CHAPTER 4
MINING OF FUNCTIONAL DEPENDENCY EXTENSIONS

4.1 INTRODUCTION

In the conventional relational databases, Functional Dependencies (FDs) capture semantic constraints existing in data. Although FDs are used as primary tools in enforcing data consistency, a richer semantics is required to capture constraints that are specific to the application domain. In order to resolve inconsistencies and to tolerate data uncertainty inherent in real-life data, an appropriate set of constraints need to be defined over data. This necessitates the need for more general types of constraints and this is achieved by extending FDs in various ways including Conditional Functional Dependencies (CFDs), Fuzzy Functional Dependencies (FFDs) and Matching Dependencies (MDs). This chapter describes information theory based discovery methods for extracting different extensions of FDs from data. As entropy serves as a good measure of information content in data distribution, it is exploited for framing pruning rules to reduce the dependency discovery search space to a larger extent. The proposed discovery methods of FD extensions optimize the search time of dependencies comparatively to the existing approaches.

4.2 CONDITIONAL FUNCTIONAL DEPENDENCY

Data quality is of major concern in today’s information world. Dirty data is harmful to any successful business because inconsistent and
erroneous data negatively impacts the business process and also causes revenue loss. Data cleaning is the essential step in every data integration project. About 30% to 80% of total development time is devoted to data pre-processing step to handle imprecise data in most of the data analysis applications. In recent times, CFDs were introduced for detecting inconsistencies in data and were shown to be more effective than standard FDs in contributing to the improvement of data quality (Bohannon et al. 2007, Batini 2006, Fan et al 2009, Bertossi et. al 2011). CFDs are used to formulate data quality rules based on the bindings of semantically related values and any contravention to the discovered rules can be identified as inconsistencies. As mining traditional FDs is computationally intensive, mining CFDs is still more expensive. In Section 4.2.2, an Information Theoretic CFD Discovery (ITCFD) approach which is scalable to large datasets, is discussed.

4.2.1 Preliminaries

CFDs are the extensions of FDs with conditions which can be leveraged to detect inconsistencies in data. For example, consider a sales relation that has the following attributes: Product_Code(PC), Country_Code(CC), Shipping_Cost(SC), Total_Cost(TC). An instance of sales relation is shown in Table 4.1.

### Table 4.1 Part of Sales Relation

<table>
<thead>
<tr>
<th>PC</th>
<th>CC</th>
<th>SC($)</th>
<th>TC($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01012</td>
<td>34</td>
<td>0</td>
<td>750</td>
</tr>
<tr>
<td>P01014</td>
<td>22</td>
<td>50</td>
<td>940</td>
</tr>
<tr>
<td>P02015</td>
<td>34</td>
<td>0</td>
<td>470</td>
</tr>
<tr>
<td>P01012</td>
<td>55</td>
<td>20</td>
<td>770</td>
</tr>
<tr>
<td>P03217</td>
<td>34</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>P01012</td>
<td>47</td>
<td>25</td>
<td>775</td>
</tr>
</tbody>
</table>
The following semantic constraints (CFDs) are identified in Table 4.1:

\[ cf_1: [CC, PC] \rightarrow TC, (34, _, _) \]

\[ cf_2: CC \rightarrow SC, (34 \parallel 0) \]

\[ cf_3: [CC, PC] \rightarrow SC, (_, _, _) \]

cf1 is an FD that holds on the subset of tuples that satisfies the condition “CC = 34”, rather than on the entire Sales relation. It is generally not considered as an FD in the standard definition since cf1 does not hold true in the entire set of tuples of the relational instance. The constraint cf1 assures that only in the US (Country code 34) the Product code CC determines the Total cost (TC) of an item. For all other tuples that does not satisfy this condition, this CFD will not hold. cf1 has mixture of constant values and unknown variable ‘_’ in its condition pattern and it is an example for variable CFD(Fan et al 2009). cf2 assures that for any product sold in the US (CC=34), the Shipping cost SC is 0. cf2 has only constant values in its pattern \( t_p \) and it is a constant CFD(Fan et al 2009). cf3 specifies that for all tuples, Country code and Product code determines the Shipping cost. Since cf3 is applicable on all the tuples, this CFD is an FD. In other words when the pattern \( t_p \) has all unknown variables, then the CFD is also an FD. cf3 can also be stated with the pattern \( t_p \) as (CC \not= 34, _, _). since for CC=34, the condition SC =0 is already represented by cf2. cf3 with the reformulated pattern is given below as cf4.

