organizations leading to incorrect decision making, inefficient daily operations, and ultimately wasting both time and money. Dirty data in a database often emerge as violation of integrity constraints, meant to preserve data consistency and accuracy.

Conditional Functional Dependencies (CFDs) have recently been introduced for data cleaning. CFDs extends Functional Dependencies (FDs) by enforcing patterns of semantically related values , and have proved more effective in catching data inconsistencies than FDs , which were currently the basis of many data-Cleaning tools

Discovery of CFDs existing in an instance of a relation is an expensive process that involves intensive manual effort. In this thesis, the researcher develops an effective algorithm, called **CFD_Mine** for discovering CFDs in a relation instance. **CFD_Mine** is a Levelwise algorithm that extends TANE, a well-known algorithm for discovering FDs. it searches for minimal CFDs among the data values and prunes redundant candidates.

An experimental study is presented for showing the scalability of our algorithm .Finally the results show that **CFD_Mine** works well when a given sample relation is large and scales well will the arity of the relation.

# الملخص


البيانات المحتوية على أخطاء هي مشكلة حقيقية في معظم المنظمات و الشركات , تؤدي هذة الأخطاء الى الخطأ في أتخاذ القرارت , و عدم فعالية العمليات اليومية , وضياع في الوقت و المال. في العادة تصنف البيانات على أنها تحتوي أخطاء أذا كانت تخالف حالات التقيد السليمة.

العلاقات الوظيفية المشروطة قُدمت موخراً لتنظيف البيانات من هذه الأخطاء , وهي عبارة عن أمتداد للعلاقات الوظيفية التقليدية مدعومة بقيم ذات معنى من نفس جداول البيانات , و قد أثبتت كفاءتها في أكتشاف الأخطاء أكثر من العلاقات الوظيفية التقليدية التي كانت لمدة من الزمن تستخدم في تنظيف البيانات .

أكتشاف العلاقات الوظيفية المشروطة الموجودة في الجداول هي عملية مكلفة أذا ما قمنا بها يدوياً , في هذه الرسالة طورنا خوارزمية **( CFD_Mine)** لأكتشاف هذه العلاقات , هذه الخوارزمية من النوع المتسلسل في المستوى أثناء البحث و تقلم التكرار في هذه العلاقات.

قدمنا أيضا في بحثنا هذا دراسة تجريبية تظهر النمو في الخوارزمية , و نتائجنا أثبتت أن هذه الخوارزمية تعمل بشكل صحيح عند أزدياد حجم قاعدة البيانات المعطاة.

# CHAPTER ONE

# INTRODUCTION

## 1.0 Overview

The prevalent use of information systems has made data one of the most valuable assets in most organizations. Nevertheless, the value of data highly depends on its quality.

Dirty data (i.e. containing inconsistencies and errors) is a serious problem for businesses; leading to incorrect decision making, inefficient daily operations, and ultimately wasting both time and money.

The presence of errors and inconsistencies in data dramatically reduce the value of data, making it worthless, or even harmful. Recent statistics reveals that dirty data costs US businesses 600 billion dollar annually [English, 2000]. It is also estimated that data cleaning, a labor-intensive and complex process, accounts for 30 to 80% of the development time and budget in most data warehouse projects [Shilakes and Tylman, 1998] .A study conducted by Gartner in 2005 forecasts that more than 50 percent of data warehouse projects will have limited success, or will be outright failures, as a result of the lack of attention to data quality issues [Gartner, 2005]. In light of these, there has been increasing demand for data cleaning /quality tools to automatically detect and effectively remove inconsistencies and errors from the data.

Dirty data often arises due to changes in use and perception of the data, and violation of integrity constraints (or lake of such constraints). Integrity constraint - meant to preserve data consistency and accuracy - are defined according to domain specific business rules, these rules define relationships among a restricted set of attribute values that are expected to be true under a given context. For example, an organization may have rules such as: all new customers will receive a 15% discount on their first purchase and preferred customers receive a 25 % discount on all purchases [Chiang and Miller, 2008].

Inconsistencies and errors in a database often emerge as violations of integrity constraints [Arenas et al., 2003], [Rahm and Do, 2000]. Integrity constraints (a.k.a. data dependencies) are been widely used for improving the quality of schema. Recently constraints have enjoyed a revival for improving the quality of data.

Constraint-based data cleaning has mostly focused on two topics, introduced in [Arenas et al., 2003], repairing: is to find another database that is consistent and minimally differs from the Original database, and consistent query answering: is to find an answer to a given query in every repair of the original database, without editing the data,

There has been a host of work on data cleaning (*e.g.,* [Lopatenko and Bravo, 2007] [Arenas and Bertossi, 1999] [Bohannon et al., 2005] [Chomicki and Marcinkowski, 2005] [Jef, 2003]). However, to develop practical data-cleaning tools there is much more to be done. First, the previous work often models the consistency of data using traditional dependencies, *e.g.,* Functional Dependencies (FDs). Traditional FDs were developed mainly for schema design, but are often inadequate for data cleaning. These call for the use of constraints particularly developed for data cleaning that are able to catch more inconsistencies than traditional dependencies [Rahm and Do, 2000]. Second, few algorithms have been developed for automatically finding repairs, and even less incremental methods are in place. Third, none of the previous automated methods provides performance guarantee for the *accuracy* of the repairs found.

These limitations in Traditional Dependencies lead the authors in Data Cleaning to revive action by considering extensions of **FDs** and **INDs (** Inclusion Dependencies **)**, referred to as Conditional Functional Dependencies **CFDs** and Conditional Inclusion Dependencies **CINDs** ( Conditional Inclusion Dependencies ) , respectively, by additionally specifying patterns of semantically related values; these patterns impose conditions on what part of the relation(s) the dependencies are to hold and which combinations of values should occur together [Wenfei et al., 2008 (2)].

## 1.1 Motivation

Research on data quality has been mostly focusing on (a) error correction, (a.k.a. data imputation), (b) object identification, (a.k.a. record linkage, merge-purge, data deduplication and record matching), and (c) profiling, to discover meta-data from sample data. There is also an intimate connection between data quality and data integration, data standardization, data acquisition, cost estimation, schema evolution, and even schema matching.

Cleaning the data manually is unrealistic when the dataset is large. Indeed, manually cleaning a set of census data could easily take months by dozens of clerks [Winkler, 2004]. This highlights the need for automated data cleaning tools to detect and effectively remove inconsistencies and errors in the data. The need for discovering the constraint that the relation based on is important and easily helps to detect the tuples that violate this constraint, and prevent the end-user to add a new errornuce tuples, so, contribute to increase the consistency of the relational database in business and other domains.

There has been increasing demand for data quality tools, to add accuracy and value to business processes. A variety of approaches have been put forward: probabilistic, empirical, rule-based, and logic-based methods. There have been a number of commercial tools for improving data quality, most notably ETL tools (Extraction, Transformation, Loading), as well as research prototype systems, e.g., Ajax, Potter's Wheel, Artkos and Telcordia. [Maletic and Marcus, 1999][Rahm and Do, 2000].

Most data quality tools, however, are developed for a specific domain (e.g., address data, customer records). Worse still, these tools often heavily rely on manual effort and low-level programs that are difficult to write and maintain.

Our approach presents recent advances in constraint-based data cleaning , **CFDs** Rules repair the relation dataset based on two main phases , (a) Discovering Rules, to find the Rules that the relation depends on, and (b) Repairing Inconsistencies, to identify tuples that have some error in some of its fields (violate the discovered Rules).

The demand of finding an approach for automatically discovering the rules form relational dataset presents the initial step for data cleaning phases, by getting the most correct values from the relation , and prepare them to the next step of cleaning the data which is a new approach based on repairing the relation data on the discovered rules .

## 1.2 Contributions of the Thesis

The thesis contributions are the following:

**1.** Proposing a method for discovering both a minimum set of Conditional Functional dependencies **CFD** and a Functional Dependencies **FD**. Even though, the underlying ideas are not new, this is the first algorithm concentrates on discovering **both** Rules from database.

**2.** Implementing two new optimizations for finding correct and more accrue **CFD**, the first one is *merging* the similar **CFD** for finding a few and more accrued Rules, while as the second one is *finding* the minimum set of **CFD** rules based on the intersect Partitions (Common Partitions) between the Candidates (Element on the lattice).

**3.** Developing an application for finding the **CFDs** and **FDs** from any relation located anywhere, and with any extension , this application generates the partitions for the attribute set and then generates the Rules , you can change the accuracy of the discovered **CFD** rules , and you can filter the discovered rules after generate them.

## 1.3 Outline of the Thesis.

In **Chapter Two** we give some background and survey the literature about integrity constraints and their discovery.

In **Chapter Three** we present our general architecture for Rules discovery, and give instance case for each method in our approach.

In **Chapter Four** we describe the testing of our algorithm on both real life and study how input parameters and data characteristics influence the performance of our application.

Finally, we conclude in **Chapter Five.**

# CHAPTER TWO

# LITERATURE REVIEW

## 2.0 Background

A few works are done on the area of **CFD** Mining (Discovery), because this area of research is fresh, nevertheless, this area depends mainly on Mining **FD** rules which present the backbone to **CFD**.

Therefore, we cover the mining rules briefly in this chapter, and investigate the relation between the **FD** and Data Cleaning, and between **FD** and **CFD**.

**Definition** (Basic relational database concepts).

A relation schema R is a finite set of attributes. The domain of an attribute A, denoted by Dom (A) is the set of all possible values of A.

A tuple t over a relation schema R = $\{A_1 \ldots\ldots A_m\}$ is a member of the Cartesian product Dom $(A_1) \times \ldots \times$ Dom $(A_m)$.