\[ cf_4: [CC, PC] \rightarrow SC, (\in 34, _, _) \]

This type of CFD represented by cf4 is called extended conditional functional dependency (eCFD) which is discussed by Bravo et al (2008). CFDs are also a kind of AFD (Huhtala et al 1999, Atoum 2009), because they hold on a subset of tuples instead of the entire set of tuples in the relation. There are a few works like (Cong et al 2007; Fan et al 2008, Golab et al...
2008), contributing to CFD discovery by extending traditional dependency discovery methods. Always CFDs are associated with a support which indicates the minimum number of tuples that are required to satisfy the CFD. The CFD discovery methods extract only minimal CFDs that are k-frequent where k is the support. The minimal k-frequent CFD is defined as follows.

**Minimal k-frequent CFD**

For any relation R with attributes X, Y, Z and a natural number $k \geq 1$, a CFD is called k-frequent if the support of the CFD is at least $k$. Moreover, the minimal CFD $(X \rightarrow Y, \tau_p)$, ensures that there is no CFD of the form $(Z \rightarrow Y, \tau_p)$ satisfied by R for a strict subset Z of X.

The CTANE algorithm (Fan et al 2009) is the one that is closely related to the proposed work. The CTANE algorithm is an extended version of TANE algorithm which is used for conventional FD discovery. The CTANE algorithm usually works well when the number of attributes of a relation is small and the support threshold is large, but it scales poorly when the number of attributes increases. The computational cost of CTANE algorithm is dominated by the cost involved in finding the closure of possible RHS attributes for every candidate in the search space. The proposed ITCFD method computes a support entropy called $H_s$ based on the support specified by the user. The ITCFD approach filters out candidates based on the entropy value, which is a single non-negative real number and thus takes lesser time compared to CTANE algorithm even when the dataset has large number of attributes.
4.2.2 ITCFD Method

The ITCFD method uses attribute entropy and joint entropy between attributes to compare the candidates in the search space. The entropy decreases as the number of duplicate values in a distribution increases. In general, $H_s$ is the entropy of a candidate that has one partition with $k$ number of tuples and the remaining partitions with 1 tuple each in its value distribution. When a relation $U$ with $m$ tuples is considered, the support entropy is calculated using Equation 4.1 given below:

$$H_s = -\left( \frac{m_r}{m} \log_2 \left( \frac{m_k}{m} \right) + m_r \left( \frac{1}{m} \right) \log_2 \left( \frac{1}{m} \right) \right)$$

(4.1)

In Equation 4.1, $m$ is the total number of tuples, $m_k$ is the minimum number of tuples that should have the same constant value for the CFD to get satisfied and $m_r$ is the remaining number of tuples in the relation, i.e., $m_r = m - m_k$.

Any candidate that has partitions with less than $k$ tuples will have entropy higher than the support entropy $H_s$, because entropy increases as the number of duplicates decreases. Candidates with entropy higher than $H_s$ are removed from CFD discovery search space. However, there is no guarantee that an attribute with entropy less than $H_s$ has partitions with $k$ tuples. This is because the entropy of candidates with skewed distribution of values and that of candidates with uniform distribution of values is more or less same. With $H_s$ as the threshold entropy, attribute sets that have entropy higher than $H_s$ can be removed, because such sets will not even have one partition of size $k$.

For attributes with entropy lesser than $H_s$, the size of each partition is checked to see if it is atleast $k$. Once when it is found that the candidate has
atleast one partition of size greater than or equal to k, then the entropy of their supersets at the next higher levels are compared with $H_s$ as shown in Equation 4.2 given below. If there is no partition of size $\geq k$, then the candidate and its supersets are removed from the search space.