A relation r over R is a finite set of tuples over R. The cardinality of a set X of tuples is denoted by |X|.

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

## 2.1 Functional Dependency (FD)

Functional dependencies are relationships between attributes of a relation: a Functional Dependency states that the value of an attribute is uniquely determined by the values of some other attributes.

Let r (U) be a relation and X, Y $\subseteq$ U. A Functional Dependency (FD) is a constraint, denoted X $\rightarrow$ Y. The FD X $\rightarrow$ Y is satisfied by r(U) if every two tuples $t_i$, $t_j \in$ r(U) that have $t_i(X) = t_j(X)$ also have $t_i(Y) = t_j(Y)$.

In an FD X$\rightarrow$Y, we refer to X as the *antecedent* and Y as the *consequent*.

### 2.1.1 Functional Dependencies and Data Cleaning

Functional Dependency (**FD**) is an important feature for referencing to the relationship between attributes and Candidate keys in tuples. It also shows the relationship between entities in a data model [Calvanese et al., 2001]. In research areas of data cleaning [Arenas and Bertossi, 1999] [Bohannon et al., 2005], the FD is used for improving the data quality. In a data mining research, an **FD** discovery technique has been studied [Huhtala et al., 1998] [Flach and Savnik, 1993]. However, an **FD** discovery could find too many FDs and, if used directly in a cleaning process, could cause it to NP time [Bohannon et al., 2005]. Many techniques developed as cleaning engine by combining an **FD** discovery technique with Data Cleaning technique.

Maletic and Marcus [Maletic and Marcus, 1999] introduced an automated data cleaning framework. Their work is divided into 2 parts: identifying error and cleaning data. The underlying theoretical aspects of the data quality of their research is a combination of existing problem-solving methods in software testing, data mining, knowledge based systems, and machine learning to address the framework. According to their research, to design automated data cleaning, one has to identify errors and then clean such dirty data. Several approaches use the **FD** discovery algorithm for identifying errors and cleaning algorithm together to produce **FD** cleaning tool. Several researchers in this field have mentioned that too many **FDs** have been generated [Arenas and Bertossi, 1999] [Ilyas et al., 2004].

The authors in [Huhtala et al., 1998] showed a pruning technique for generating a Candidate set and computing each Candidate member to determine FDs. The ranking technique has been proposed in [Ilyas et al., 2004] and [Andritsos et al., 2004].

Applied a selectivity value for ranking FDs from generated FDs (called "SoftFD") [Ilyas et al., 2004]. Their work proposed that if p1 and p2 are predicates on respective columns C1 and C2, then the selectivity of the conjunctive predicate p1 ∧ p2 is estimated by simply multiplying together the individual selectivity of |C1||C2| / |C1, C2|. The Authors in [Andritsos et al., 2004] proposed that the FD ranking should be concerned on the first merge of the attribute that has the most amount of duplicate attribute value. These 2 ranking techniques give us the idea of ranking by looking at the data distribution.

However, the merging technique will take more time than the selectivity value because it generates the clustered matrix but the selectivity value which can be found by counting attribute value directly. Therefore, this work will choose the selectivity value technique for ranking the generated FDs. There are 2 parts for cleaning algorithms: FD repairing technique and Duplicate Elimination. FD repairing which has been proposed by [Bohannon et al., 2005]. Their research used a cost based technique which used a low cost data to repair a high cost data. [Hernandez and Stolfo 1995] Proposed Sorted Neighborhood methods for Data Duplicate elimination by finding keys to determine duplicate tuples, then sorting the duplicate tuples and finally, matching tuples in the window to identify its duplication.

### 2.1.2 Levelwise Search Technique

Mannila and Toivonen [Mannila and Toivonen. 1997] , study thoroughly a breadth first or levelwise algorithm, also called generic data mining algorithm for finding all potentially interesting sentences. Their paper includes a complexity analysis, as well as some applications including functional and inclusion dependency discovery. The levelwise algorithm has been used among other applications for discovering association rules [Agrawal and Srikant, 1994] [Mannila et al., 1994], for discovering functional dependencies [Huhtala et al., 1998] [St´ephane et al., 2000] [Novelli and Cicchetti, 2001], and for discovering inclusion dependencies [Fabien et al., 2002].

The idea in the levelwise algorithm is to start from the most general attributes and try to generate and evaluate more and more specific attribute set. The semi-lattice illustrated in Figure 2.1, shows the search space of an exhaustive algorithm for finding Rules for four attributes. Figure 2.1 shows all possible nonempty combinations of the four attributes (*A, B, C, and D*).

For these attributes, there are $2^n = 2^4 = 16$ possible subsets of attributes, of which the $2^n - 2 = 14$ nonempty, proper subsets are the Candidates. The levels of the semi-lattice are numbered from the top to the bottom. The set *U* at level 4 are not a Candidates, because for any **CFD** or **FD** with the form $U \rightarrow vi$, we have $vi = U - U = \emptyset$. There are $n^2 (n-1)$ edges in a full lattice for n attributes. Since the semi-lattice of the total search space of Rules starts from level 1, rather than the empty set, there are $n2^{n-1} - n$ *edges* in the semi-lattice of the complete search space for Rules ,the size of the search space is exponential to the number of variables in *U*.

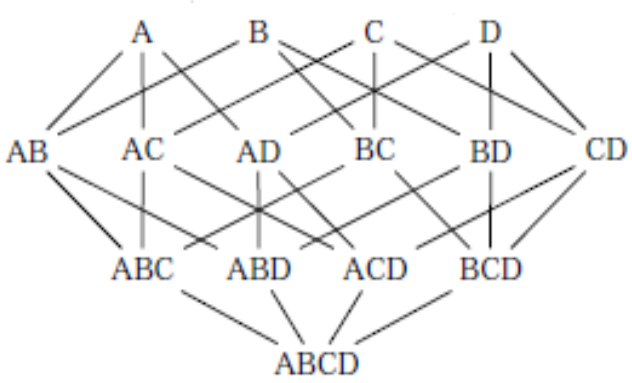


**Figure 2.1: Lattice for 4 attributes.**

## 2.2 Conditional Functional Dependencies

Constraints adopted for detecting inconsistencies are mostly traditional dependencies such as functional dependencies **FDs** and inclusion dependencies **INDs**, These constraints are required to hold on entire relation(s), and often fail to capture errors commonly found in real-life data.

These limitations lead the researchers to considering an extensions of **FDs** and **INDs**, referred to as Conditional Functional Dependencies **CFDs** and Conditional Inclusion Dependencies **CINDs**, respectively, by additionally specifying patterns of semantically related values; these patterns impose conditions on what part of the relation(s) the dependencies are to hold and which combinations of values should occur together.

**CFDs** extend **FDs** by incorporating a pattern tuple of semantically related data values. For each attribute *A* in a schema *R*, we denote its associated domain as Dom (*A*), which is either infinite (*e.g.*, string*;* real) or infinite (*e.g.*, Boolean*;* date).

A **CFD** $\varphi$ on R is a pair (R: X → Y, Tp), where,

(1) X and Y are sets of attributes in attr(R),

(2) X → Y is a standard **FD**, referred to as the FD embedded in $\varphi$,

(3) Tp is a tableau with attributes in X and Y , referred to as the pattern tableau of φ, where for each A in X ∪ Y and each tuple t ∈ Tp, t[A] is either a constant 'a' in dom(A), or an unnamed variable '_' that draws values from Dom(A)[Wenfei et al., 2008 (1)].

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

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

Table 2.1 shows a sample records from library instance which contains records about items available in the library and its *Name*, *Type*, *Country*, *Price* and *Tax*, and this relation holds on this Functional Dependency:

**FD: [Name, Type, Country] → [Price, Tax].**

**Table 2.1: Library Relation Instance.**

| | **Name** | **Type** | **Country** | **Price** | **Tax** |
|---|---|---|---|---|---|
| t1 | Harry Potter | Book | France | 10 | 0 |
| t2 | Terminator | DVD | USA | 40 | 0.08 |
| t3 | Harry Potter | Book | France | 10 | 0 |
| t4 | Armani Suit | Clothing | UK | 500 | 0.05 |
| t5 | Armani Slacks | Clothing | UK | 250 | 0 |
| t6 | Star Wars | DVD | UK | 25 | 0 |
| t7 | Terminator | DVD | USA | 25 | 0.08 |
| t8 | Prada Shoes | Clothing | France | 500 | 0.05 |
| t9 | Harry Potter | Book | France | 10 | 0.05 |
| t10 | Harry Potter | Book | France | 10 | 0 |
| t11 | Prada Shoes | Clothing | France | 200 | 0.05 |

Intuitively, we can recognize the following inconsistencies:

1. In tuple $t_9$; the entire Harry Potter books sell in France don't have any tax rate, but in $t_9$ we notice that this tuple has 0.05 tax rates, which violate the semantic constraint.
2. In tuple $t_7$; there are two different prices to the same item in same country, which means that one of these tuples violates the semantic constraint.

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

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

- $\varphi_1$ :( [Name, Type="Book", Country="France"]→ [Price, Tax=0]).

- $\varphi_2$ :( [Name, Type, Country="USA"]→ [Price, Tax]).

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

The first **CFD** ($\varphi_1$) prevents the user in the library system from violating this rule: if the book sells in France, then no tax rate is added. While as the second **CFD** ($\varphi_2$) means that in USA country, the name and the type of items define the price and the tax for them, which prevent the same item to have two different prices. These rules do not violate the **FD** that the relation holds in, but added some consistency and accuracy to the relation.

**Example 2.1:**

The library relation in Table 2.1 satisfies φ1 and φ2, However, tuple t9 violates the pattern tuple tp = (-, Book, France -, 0) in tableau T1 of φ1: t1 [Name, Type, Country] = t2 [Name, Type, Country] $\asymp$ tp (-, Book, France), but t1 [Price, Tax] $\neq$ t2 [Price, Tax]

*φ1* :( [Name, Type="Book", Country="France"]→ [Price, Tax=0]).