As Theorem 1 stated in Chapter 1 indicates, for a FD to hold between any two attributes or attribute sets $X, Y$, the joint entropy $H(X,Y)$ must be equal to $H(X)$. Any CFD need not be strictly true on all the tuples in the relation and the equality need not strictly hold. Even when the difference between $H(X,Y)$ and $H(X)$ is approximately equal to 0 (Equation 4.2), there are possibilities that a CFD holds between attributes $X, Y$.

$$H(X,Y) - H(X) \approx 0$$  \hspace{1cm} (4.2)

Any approximate measure discussed by Giannella and Robertson (2004), can be used to check whether a dependency approximately holds.

### 4.2.3 Pruning Strategies

The proposed work uses two pruning strategies to prune the edges and nodes of level-wise semi-lattice representing the CFD discovery search space. Pruning rule 1 is the original contribution of the proposed work, which prunes the candidates and their superset based on the support entropy $H_s$. Pruning rule 2 is similar to the one used in CTANE and FastCFD algorithms proposed by Fan et al (2009).

**Pruning Rule 1: Discover only k-frequent CFDs.**

Candidates that do not even have one partition with size greater than or equal to $k$ need not be verified for the presence of CFDs. CFDs with
support less than k are not k-frequent and need not be checked. Supersets of such candidates are also not k-frequent. So, candidates with entropy higher than $H_s$ can be removed from the search space.

For example, consider the relation $U$ given in Table 3.1 and the entropy values of level-1 and level-2 candidates that were given in Table 3.2. To find 4-frequent CFDs from the relation $U$, $H_s$ is computed as 2.0. From Table 3.2, it is seen that all level-1 candidates have entropy lesser than $H_s$ and hence they are not pruned. When considering level-2 candidates, it is seen that $H(PQ), H(QR), H(QS), H(QT), H(RS), H(ST)$ are greater than $H_s$ and can be pruned from the search space. Candidates with entropy less than $H_s$ are first checked for partition of size k and then checked for the presence of CFD using Equation 4.2. It is also seen that even though $H(Q)$ is equal to $H_s$, Q does not has any partition of size $\geq k$. Once, when Q is found not to have atleast one partition of size $\geq k$, Q and its supersets are removed from the search space.

**Pruning Rule 2: Discover only minimal CFDs.**

The supersets of the LHS candidate of the CFDs need not be verified further, once when a CFD is satisfied by the candidate. This eliminates redundant CFDs from the set of CFDs discovered.

For the relation given in Table 3.1, once when $H(P)$ is equal to $H(P,R)$, it is seen that $P \rightarrow R$. Supersets of $P$ need not be compared with that of $R$, because $P \rightarrow R$ is the minimal CFD already discovered.
Correctness of the Proposed Approach

Pruning the candidates that have entropy higher than $H_s$ will remove only those candidates that do not have even one partition of size $\geq k$. Candidates having entropy lesser than $H_s$ are checked first for the presence of partition of size $\geq k$ and then checked for the presence of CFD by comparing the entropy of candidates with that of their supersets in the next higher level. This type of checking will not exclude any of the candidates from the CFD discovery search space other than additional checking.

4.2.4 ITCFD Algorithm

A relational instance $r(U)$ with $n$ attributes and $m$ tuples are given as input to the ITCFD algorithm. The support ratio, which is the ratio of support to the total number of tuples in the relational instance, is also given as input in percentage. The set of FDs discovered is the output of the discovery process.

The compute_SupportEntropy() method computes the value of $H_s$ using Equation 4.1. This method takes in $m_k$ and $m_r$ as inputs that are computed from support ratio $k/m$. The computeEntropy( ) method computes the entropy of the candidates passed as parameter. The pruning based on support entropy is applied on candidates at level $i$. CandidateGenerate( ) procedure generates candidates of level $i+1$ from the candidates existing in level $i$. ObtainCFDs() procedure applies theorem 1 to verify the presence of CFD between the candidates of $C_i$ and $C_{i+1}$. Prune_rule2 () is applied to ensure that only minimal $k$-frequent CFDs are discovered by the algorithm.
### ITCFD Algorithm