*φ2* :( [Name, Type, Country="USA"]→ [Price, Tax]).

**Table 2.2: CFDs hold in Library Dataset.**

| Name | Type | Country | Price | Tax |
|---|---|---|---|---|
| - | Book | France | - | 0 |

| Name | Type | Country | Price | Tax |
|---|---|---|---|---|
| - | - | USA | - | - |

For a pattern tuple tp in Tp (Tableau) shown in Table 2.2, we define an instantiation ρ to be a mapping from tp to a data tuple with no variables, such that for each attribute A in X ∪ Y, if tp [A] is '-', ρ maps tp [A] to a constant in dom (A), and if tp [A] is a constant 'a', ρ maps tp [A] to the same value 'a'.

For example, for tp [A, B] = (a,-), one can define an instantiation ρ such that ρ (tp [A, B]) = (a, b), which maps tp [A] to itself and tp [B] to a value 'b' in Dom (B). Obviously, for an attribute A occurring in both X and Y, we require that ρ (tp [$A_L$]) = ρ (tp [$A_R$]). Note that an instantiation ρ may map different occurrences of '-' in tp to different constants; e.g., if tp [A, B] = (-, -), then ρ (tp [A, B]) = (a, b) is well-defined if a ∈ Dom (A) and b ∈ Dom (B).

A data tuple t is said to match a pattern tuple tp, denoted by t ≍ tp, if there is an instantiation ρ such that ρ (tp) = t. For example, t [A, B] = (a, b) ≍ tp [A, B] = (a, -). An instance I of R satisfies the CFD ϕ, denoted by I |= ϕ, if for each pair of tuples t1, t2 in the instance I, and for each tuple tp in the pattern tableau Tp of ϕ, if t1[X] = t2[X] ≍ tp[X], then t1[Y] = t2[Y] ≍ tp[Y]. That is, if t1[X] and t2[X] are equal and in addition, they both match the pattern tp[X], then t1[Y] and t2[Y] must also be equal to each other and both match the pattern tp[Y].

Intuitively, each tuple tp in the pattern tableau Tp of ϕ is a constraint defined on the set I (φ, tp) = {t | t ∈ I, t[X] ≍ tp[X]} such that for any t1, t2 ∈ I (φ, tp), if t1[X] = t2[X], then (a) t1[Y] = t2[Y], and (b) t1[Y] ≍ tp[Y].

Here (a) enforces the semantics of the embedded FD, and (b) assures the binding between constants in tp[Y] and constants in t1[Y]. Note that this constraint is defined on the subset I (φ, tp) of I identified by tp[X], rather than on the entire instance I. If ∑ is a set of CFDs, we write I |= φ if I |= φ for each CFD φ ∈ ∑. If a relation I |= ∑, then we say that I is clean with respect to ∑. [Wenfei et al., 2008 (1)].

### 2.2.1 Conditional Functional Dependency Discovery

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

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

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

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

## 2.3 Related Work

When talking about the **CFDs** which present a special case of **FD**, we have to give some related works about **FDs** before discussing CFDs.

### 2.3.1 Related Work on Functional Dependencies

The size of the search space is exponential to the number of variables in R. One main issue in the discovery of functional dependencies is to prune the search space as much as possible. Existing algorithms can be classified into three categories: the Candidate generate-and-test approach [Flach and Savnik, 1999] [Hernandez and Stolfo 1995] [Savnik and Flach, 1993], the minimal cover approach [Shilakes and Tylman, 1998] [Mannila and Toivonen. 1997] [St´ephane et al., 2000].

The Candidate generate-and-test approach uses level wise search to explore the search space. It reduces the search space by eliminating Candidates using pruning rules. TANE [Huhtala et al., 1998] and FUN [Novelli and Cicchetti, 2001] [Savnik and Flach, 1993] both are level wise methods. They begin by testing FDs with small left-hand sides and prune the search space as soon as possible. More specifically, both methods are based on partitioning the set of tuples with respect to their attribute values.

Using partitions, TANE and FUN can test the validity of FDs efficiently even for large number of tuples. They search the set containment lattice in a level wise manner. By computing closure of Candidates in level k, the FDs in this level are discovered, and results from level k are used to generate Candidates in level k +1. The difference among the algorithms TANE and FUN is that they use different pruning rules to eliminate Candidates. The minimal cover approach discovers the minimal cover of the set of FDs given a database.

FDEP [Flach and Savnik, 1999] consists of three algorithms: bottom-up algorithm, bi-directional algorithm and top-down algorithm. The experiment showed the bottom-up method is more efficient. For bottom-up method, it first

gave two hypotheses: positive cover (the dataset that FD holds) and negative cover (the dataset that FD does not hold). In second step, it computed the maximum negative cover (all possible dataset that FD does not hold); next this approach iterates negative cover again and only considers the dataset that the least general violated FD to get the minimum cover. Finally, by repeating specialize negative dependencies, the positive cover would be constructed, and then the FDs can be obtained from this cover.

FastFDs [Wyss et al., 2001] and Dep-Miner [St´ephane et al., 2000] discover FDs by considering couples of tuples, i.e. agree sets. First, a stripped partition database is extracted from the initial relation. Then, using such partitions, agree sets are computed and maximal sets are generated. Thus, a minimum FD cover according maximal sets is found. FastFDs differs from Dep-Miner only in that Dep-Miner employs a levelwise search, whereas FastFDs use a first-depth search strategy. Formal concept analysis approach discovers functional dependencies from the view of formal concept analysis. By considering the relationship between relational database theory and formal concept analysis [Demetrovics et al., 1992], the functional dependencies hold in a database can be extracted by using pre-defined formal concept analysis closure operator.

In [Lopes et al., 2002], Lopes et al. conclude that the qualitative comparison between DepMiner (or FastFDs) and TANE (or FUN) is more difficult because the approaches widely differ. The drawback of the former is the time-consuming computation of agree sets since it is quadratic with respect to the number of tuples in the relation. The drawback of the latter is their heavy manipulations of attribute sets and the numerous tests which have to be performed.

FD_ Mine approach [Yao et al., 2002] belongs to generate-and-test approach. FD_Mine differs from TANE or FUN in that more effective pruning rules are identified such that a faster and more efficient algorithm is designed for mining FDs from data.

### 2.3.2 Related Work on Conditional Functional Dependencies

A few works are done on **CFDs**, but this problem is still open to the data cleaning and database researchers, in Maher [Arenas and Bertossi, 1999] constraint functional dependencies presented, which is the backbone of **CFDs**, and the same author gave a modified form to Armstrong axioms for conditional functional dependencies in [Bohannon et al., 2005], these axioms present a minimal set of inference rules for **CFDs**.

W.Fan, et.al [Wenfei et al., 2008 (1)], propose a class of integrity constraints for relational databases, referred to as conditional functional dependencies (**CFDs**), and study their applications in data cleaning.

To be able to find which tuples in a relation violate the semantic of relation, you first need to discover the CDFs rules and then compare it with violated relation; a few works are presented to discover CFD. In [Medina and Nourine, 2008] the authors propose an algorithm for discovering CFDs based on levelwise search on the lattice to find all possible constraints on the relation; another approach is presented by Chiang and Miller [Chiang and Miller, 2008] which is again a levelwise search algorithm but has additional pruning rules to filters the Candidates CFDs and reduce its numbers. Finally in [Golab et al., 2008] the authors present an approach for generating the Tableau and give some criteria for classification the good Tableau.

W.Fan, et .al [Wenfei et al., 2008 (3)] present SEMANDEQ, which is a prototype system for improving the quality of relational data, based on conditional functional dependencies, W.FAN, et al [Wenfei et al., 2008 (1)], proposed a frame work for improving data quality concern on consistency and accuracy, by modifying the relation $D$ that is inconsistence to $D'$ which is satisfied the constraint and minimally differs from D and ensure that is accurate.

# CHAPTER THREE

# CFD_MINE ALGORITHM

## 3.0 Overview

Having defined the necessary basics of **CFDs** in Chapter 2 earlier, this chapter describes the algorithmic details for mining the minimum set of **CFDs** from a relational database.

Our approach, ***CFD_Mine*** is a levelwise search algorithm for mining the **CFD** Rules, which means that each ***Candidate*** (element on the lattice) at level *k* is used to discover the results at level *k+1*. Our approach has multi pruning phases, which filter the discovered Rules, to finds a set of minimum **CFDs** and equivalent to another set of **CFDs** discovered by another approach.

To find all Conditional Functional Dependencies according to the definition above, we search through the space of *non-trivial* dependencies, and ***CFD_Mine*** faces two costs:

1) The cost of searching the rule space
2) The cost of visiting the relational dataset to calculate the required partitions for the rules.

The dominant factor is the combinatory complexity of searching a space related to the power set lattice of the set of attributes. An example of such a lattice for a dataset with five attributes shown in next Figure 3.1.

## 3.1 CFD_Mine Algorithm

***CFD_Mine*** approach follows a *breadth first* search strategy, and performs a *level-wise* search in the lattice for finding the partitions to the Candidates and for generating the **CFDs** between adjacent levels, top-down search in the lattice starts from singleton sets and proceeds upwards level-wise in the lattice, searching bigger sets.

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

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

- ***φ:* [ *q* = *xi*, *p* = *ø* ] →[ *vi* ].**

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

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

- ***φ:* [ *q* = *B*, *p* = *ø* ]→ [ *A* ].**

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

- ***φ:* [ *q* = *xi*, *p* ] → [ *vi* ] .**

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

- ***φ:* [ *q* = *A*, *p* = *B* ] →[ *C* ] ,** *or*
- ***φ:* [ *q* = *B*, *p* = *A* ] →[ *C* ] ,** *or*
- ***φ:* [[ q = *B, A], p* = *ø* ] →[ C ].**

**Table 3.1: Balloon Database.**

| | **Color** | **Size** | **Act** | **Age** | **Inflated** |
|---|---|---|---|---|---|
| $t_1$ | YELLOW | SMALL | STRETCH | ADULT | T |
| $t_2$ | YELLOW | SMALL | STRETCH | CHILD | T |
| $t_3$ | YELLOW | SMALL | DIP | ADULT | T |
| $t_4$ | YELLOW | SMALL | DIP | CHILD | F |
| $t_5$ | YELLOW | SMALL | DIP | CHILD | F |
| $t_6$ | YELLOW | LARGE | STRETCH | ADULT | T |
| $t_7$ | YELLOW | LARGE | STRETCH | CHILD | T |
| $t_8$ | YELLOW | LARGE | DIP | ADULT | T |
| $t_9$ | YELLOW | LARGE | DIP | CHILD | F |
| $t_{10}$ | YELLOW | LARGE | DIP | CHILD | F |
| $t_{11}$ | PURPLE | SMALL | STRETCH | ADULT | T |
| $t_{12}$ | PURPLE | SMALL | STRETCH | CHILD | T |
| $t_{13}$ | PURPLE | SMALL | DIP | ADULT | T |
| $t_{14}$ | PURPLE | SMALL | DIP | CHILD | F |
| $t_{15}$ | PURPLE | SMALL | DIP | CHILD | F |
| $t_{16}$ | PURPLE | LARGE | STRETCH | ADULT | T |
| $t_{17}$ | PURPLE | LARGE | STRETCH | CHILD | T |
| $t_{18}$ | PURPLE | LARGE | DIP | ADULT | T |
| $t_{19}$ | PURPLE | LARGE | DIP | CHILD | F |
| $t_{20}$ | PURPLE | LARGE | DIP | CHILD | F |

Before exploring the pseudo code of ***CFD_Mine*** algorithm, we will give some information about the dataset used to explain our approach; we used a database called **Balloon** Dataset as shown in Table 3.1, which is located in **UCI** (Machine Learning Repository) [UCI, 2008], and present a set of trousers with different characteristics, and has *five* attributes and *twenty* tuples.

The semi-lattice in Figure 4.1, illustrates the search space of an exhaustive algorithm for finding the Rules for five attributes. It shows all possible nonempty combinations of the five attributes (*Color, Size, Act, Age* and *Inflated*).

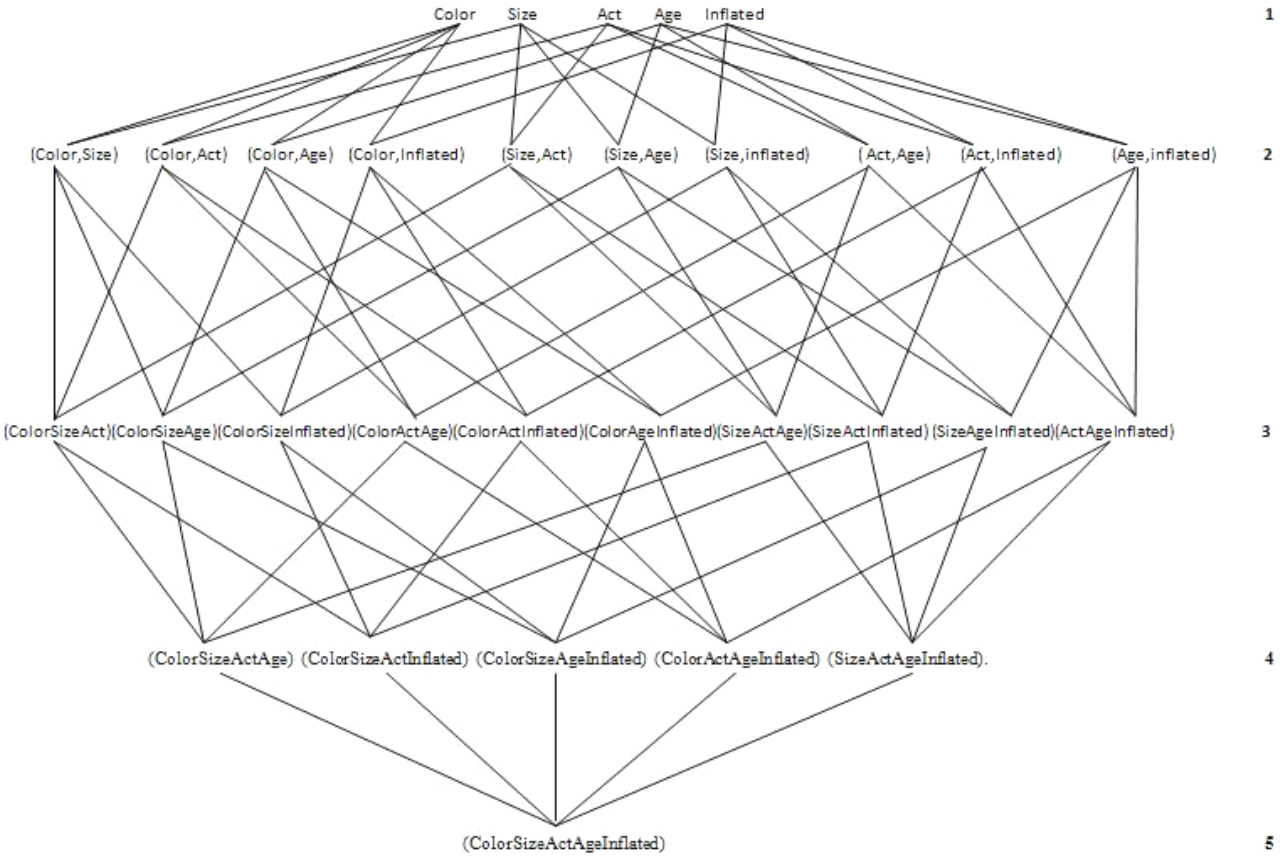


**Figure 3.1: CFD_Mine levelwise semi-lattice.**

***CFD_Mine*** algorithm as Figure 3.2 shows is an Object Oriented algorithm (OO), which means that the Main algorithm calls different **procedures,** each one has its own **Functionality**, and the result of each one of the procedures comes back to the main algorithm, and uses in the next procedure.

**CFD_MINE Algorithm (r (U))**
Input: A relation r (U) over U = {$v_1$...$v_m$}
Output: A set of **FDs** and **CFDs** over r (U).
{
Initialize variables step:
1. CF = ø;
2. $C_1$ = U;
3. SingletonCalculatePartition (C1, r (U));
Iteration step:
4. **while** |Ck| > 0 **do**
5. {
6. k = k + 1;
7. AprioriGen ($C_{k-1}$);
8. CalculatePartition ($C_k$, r (U));
9. CF ∪ ObtainCFDs ($C_{k-1}$, $C_k$);
10. MinimalCover (CF);
11. }
12. **return** (CF);
}

**Figure 3.2: CFD_Mine pseudo code.**
Where,

**CF** **:** variable to store all **CFDs** discovered during the algorithm progress.
**$C_1$** **:** variable to store singleton Candidates attribute in relation.
**$C_k$** **:** variable to store all results comes from calling the sub algorithms.
**k = 1** **:** variable present the level; where the algorithm works on.

The procedures called by the ***CFD_Mine*** algorithm are *SingletonCalculatePartition*, *CalculatePartition*, *AprioriGen*, *ObtainCFDs*, and *PartitionMinimalCover*.

### 3.1.1 Generate Next Level Candidates.

The ***AprioriGen*** algorithm in Figure 3.3 generates all possible Candidates at level *k* from the Candidates at level *k-1*. For example in Table 1 given $C_1$ = {Name, Type, Country, Price, and Tax}, by applying *AprioriGen* procedure, the results at *second* level which stores in $C_2$ is = {(Name, Type), (Name, Country), (Name, Price)…etc}.

**AprioriGen ($C_{k-1}$)**
{
1. $C_k = \emptyset$;
2. **for each** $\{y, z\} \subseteq C_{k-1}, y \neq z$ **do**
3. $x = y \cup z$;
4. **if for each** $A \in x, x \setminus \{A\} \in C_{K-1}$ **then**
5. $C_k = C_k \cup \{x\}$;
6. **return** $C_k$;
}

**Figure 3.3: *AprioriGen* Algorithm.**

Where,

**z, y** : Attribute set at level *k*.
**x** **:** The new value at level *k+1*.

**Example 3.1:**

First of all, the ***AprioriGen*** algorithm takes the *singleton* elements (Candidates), which are the attributes names in the relation and stores them in $C_1$, as shown in Table 3.2.

**Table 3.2: The Candidates in $C_1$.**

| Level | Candidates |
|---|---|
| $C_1$=U | {Color, Size, Act, Age, Inflated}. |

At level *two* the ***AprioriGen*** uses the Candidates available in $C_1$ to finds the Candidates in $C_2$, as in Table 3.3.

**Table 3.3: The Candidates in $C_2$.**

| Level | Candidates |
|---|---|
| $C_2$ = | { (Color, Size) , (Color, Act) , (Color, Age) , (Color, Inflated) , (Size, Act) , (Size, Age) , (Size, Inflated) , (Act, Age) , (Act, Inflated) , (Age, Inflated) }. |

## 3.1.2 Computing Partitions

**Definition** (Partitions):

Two tuples *t* and *u* are equivalent with respect to a given set X of attributes , if *t* [A] = *u* [A] for all A in X . Any attribute set X partitions the tuples of the relation into equivalence classes. We denote the equivalence class of a tuple t ∈ r with respect to a given set X ⊆ R by [t] X, i.e. [t] X = {u ∈ r | t [A] = u [A] for all A ∈ X}. The set ΠX = {[t] X | t ∈ r} of equivalence classes is a partition of r under X.