**Input:** A relational instance \( r(U) \) with \( n \) attributes and \( m \) tuples, Support ratio \( (k/m) \).

**Output:** Set of CFDs holding in \( r(U) \).

**Procedure:**

```plaintext
Begin
Set \( CFD = \Phi, \ C_1 = \{ \text{set of all attributes of } r(U) \} \), \( k = \text{support ratio x m} \),
\( m_k = k, \ m_r = m - m_k \)
\( H_s \leftarrow \text{Compute}_\text{supportEntropy}(m_k, m_r) \)
for each candidate in \( c_{ij} \) from candidate set \( C_i (i=1,2,..,n-1 \text{ and } j=1,2\ldots \binom{n}{i}) \)
begin
ComputeEntropy( \( c_{ij} \) );
Prune_rule1(\( c_{ij} \) );
CandidateGenerate(\( C_{i+1} \) );
CFD= CFD \( \cup \) ObtainCFDs(\( C_i, C_{i+1} \) );
Prune_rule2();
end;
End
```

The time complexity of the ITCFD algorithm varies exponentially with respect to the number of attributes and varies linearly with respect to the number of tuples in the dataset. As the pruning rules used by ITCFD algorithm prunes the search space by a larger extent and the value distribution of candidates in the search space is represented using a single entropy measure, the set closure and set comparison operations used by CTNAE algorithm are not required to test the presence of CFDs.
Experimental Results

The ITCFD algorithm is implemented in java and tested on Intel®
core™ 2 Duo CPU at 2.54 GHz speed with 2 GB RAM running on Windows
XP operating system. The experiments are tested using real life data sets from
UCI machine learning repository. The performance of the proposed approach
is compared with that of CTANE algorithm, which also uses level-wise search
method.

The execution time is measured by varying the number of attributes
and the number of tuples. Figure 4.1 shows the increase in execution time
when the number of attributes (arity) increases. The number of tuples is fixed
as 30K for measuring the variation in execution time with respect to the
number of attributes in the datasets.

![Figure 4.1 Execution Time Vs Number of Attributes](image)

The support entropy based pruning of candidates eliminates as
much of candidates as possible than what CTANE algorithm eliminates and
hence the proposed approach takes lesser time to discover the CFDs.
Figure 4.2 shows the variation of response time with respect to the number of
tuples, when the support ratio is fixed at 60% and the number of attributes is kept as 20. As the number of tuples increases, the computing time of the algorithms also increases.

Figure 4.2 Execution Time Vs Number of Tuples

The computation time taken by the proposed ITCFD method is lesser than that of CTANE algorithm because it does not compute closure sets of the candidates to check the presence of CFDs. Figure 4.3 shows the variation of execution time with respect to support ratio, when the number of tuples are fixed at 30K and the number of attributes as 20.

Figure 4.3 Execution Time Vs Support Ratio(%)
As support ratio increases, the number of candidates qualifying for CFD discovery decreases and large number of candidates are pruned from the search space at the initial stages. This reduces the execution time in detecting the CFDs from data. CTNAE algorithm takes longer time than the proposed method because it checks large number of candidates before eliminating them from the search and also it takes a longer time to converge.

The ITCFD method is tested on three real life data sets from the UCI machine learning repository. The description on the datasets is shown in Table 4.2. The number of CFDs extracted from these datasets when the support ratio is varied is shown in Figures 4.4 to 4.6.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Attributes (arity)</th>
<th>#tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>7</td>
<td>28056</td>
</tr>
<tr>
<td>Mushroom</td>
<td>22</td>
<td>8124</td>
</tr>
<tr>
<td>Adult</td>
<td>14</td>
<td>48842</td>
</tr>
</tbody>
</table>

The chess dataset is considered, because it has small arity and large number of tuples. The number of CFDs present in the Chess dataset is shown in Figure 4.4. Only less number of CFDs are discovered from the chess dataset. The number of FDs present in the chess dataset is 1 and hence the number of CFDs are also less.
The mushroom dataset has a large number of attributes but lesser number of tuples. This dataset is a sample with large arity and small data size. The number of CFDs present in the mushroom dataset is shown in Figure 4.5. The number of CFDs extracted is quite adequate to frame data quality rules.

The Adult dataset is chosen to test a data sample with large arity and larger data size. The number of CFDs present in the Adult dataset is
shown in Figure 4.6. Large number of CFDs are discovered from the Adult dataset.

![Figure 4.6 Number of CFDs in Adult Dataset.](image)

As the support ratio increases, the number of CFDs extracted decreases, because for larger values of \( k \), there may not be sufficient tuples to support CFDs. The ITCFD based CFD discovery algorithm prunes large number of search space and extracts CFDs at a reasonably low cost.