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

**Example 3.2:**

From data available in Table 2.1 , suppose $X_1$ = Name , then $\Pi_{Name}$ = {{1, 3, 9, 10},{2, 7},{4}, {5}, {6}, {8,11}}, for $X_2$ = Country, $\Pi_{Country}$ = {{1,3,8,9,10,11}, {2,7}, {4,5,6}}, and for $X_3$ = (Name, Country) , $\Pi_{Name, Country}$ = {{1,3,9,10}, {2,7}, {4}, {5}, {6} , {8,11}}. The cardinality to each attribute set presents the number of equivalence classes in their partitions. For example, $|\Pi_{Name}|$ = 6, and $|\Pi_{Name, Country}|$ = 6. We will use the idea of cardinality for the equivalences in each group.

**Definition** (Stripped Partition):

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

The algorithm below shown in Figure 3.4 presents ***SingletonCalculatePartition***, which finds all same tuples in the attribute set that have the same value in the domain for the single attribute set only.

```
SingletonCalculatePartition (C1, r (U))
{
1.      i , j = ø;
2.              for each t[i ] ∈ dom(attr[A]) do
3.                      for j=0 to Table. length
4.                      If (value [i] = value [j])
5.                              t[j]=t[j] ∪ i ;
6.                              break;
7.      return t[j];
}
```

**Figure 3.4: *SingletonCalculatePartition* Algorithm**

Where,

**t[i] :** The index number of the tuples.
**attr[a]:** The attribute set in the level.
**value :** Value domain available in the tuple of attribute, Dom (attr [A]).
**t[j] :** Array of lists to store the partitions.

**Example 4.3:**

At level *one* , the ***SingletonCalculatePartition*** finds all equivalence classes to the singleton attributes stored in $C_1$= {Color, Size, Act, Age, inflated}, updates the values of these Candidates with their partitions , and finally finds the cardinality to each attribute by counting the number of equivalent classes, as Table 3.4 shows.

**Table 3.4: Partitions of the Candidates in $C_1$**

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

The partitions are not computed from scratch (Lattice) for each attribute set. Instead, when ***CalculatePartition*** works its way through the lattice, it computes a partition as a product of two previously computed partitions (in the previous level), the product of two partitions Π' and Π", denoted by Π'. Π" is the least partitions that refines both Π' and Π".

We compute the partitions ΠX, for each X ∈ R, directly from the database if the value of X =1. Where the Partitions ΠX for X ≥ 2, are computed as a product of partitions with respect to the two subsets of X. Any two different subsets of size |X|− 1 will do, which is convenient for the levelwise algorithm since only the partitions from the previous level are needed.

```
CalculatePartition (Ck, r (U))
{
1.      n, m = ø;
2.      for each (t[y] , t[z]) ⊆ Ck, t[y] ≠ t[z] do
3.              for each part[n] ∈ t[y]
4.              for each part[m] ∈ t[z]
5.              t[x] = (part [n] - part [m]) ∪ (part [m] - part [n]);
6.                              break;
7.      return t[x];
}
```

**Figure 3.5: *CalculatePartition* Algorithm**

Where,

**n, m** **:** Numeric values present the index to the partitions to each Candidate.
**part** **:** Array of tuples, present the partitions.

**Example 4.4:**

At level *two*, the ***CalculatePartition*** finds all equivalence classes for the Candidates attribute set stored in $C_{2,}$ update the values of these Candidates in $C_2$, and finally finds the cardinality to each attribute by counting the number of equivalences classes, as shown in Table 3.5.

**Table 3.5: Partitions of the Candidates in $C_2$.**

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

### 3.1.3 Searching for Rules

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

The following steps illustrate how the procedure ***ObtainCFDs*** generates the **CFDs**, and how its sub procedures work.

***First Step*** : ***ObtainCFDs*** algorithm receives two complete levels and compares each element at level $C_{k-1}$ with each *related* element in level $C_k$ , only if the Candidate element at level $C_{k-1}$ is a portion of the Candidate element in $C_k$ , (i.e. $C_{k-1} \subseteq C_k$ ) .

**Example 3.5:**

If there is an element in the *first* level such as **(Age)**, and there are elements in the *second* level such as **(Size, Age)**, **(Color, Inflated)**, then the algorithm compare the element **(Age)** only with **(Size, Age)**, because the **(Age)** Candidate in the *first* level is a portion of the **(Size, Age)** Candidate in *second* level.

***Second Step***: If the partitions of these elements are *exactly* equal (identical) then there is a Functional Dependency **FD** between them, so the algorithm forms a *nontrivial* Functional Dependency between them.

**Example 3.6:**

If there is an element in *second* level such as **(Act, Age)** and its Partitions are $\Pi_{Act,Age}$ = {{1,6,11,16},{2,7,12,17},{3,8,13,18},{4,5,9,10,14,15,19,20}} , and one of its *related* elements in the *third* level is **(Act, Age, Inflated)**, and its Partitions are $\Pi_{Act,Age,Inflated}$ = {{1,6,11,16} ,{2,7,12,17}, {3,8,13,18}, {4,5,9,10,14,15,19,20}}. These two groups of partitions are identical so there is a **FD** between them, and the discovered **FD** is:

- **FD = [ Act, Age → Inflated ].**

***Third Step***: If the partitions of these elements are not exactly equal (i.e. there is at least one partition shared) then ***maybe*** there is a Conditional Functional Dependency available, so go next to ***IntersectPartitions*** procedure which finds the same equivalence classes (Intersect Partitions) that are equal in both elements.

**Example 3.7:**

If there is an element in *first* level such as **(Age)** and its Partitions are $\Pi_{Age}$ = {{1,3,6,8,11,13,16,18},{2,4,5,7,9,10,12,14,15,17,19,20}}. and one of its related element in the *second* level is **(Age, Inflated)** and its partitions are $\Pi_{Age,Inflated}$ = {{1,3,6,8,11,13,16,18} ,{2,7,12,17} ,{4,5,9,10,14,15,19,20}}.

Therefore, there is a common partition between these two Candidates {1, 3, 6, 8, 11, 13, 16, 18}, the ***IntersectPartitions*** procedure in Figure 3.7 finds this common partition and then ***CreateCFD*** procedure takes this common partition and forms a **CFD** between these two Candidates.

***Fourth Step***: ***CreateCFD*** algorithm receives two Candidates and the shared partitions between them to produces **CFD** Rule in this manner, and try to find an element in the LHS Candidate that contains the same Shared partition found by ***IntersectPartitions*** algorithm, if it is found, then it's a *Condition* Partition or *Constant*, if not then the element is *variable*, and its values are from the domain of its attribute.

And this operation is repeated for the RHS. This idea reduces the number of **CFDs** discovered and gives you direct **CFD** and merges a lot of **CFDs** Rules.

**Example 3.8:**

If we have two Candidates **(Act, Inflated)** and **(Act, Age, Inflated)** and there is a Shared partition (Ωx) between them Ωx = {{3,8,13,18}, {4,5,9,10,14,15,19,20}}, The ***CreateCFD*** algorithm in Figure 3.8 checks the element in the LHS **(Act)** and **(Inflated)** to see which one of them has the same partition in its partitions , in this case (**Act**) has the same common partition but (**Inflated**) doesn't , then the (**Act**) is the *Condition* portion and the (**Inflated**) is the *Variable* portion, and we repeat this operation on the RHS but we check only

the element that is not in the LHS ( i.e. **Age**) and we find that it doesn't have the same partition so it's a *Variable* portion.

- ***φ*: [Act = DIP, Inflated] → [Age].**

```
ObtainCFDs (Ck, Ck-1)
{
1.      F= ø;
2.      for each x in Ck -1
3.              for each vi ϵ U – x+
4.                      if (|Πx| = |Πxvi|)
5.                      F =F ∪ (FD : [x →  vi ])
6.                      else
7.                      Ωx = IntersectPartitions (x, xvi);
8.                              if (Ωx = ø)
9.                              break;
10.                             else
11.                             CreateCFD (Ωx, x, vi);
12.     return Ωx;
}
```

**Figure 3.6: *ObtainCFDs* Algorithm.**

Where,

**x+** **:** The closure of the element x, i.e. the element that x contains it.
**Ωx** **:** Variable to store the share partition between two elements in two levels.

```
IntersectPartitions (x, xvi)
{
1.      n, m = ø;
2.      for each part[n] ∈ x
3.              for each part[m] ∈ xvi
4.                      if((part[n] = part[m] )&&(part_size ≥r))
5.                      Ωx = Ωx ∪ part[m] ;
6.                      break;
7.      return Ωx.
}
```

**Figure 3.7: IntersectPartitions Algorithm.**

Where,

**n, m** **:** Numeric values present the index to the partitions in each Candidate.
**part** **:** Array of tuples, which represent the partitions.
**part_size** **:** Number of tuples in each group.
**r** **:** Threshold value.

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

**Figure 3.8: *CreateCFD* Algorithm.**

Where,

**q** **:** variable to store the *Conditional* Attributes.
**p** **:** variable to store the *Variable* Attributes.

### 3.1.4 Pruning the Discovered CFDs

Our algorithm contains many pruning phases; these phases reduce the number of **CFDs** to be checked and have an effect on the performance of the algorithm as we will see later.

#### 3.1.4.1 Stripped Partitions

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

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

**Example 3.9:**

Let's suppose we want to find the **CFDs** between a Candidate at *fourth* level such as **(Color, Size, Act, Inflated)** and a Candidate at fifth level like (**Color, Size, Act, Age, Inflated).**

we have all of these equivalence classes shared between them {{3},{4,5},{8},{9,10},{13},{14,15},{18},{19,20}}, but we care only for partitions that have more than or equal two tuples (*stripped* partitions) {{4,5},{9,10},{14, 15},{19, 20}}.

#### 3.1.4.2 Merge Similar CFDs

In our approach we added the idea of merge **CFDs** Rules based on the similarity between their attributes, and this idea is presented in the third inference rules in [Wenfei et al., 2008 (1)].

**Example 3.10:**

Between **(Color, Inflated)** and **(Color, Act, Inflated),** we have two equivalence classes {4, 5, 9, 10} and {14, 15, 19, 20}, and each one of these classes gives a distinct **CFD**.

- **Ωx = {4, 5, 9, 10} produces:**
  **φ1: [Act = "STRETCH", Color = "YALLOW"] → [Inflated = "T"].**
- **Ωx = {14, 15, 19, 20} produces:**
  **φ2: [Act = "STRETCH", Color = "PRUPLE"] → [Inflated = "T"].**

If you applied our Dataset on another approach for discovering Conditional Functional Dependencies you will see these two **CFDs** rules, but in our approach you will not see it because we merge it into a single **CFD**.

- **φ: [Act = STRETCH, Color] → [Inflated=T].**

The idea of merging rules based on finding the attributes which have the same value and make it *Condition* such as Act = STRETCH and Inflated = T, are equal between φ1 and φ2 ,but Color has different values between φ1 and φ2 so we make it as *Variable* in the **CFD** rule.

### 3.1.4.3 Minimal Cover for CFDs

Before exploring the Modified algorithm for finding the minimum **CFD**, we will study the inference Axioms for **CFD** and their relations with CFD_Mine.

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

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

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

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

**Figure 3.9: Inference Rules for CFDs [Wenfei et al., 2009].**

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

**Example 3.11:**

In our case there is no Trivial **CFD** between the Candidates for the reason shown above, as follows**.**

- ***φ:*** **[Act = "STRETCH"] → [Inflated="T"].**
- ***φ:*** **[Age = "ADULT"] → [Inflated="T" ].**
- ***φ:*** **[Inflated="F"] → [Act = "DIP"].**
- ***φ:*** **[Inflated="F"] → [Age = "CHILD"].**

**FD2** Extends Armstrong's Axioms of *Transitivity*, and to cope with pattern tuples which are not found in FDs, it employs an order relation $\lesssim$, is defined as follows: For a pair ( η1, η2 ) of constants or '- ', we say that η1 $\lesssim$ η2 if either η1 = η2 = a where a is a constant, or η2 ='-'.

The $\lesssim$ relation naturally extends to pattern tuples. For instance, (a, b) $\lesssim$ (- , b). Intuitively, the use of $\lesssim$ in FD2 assures that ($t_1$ [$A_1$] . . . $t_k$ [$A_K$]) is in the "scope" of tp [$A_1$… $A_K$], (i.e.), the pattern tp [$A_1$. . .A] is applicable.

**Example 3.12:**

Transitive **CFDs** between the Candidates

- **φ = [Act=STRETCH] → [Inflated=T].**
- **φ = [Inflated=F] → [Act=DIP].**

**FD3** Tells us that for a CFD ϕ = (R: [B, X] → A, tp), if tp [B] = '-' and tp [A] is a constant 'a', then ϕ can be simplified by dropping the B attribute from the LHS of the embedded FD. To see this, consider an instance I of R such that I |= ϕ, and any tuple t in I. Note that since tp [B] = '-', if t[X] ≍ tp[X] then t [B, X] ≍ tp [B, X] and t [A] has to be 'a' regardless of what value t [B] has. Thus ϕ entails (R: X → A, tp), and I |= (R: X → A, tp).

**Example 3.13:**

Suppose you have this **CFD**, which have Variable attributes and Condition or constant attributes,

- **φ = [Age=ADULT, Color] → [Inflated=T].**

If we remove the variable attribute from the previous **CFD** it will produce:

- **φ = [Age=ADULT] → [Inflated=T].**

**FD4** deals with attributes of finite domains, which are not an issue for standard FDs since FDs have no pattern tuples. They are given w.r.t. a set Σ of **CFDs**. More specifically, to use this rule one needs to determine, given Σ on a relation schema R, an attribute B in attr(R) with a finite domain and a constant b ∈ dom (B), whether or not there exists an instance I of R such that I |= Σ and moreover, there is a tuple t in I such that t [B] = b. We say that (Σ, B = b) is consistent if and only if such an instance I exists. That is, since the values of B

have finitely many choices, we need to find out for which b ∈ Dom (B), Σ and B = b make sense when put together.

Already this inference rule implied in our merge algorithm, but we add some modification on this rule, as the definition on the **CFD** suppose, there is no **CFD** available without having at least one attribute constant,

**Example 3.14:**

Suppose you have these **CFDs**, which have *Variable* attributes in both sides LHS and RHS,

- **φ = [Color = PURPLE, Inflated = F, Size = LARGE] → [Act = DIP]**
- **φ = [Color = PURPLE, Inflated = F, Size = LARGE] → [Age = CHILD]**

Merging these two **CFDs** produce:

- **φ = [Color = PURPLE, Inflated = F, Size = LARGE]→[Age]**

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

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

1. Each CFD in MCF is of the form (R : X → A, tp) as mentioned earlier,
2. MCF ≡ CF.
3. No proper subset of MCF implies MCF, and
4. For each ϕ = (R: X → A, tp) in MCF, There exists no φ = (R: X → A, tp [X∪A]) in MCF such that X ⊂ X. Intuitively, MCF contains no redundant CFDs, attributes or patterns.

Now, if we applied the inference Axioms for finding the minimum cover set of **CFDs** , we will see that the **FD1** , doesn't have any effect on CFD_Mine , because already the finds the *nontrivial* **CFDs** between adjacent levels.

About the second Axiom **FD2**, maybe there is an equivalence set of attributes between the discovered **CFDs**, and as proposed in [Yao et al., 2002] the equivalence Rules are removed, for instance,

- **[Act=STRETCH] → [Inflated=T].**

- **[Inflated=T] → [Act=STRETCH].**

Then the second one will be removed from the set of the discovered **CFD** Rules.

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

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

After applying these Axioms on a set of **CFDs**, and applying the Minimum Cover algorithm proposed by the authors in [Wenfei et al., 2008 (1)], we will have a minimum set of CFDs.

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

**PartitionMinimalCover** algorithm works as follow, the algorithm chooses all the CFDs that have the same intersect partitions , and then chooses between them the **CFDs** that have the same RHS and have some intersect LHS attributes between them, after that the algorithm applies the inference Axioms which will produce the Minimum set of CFDs.

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

**Figure 3.10: PartitionMinimalCover algorithm.**

**Example 3.15:**

In Figure 3.15**,** If we choose all the discovered CFDs which have the same intersect partition equal **Ωx = {4,5,9,10,14,15,19,20} ,** and choose between them the Rules that have the same LHS ( all of them ) and some RHS (Inflated, available between all of them again ) and applying the inference Axioms (FD3). Then the algorithm will produce single CFD as a minimum between them.

- **φ = [Inflated=F] → [Age=CHILD].**
- **φ = [Inflated=F, Size] → [Age=CHILD].**
- **φ = [Color, Inflated=F] → [Age=CHILD].**
- **φ = [Color, Inflated=F, Size] → [Age=CHILD].**

After applying **PartitionMinimalCover** algorithm, the algorithm will produce:

- **φ = [Inflated=F] → [Age=CHILD].**

**Figure 3.11: Choose implies CFD.**

# CHAPTER FOUR

# EXPERIMENTAL EVALUATION

## 4.0 Overview

In this section, we present an experimental study of CFD_Mine algorithm and implementation software. We investigate the *Utility, Correctness and Accuracy, Scalability* for our program to find out the most correct **CFD** and **FD** in a suitable time.

## 4.1 The Utility of the Approach

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

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

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

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

## 4.2 Accuracy of Discovered Rules

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

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

1. It lets the algorithm produce only the Rules that have this value and above,
2. It reduces the search time for finding the Rules.

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

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

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

***g*** **=** **(the ratio *r* * number of tuples in relation *t*) / 100.**

***g*** **=** **(20 * 400)/100= 80 (tuple/partition).**

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

**Example 4.1:**

In Table 3.1, suppose that we want to find only the **CFDs** that covered 40 % of the relation,

**The value of the threshold** ***r* = 40.**

**The number of the tuples in the relation** **t = 20.**

***g*** **=** **(r * *t*) / 100 = (40 * 20)/100 = 8**

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

**Act =** {1,2,6,7,11,12,16,17},{3,4,5,8,9,10,13,14,15,18,19.20}.

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

**Intersect Partition Ωx =** {1,2,6,7,11,12,16,17} .

**Cardinality of Ωx = | Ωx | =** 8.

**The discovered CFD is:** $\varphi$: [Act = "STRETCH"] → [ Inflated=" T"]

Now, if you set the value of ***r* = 0,** then the algorithm will discover all possible CFDs, this means that if there is intersect partition with cardinality =1, the algorithm will discover CFD to it.

**Example 4.2:**

Between these two Candidates we have two intersect partitions and each one produces its own **CFD,** but each one has different percentage**.**

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

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

**Intersect Partition = Ωx =** {2,7,12,17},{4,5,9,10,14,15,19,20}.

**{2, 7, 12, 17}**: produces :

- **φ = [Age=CHILD, Inflated=T] → [Act=STRETCH]**

With 20% of the tuples in the relation.