### 4.3 FUZZY FUNCTIONAL DEPENDENCIES

FDs with more expressiveness are required to specify constraints in real-world data that are often imprecise or non-deterministic. All real data cannot be precise because of their fuzzy nature. In general, based on the data type of an attribute domain, attributes are classified as either crisp or fuzzy. An attribute with precise data value is called crisp attribute. For example, Name, City and so on. are attributes with crisp values. An attribute with its data values expressed as fuzzy set is called a fuzzy attribute. For example, Age, Salary, Price, Grade and so on. are fuzzy attributes. Consequently, for comparing attributes of a relation that has both crisp and fuzzy data, typical
equality logic is not suitable. Fuzzy set theory and fuzzy logic proposed by Zadeh(1968) provide mathematical framework to deal with imprecise information.

4.3.1 Preliminaries

The basic definitions required to understand fuzzy functional dependencies are discussed in this Section.

Fuzzy Logic

Fuzzy logic is a form of many-valued logic, which deals with reasoning that is approximate rather than fixed and exact. In contrast to the traditional logic theory, where binary sets have two-valued logic: true or false, fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false.

Fuzzy Set

Fuzzy sets are sets whose elements have degrees of membership. A fuzzy set is a pair \((U, m)\), where \(U\) is a set and \(m\) is a function, mapping every element of \(U\) with a value in the interval 0 to 1 i.e \(U(x) \rightarrow [0,1]\). For each \(x \in U\), the value \(m(x)\) is called the degree of membership of \(x\) in \((U, m)\). For a finite set \(U=\{x_1,x_2,...,x_n\}\), the fuzzy set \((U, m)\) is often denoted by \{ \(m(x_1)/x_1, \ldots m(x_n)/x_n \}\).

Each element of the fuzzy set has an associated degree of membership based on the membership function linked with the attribute domain. For any set \(X\), the membership function usually denoted by \(\mu(X)\) is
any function from $X$ to the real unit interval $[0,1]$. For an element $x$ of $X$, the value $\mu_x(x)$ is called the membership degree of $x$ in the fuzzy set $X$.

**Membership Degree**

The membership degree $\mu_x(x)$ quantifies the grade of membership of the element $x$ to the fuzzy set $X$. The degree of membership is a real number between zero and one, and measures the extent to which the element belongs to the fuzzy set. The value 0 means that $x$ is not a member of the fuzzy set and the value 1 means that $x$ is fully a member of the fuzzy set. The values between 0 and 1 characterize fuzzy members, which belong to the fuzzy set only partially. A typical membership function is shown in Figure 4.7.

![Figure 4.7 Fuzzy Membership Function](image)

**Fuzzification**

The process of transforming crisp values into grades of membership for each fuzzy set is called fuzzification. In a relational table fuzzy attributes are also represented as crisp values. A fuzzy attribute is represented using multiple fuzzy sets based on the linguistic variables that are relevant to the attribute domain. Fuzzy sets are associated with membership functions and they allow the fuzzification of the crisp values of the attributes by estimating the degree of membership with respect to a fuzzy set. Two elements of a fuzzy set are called nearer only if their membership degree is above a specific
threshold. For example, two elements \( x_1, x_2 \in X \) are said to be \( \theta-nearer \) if 
\[ \mu_{x_1}(x_1) \text{ and } \mu_{x_2}(x_2) \geq \theta \]
where \( \theta \) represents the membership threshold. If the 
two elements \( x_1 \) and \( x_2 \) are \( \theta-nearer \) then they are said to be similar and this 
similarity is represented as \( x_1 \approx_\theta x_2 \).

Usually, a functional dependency, denoted by \( X \rightarrow Y \), expresses 
that a function exists between the two sets of attributes \( X \) and \( Y \), and it can be 
stated as follows: for any pair of tuples \( t_1 \) and \( t_2 \), if \( t_1 \) and \( t_2 \) share a common 
value on \( X \), they also have the same value on \( Y \). Such a statement can be 
extended along different lines and fuzzy sets have been used in various ways, 
among which:

- The universal quantifier “for any pair” is weakened into “almost all”.
- The strict equality is replaced by a resemblance relation.
- Precise values are rewritten using linguistic labels related to the attribute.
- The values taken by the sets of attributes \( X \) and \( Y \) may be imprecise.