**{4, 5, 9, 10, 14, 15, 19, 20}**: produces:

- **φ = [Age=CHILD, Inflated=F] → [Act=DIP]**

With 40% of the tuples in the relation.

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

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

## 4.3 Scalability Experiments

### 4.3.1 Parameters

CFD_MINE was applied on a datasets obtained from the UCI Machine Learning Repository [UCI, 2008]. Our experiments were run using a Dual T2350 INTEL Processor 1.86 GHz (1.86 GHz) with 3GB of memory; we used the **Adult** dataset and **agaricus-lepiota** dataset, and varied the parameter of interest to test its effect on the discovery running time.

### 4.3.2 Scalability on the Number of Tuples

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

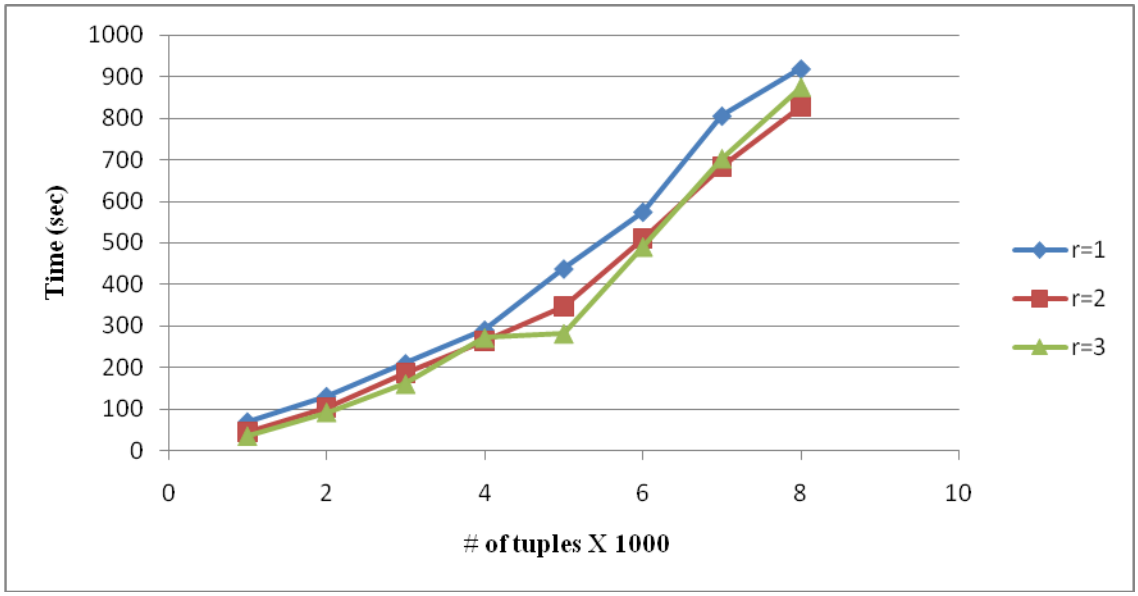


**Figure 4.1: Scalability per Tuples (Adult).**

When increasing the number of tuples , as we saw in Figure 4.1 and Figure 4.2, the algorithm behave semi ***Linearly***, To expound this phenomenon , we talked that our algorithm mainly divided into two main complexity issues .the first one is finding the partition of the Candidates, and because we didn't change the number of Candidates in all of the cases ($2^9$ = 512 Candidates) , then there is no added time to find it , but the little increase of time is because finding more number of partition to each Candidate, which mean more time for find the intersect partitions and more time to generate the **CFDs** Rules .

But we have other variable effects on the scalability, it is the *attribute size*, for example the size of the attribute for **Adult** dataset is much larger than the attribute size of the agaricus-**lepiota** dataset.

Because the attribute size of the **Adult** dataset is larger than the attribute size of the **agaricus-lepiota,** and because we deal with the data as *String* data type, then when the size of String increased the time for merging and separation and other operations on the string done, the time is also increased. So the time for the same number of tuples and attribute for **Adult** data set as shown in Figure 4.1 is larger than the time for **agaricus-lepiota** as shown in the Figure 4.2, but the algorithm still behaves linearly.

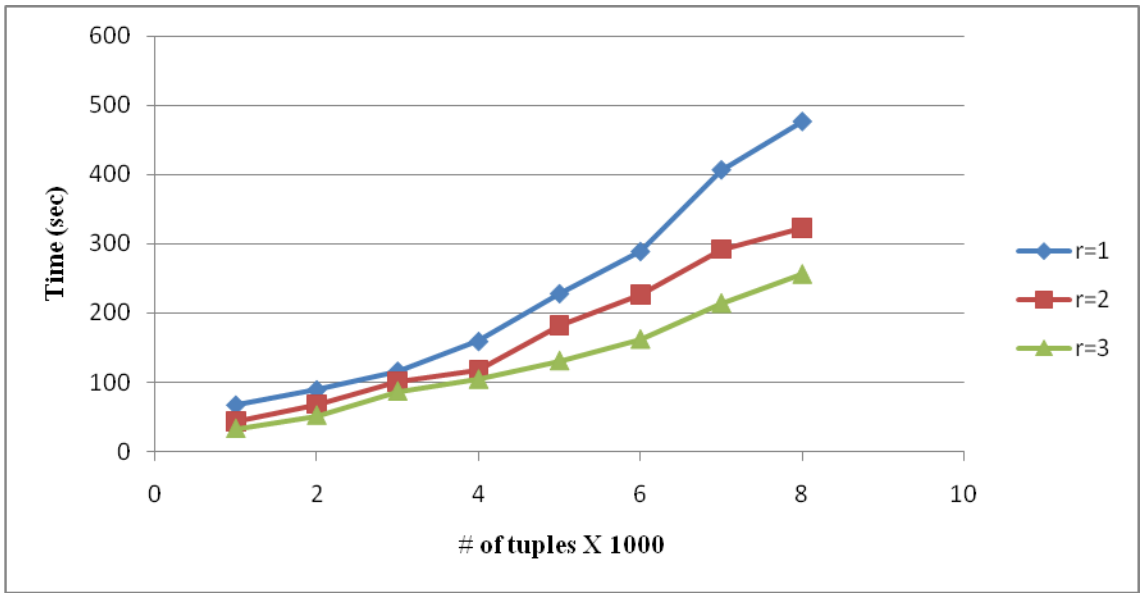


**Figure 4.2: Scalability per Tuples (agaricus-lepiota).**

### 4.3.3 Scalability on the Number of Attributes

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

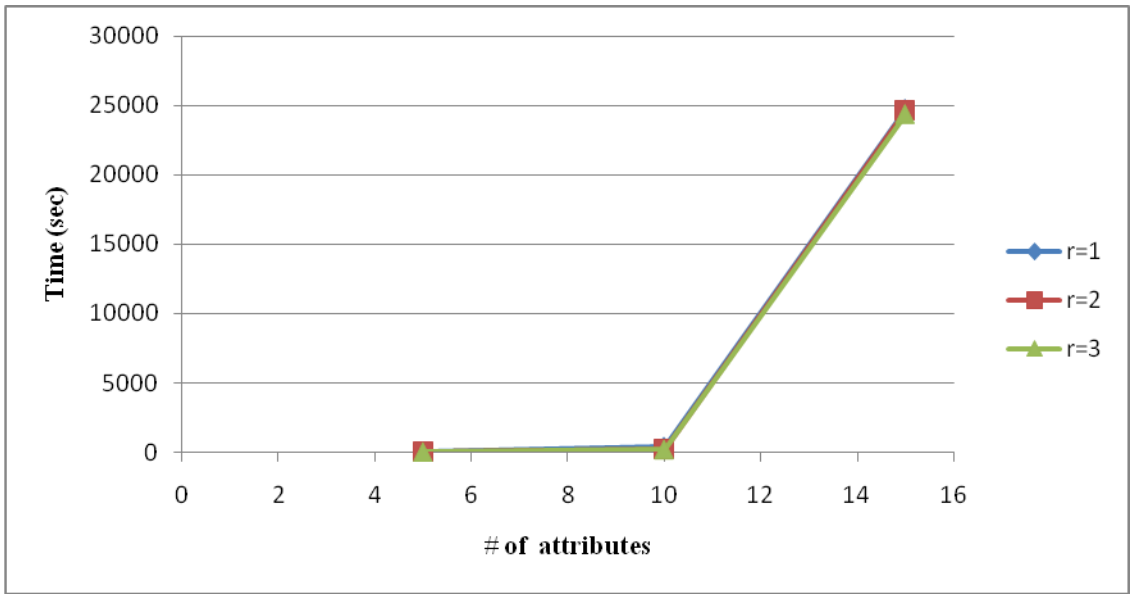


**Figure 4.3: Scalability per Attributes (Adult).**

When increasing the number of attributes as we saw in Figure 4.3 and Figure 4.4, the algorithm behaves ***Exponentially***, to expound this phenomenon, we explain that the time that the algorithm need it to find the partition for dataset with 3 attribute ($2^3 = 8$ Candidates) is much less than the time needed to calculate the partitions for dataset with 15 attributes ($2^{15} = 32768$ Candidates).

These results are identical to what the TANE comes in, because the idea of calculating the partition presented in TANE, and TANE is one of the most efficient approaches for finding the Traditional Functional Dependence.

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

Figure 4.4 confirms the effect of the attribute size on the time of discovering the **CFD** Rules.