It appeared that these extended functional dependencies were not 
really able to capture redundancy. Hence, they are not interesting for database 
modeling, but could be used to represent rules or properties in the context of 
data mining. Extracting fuzzy functional dependencies (FFDs) helps us to 
extract meaningful fuzzy rules from the dataset that are hidden otherwise. 
Fuzzy rules help in matching attributes using similarity metrics rather than 
equality functions and can be used as matching rules in entity matching on 
uncertain data. Such fuzzy rules are exploited in decision making and are also 
used in medical field for analyzing various test reports.
**Fuzzy Functional Dependency**

A fuzzy functional dependency, denoted by \( X \rightarrow_{\theta} Y \), is said to exist, if whenever \( t_1[X] \approx_{\theta} t_2[X] \), it is also the case that \( t_1[Y] \approx_{\theta} t_2[Y] \) where \( \theta \) represents the membership threshold and the operator \( \approx_{\theta} \) is used to indicate that \( t_1[x] \) is \( \theta\)-nearer to \( t_2[x] \) and \( t_1[Y] \) is \( \theta\)-nearer to \( t_2[Y] \).

Although various forms of FFDs have been proposed for fuzzy databases, they stressed upon theoretical perspective and only a few mining algorithms are given. The FFD discovery method (DDFFD) proposed by Wang et al (2010) validates and incrementally searches for FFDs from similarity-based fuzzy relational databases. For a given pair of attributes, the validation of FFDs is based on fuzzy projection and fuzzy selection operations. In the proposed Information theory based FFD discovery method (ITFFD), the presence of fuzzy FDs is discovered by computing entropy for the fuzzy columns, which does not require equivalent class refinements used in the dynamic FFD discovery method (Wang et al 2010).

### 4.3.2 Information Theory Based FFD Discovery Method

Fuzzy functional dependencies are used to capture the semantics of similarity relationships between fuzzy attributes. Certain inter-attribute dependencies may be fuzzy in nature and can not be expressed using crisp attributes. Consider a relation \( U(A,B,C) \) that includes a fuzzy attribute \( A \) and crisp attributes \( B, C \). Let it be assumed that the attribute \( A \) with crisp domain is fuzzified into \( n \) different fuzzy sets and as a result the fuzzy columns \( fA_1, fA_2, \ldots, fA_n \) are added to the relation. The entropy of the fuzzy attribute is computed by finding the entropy of each fuzzy column and summing them as shown in Equation 4.3.
\[ H(fA) = \sum_{i=1}^{n} H(fA_i) \]  

(4.3)

The joint probability between a fuzzy attribute A and a crisp attribute B is computed by finding partial joint entropy between the crisp attribute B and each of the fuzzy columns separately. The summation of partial entropies gives the joint entropy between A and B as shown in Equation 4.4.

\[ H(AB) = \sum_{i=1}^{n} H(fA_i B) \]  

(4.4)

After computing attribute entropy and joint entropy between attributes, a level-wise search through the attribute semi-lattice is carried out to discover FFDs by checking the presence of FFD using Theorem 1 stated in Chapter 1. Mining of fuzzy functional dependencies is carried out step by step as described below.

**ITFFD Algorithm**

| Input: A relational instance r(U) with n attributes and m tuples, Membership functions f_1, f_2, f_3, ... f_n, Membership threshold \( \theta \) |
| Output: Set of FFDs holding in r(U) |
| Procedure |
| Begin |
| 1. for each fuzzy attribute \( f_{a_i} \) i=1,2..k |
| \( f_{a_i}[] \) \( \leftarrow \) Fuzzification( \( f_{a_i}, f_i \) ); |
| 2. for each crisp attribute i= 1,2..m-k |
| Compute_entropy(); |
| 3. for each fuzzy column added in step one, |
| Compute_fuzzyEntropy(\( \theta \)); |
| 4. for each fuzzy attribute A and a crisp attribute B |
| \( H(AB) \) \( \leftarrow \) Compute_jointEntropy(AB); |
| if ( \( H(AB) = = H (B) \)) |
| then FFD= FFD U (B \( \rightarrow \_\_\_ \_\_ \ A \) ) |
| End |
Fuzzification($f_a$, $f_i$) is the process of converting fuzzy attribute $f_a$ to a set of fuzzy columns represented as $f_{aci}$ based on the membership function $f_i$. The procedure Compute_entropy( ) computes the entropy of the crisp attributes and the procedure Compute_fuzzyEntropy($\phi$) computes the entropy of fuzzy columns. The values in the fuzzy columns with membership degree greater than or equal to the membership threshold $\phi$ are treated as equal. The Compute_jointEntropy($AB$) function computes the joint entropy of $AB$ using Equation 4.4.