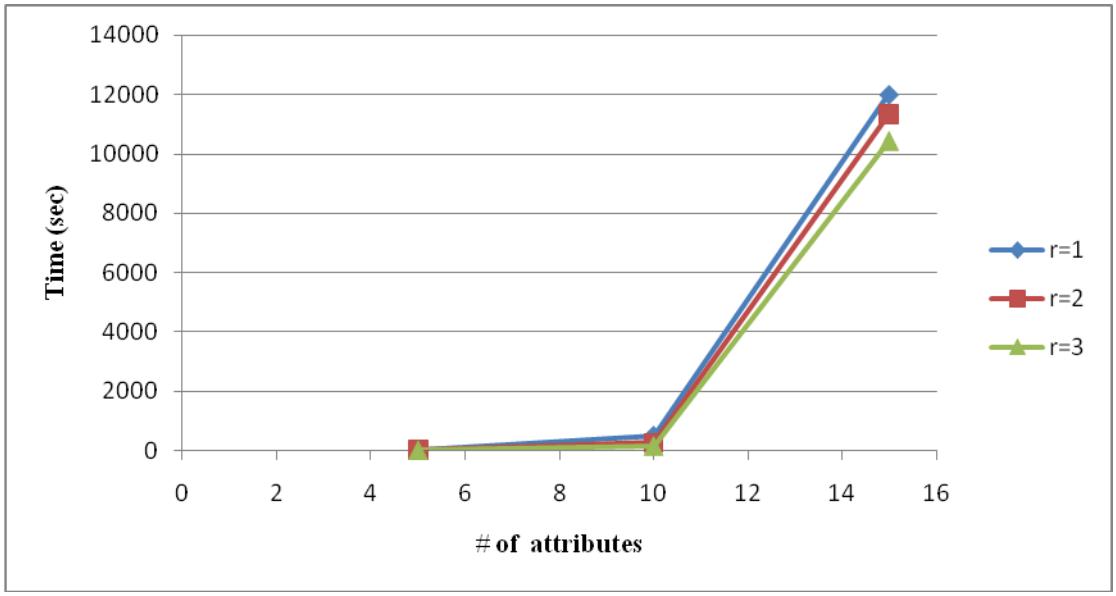


**Figure 4.4: Scalability per Attributes (agaricus-lepiota).**

# CHAPTER FIVE

# CONCLUSIONS AND FUTURE WORK

## 5.1 Conclusions

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

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

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

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

## 5.2 Future Work

For future work a lot of work still needs to be done in Data Cleaning and Data Mining to make it integrated and solid. About my area I think there are a lot of modifications that need to be done too.

First of all our contribution on discovering the Rules needs a second phase of finding the violated tuples, which violate the discovered Rules.

The other idea is making a complete system for cleaning the relation dataset based on both **CFDs** and **FDs**, and modifying the error tuples step by step after discovering the **CFD** rule.

# REFERENCES


Agrawal, R. AND Srikant, R. (1994), *Fast algorithms for mining association rules*. In Proceedings of the 20th International Conference on Very Large Data Bases, J. Bocca, M. Jarke, and C. Zaniolo, Eds. Morgan Kaufmann, pp 487–499.

Andritsos .P, Tsaparas .P, R. Miller. J. (2004). *Information-Theoretic Tools for Mining Database Structure from Large Data Sets*. Proceedings of the 2004 ACM SIGMOD international conference on Management of data. Paris, France, pp. 731-742.

Arenas, M., Bertossi, L. E., and Chomicki, J. (2003). *Consistent query answers in inconsistent Databases*. Theory and Practice of Logic Programming 3, 4-5, pp 393–424.

Arenas. M, Bertossi. L, (1999) .*Consistent Query Answers in Inconsistent Databases*. Proceedings of the 18th, ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. Philadelphia, USA, pp. 68-79.

Bohannon, P., Fan, W., Flaster, M., Rastogi, R. (2005). *A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification*. Proceedings of the 2005 ACM SIGMOD international conference on Management of data. Maryland, USA, pp. 143-154.

Calvanese. D, Giacomo .De.G, and Lenzerini .M. (2001). *Identification Constraints and Functional Dependencies in Description Logics*. Proceedings of the 17th International Joint Conference on Artificial Intelligence. Washington, USA, pp. 155-160.

Carlo Batini, Monica Scannapieco, (2006) *Data Quality: Concepts, Methodologies and Techniques* (Data-Centric Systems and Applications), Springer-Verlag New York, Inc., Secaucus, NJ.

Chiang .F and Miller .R. (2008). *Discovering data quality rules*. Proceedings of the VLDB Endowment, In International Conference on Very Large Data Bases. pp 1166-1177

Chomicki .J and Marcinkowski. J, (2005). *Minimal-change integrity maintenance using tuple deletions*. Inf. Comput.197: pp 90–121.

Demetrovics, J., Libkin, L., and Muchnik, I. B. (1992). *Functional dependencies in Relational databases: A lattice point of view*. Discrete Applied Mathematics, 40: pp 155- 185.

English, L (2000). Plain English on Data Quality: Information *Quality Management: The Next Frontier* // DM Review Magazine. Prieiga per internet : <http://www.dmreview.com/article_sub.cfm?articleId=2073>

Fabien De Marchi, Stephane Lopes, and Jean.Marc Petit, (2002). *Efficient algorithm for mining inclusion dependencies*, In 3th International Conference on Extending Database Technology (EDBT2002) volume 287 of Lecture Notes in Computer Science, pp 464-476 , Springer.

Flach, P.A. and Savnik, A. (1999). *Database dependency discovery: a machine learning approach*. AI Communications, 12(3): pp 139-160.

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

Heikki Mannila and Hannu Toivonen. (1997). *Levelwise search and borders of theories in knowledge discovery*, Data Mining and Knowledge Discovery, 1(3), pp 241-258.

Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. *Efficient algorithms for discovering association rules*, In AAAI Workshop on Knowledge Discovery in Databases KDD, Seattle Washington , pp 181-192

Hernandez, M. A. and Stolfo, S. J. (1995). *The Merge/Purge Problem for Large Databases.* Proceedings of the ACM SIGMOD international conference on Management of data. San Jose, California, USA, pp. 127-138.

Ilyas I. F., Markl. V, Haas. P, Brown .P, and Aboulnaga. A, (2004). *CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies*. Proceedings of the ACM SIGMOD international conference on Management of data. Paris, France, pp. 647-658.

Jef Wijsen. (2003). *Condensed representation of database repairs for consistent query answering*. In ICDT, pp 375–390.

Lopatenko, A. Bravo, L. (2007).*Efficient approximation algorithms for repairing inconsistent databases*. , ICDE 2007. IEEE 23rd International Conference on Data Engineering, pp 216-225

Lopes, S., Petit J.M., Lakhal L., (2002). *Functional and approximate dependency mining: database and FCA points of view*. Special issue of Journal of Experimental and Theoretical Artificial Intelligence (JETAI) on Concept Lattices for KDD, 14(2-3): pp 93-.114.

Maletic, J. I. and Marcus, A. (1999). *Progress Report on Automated Data Cleansing*. Technical Report CS-99-02. pp 1 - 13

Medina Raoul, Nourine Lhouari (2008*). Conditional Functional Dependencies Discovery*. Research Report LIMOS/RR-08-13

Novelli Noel and Cicchetti Rosine. . (2001), *FUN: An efficient algorithm for mining functional and embedded dependencies*. In ICDT '01: Proceedings of the 8th International Conference on Database Theory, pp 189–203, London, UK, 2001. Springer-Verlag. pp 189-203 springer.

Press release, Gartner, Inc., (2005). Quoting Bill Hostmann, Research Director, presenting at Gartner Business Intelligence Summit in London, UK. http://www.gartner.co m/press releases /asset 119071 11.html.

Rahm, E., Do, H.H. (2000). *Data cleaning: Problems and current approaches*. IEEE Data Eng. Bull. 23(4), pp 3–13.

Savnik and P. Flach. (1993). *Bottom-up induction of functional dependencies from relations*.In G. Piatetsky-Shapiro, editor, Knowledge Discovery in Databases, papers from the 1993 AAAI Workshop (KDD'93), pp 174-185. AAAI.

Shilakes, C. C. and Tylman, J. (1998). *Enterprise information portals*. Tech. rep., Merrill Lynch, Inc., New York, NY.

St´ephane Lopes, Jean-Marc Petit, and Lotfi Lakhal. (2000). *Efficient discovery of functional Dependencies and Armstrong relations*. In EDBT ’00: Proceedings of the 7th International Conference on Extending Database Technology, London, UK, Springer-Verlag, pp 350–364.

UCI, (10, 11, 2008), Machine Learning Repository, www.uci.edu.

W. Fan, F. Geerts, and X. Jia. (2008) *SEMANDAQ: A data quality system based on conditional functional dependencies*. In Proc. VLDB, demo.

W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. (2008) *Conditional functional dependencies for capturing data inconsistencies*. ACM Trans. Database Syst., 33(2),

W. Fan, X. Jia, and F. Geerts, (2008). *A Revival of Integrity Constraints for Data Cleaning*, Proceedings of the VLDB Endowment, Volume 1, pp 1522-1523.

Wenfei Fan, Floris Geerts, Laks V. S. Lakshmanan, Ming Xiong,(2009) *Discovering Conditional Functional Dependencies*. ICDE,pp 1231-1234

Winkler, W. E. (2004). *Methods for evaluating and creating data quality*. Information Systems 29, 7, pp 531–550.

Wyss, C., Giannella, C., and Robertson, E.L. (2001). *FastFDs, A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances*. In Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2001), pp. 101-110.

Yao, H., Hamilton, H.J., and Butz, C.J. (2002). *FD Mine: Discovering functional Dependencies in a database using equivalences.* In Proceedings of the 2nd IEEE International Conference on Data Mining, Maebashi City, Japan, pp. 729-732.

Yka Huhtala, Juha Kinen, Pasi Porkka, and Hannu Toivonen. (1998) *Efficient discovery of Functional and approximate dependencies using partitions*. In ICDE, pp 392–401.