In step 4, the presence of FFD is checked using the equality $H(AB) = H(B)$. When entropy is used to check the presence of functional dependency, set comparisons of equivalence classes are not required. For any two attributes, only their joint entropy and attribute entropy are to be compared to check the presence of functional dependencies. This reduces the computation time.

Illustration

For example, Table 4.3 shows the height of successful players of different sports. The height attribute of the Table may be fuzzified by associating linguistic variables like short, medium and tall which are quantified using the trapezoidal membership functions shown in Figure 4.8.
### Table 4.3 Sample Relation Showing Height of Successful Players

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Player_Name</th>
<th>Sports_Name</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Ivo Karlovic</td>
<td>Tennis</td>
<td>208 cm</td>
</tr>
<tr>
<td>T2</td>
<td>Mike Mentzer</td>
<td>Weightlifting</td>
<td>176 cm</td>
</tr>
<tr>
<td>T3</td>
<td>Michael Phelps</td>
<td>Swimming</td>
<td>194 cm</td>
</tr>
<tr>
<td>T4</td>
<td>John Isner</td>
<td>Tennis</td>
<td>206 cm</td>
</tr>
<tr>
<td>T5</td>
<td>Mario Lemieux</td>
<td>Ice Hockey</td>
<td>180 cm</td>
</tr>
<tr>
<td>T6</td>
<td>Juan Martín del Potro</td>
<td>Tennis</td>
<td>198 cm</td>
</tr>
<tr>
<td>T7</td>
<td>Brett Kimmorley</td>
<td>Rugby league</td>
<td>173 cm</td>
</tr>
<tr>
<td>T8</td>
<td>Franco Columbu</td>
<td>Weightlifting</td>
<td>165 cm</td>
</tr>
<tr>
<td>T9</td>
<td>Mario Ancic</td>
<td>Tennis</td>
<td>196 cm</td>
</tr>
<tr>
<td>T10</td>
<td>Michael Grob</td>
<td>Swimming</td>
<td>201 cm</td>
</tr>
<tr>
<td>T11</td>
<td>Shawn Ray</td>
<td>Weightlifting</td>
<td>170 cm</td>
</tr>
<tr>
<td>T12</td>
<td>Chris Pronger</td>
<td>Ice Hockey</td>
<td>180 cm</td>
</tr>
<tr>
<td>T13</td>
<td>Andrew Johns</td>
<td>Rugby league</td>
<td>174 cm</td>
</tr>
<tr>
<td>T14</td>
<td>Alexey Lesukov</td>
<td>Weightlifting</td>
<td>168 cm</td>
</tr>
<tr>
<td>T15</td>
<td>Marin Cilic</td>
<td>Tennis</td>
<td>198 cm</td>
</tr>
<tr>
<td>T16</td>
<td>Brett Hodgson</td>
<td>Rugby league</td>
<td>175 cm</td>
</tr>
<tr>
<td>T17</td>
<td>Wayne Gretzky</td>
<td>Ice Hockey</td>
<td>183 cm</td>
</tr>
<tr>
<td>T18</td>
<td>Billy Slater</td>
<td>Rugby league</td>
<td>176 cm</td>
</tr>
</tbody>
</table>
The membership function used to assign membership degree to various values of height along the fuzzy dimensions Tall, Short, Medium is shown in Table 4.4.

**Table 4.4 Trapezoidal Function Furnished in Figure 4.8**

<table>
<thead>
<tr>
<th>Membership Function</th>
<th>Membership degree</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x, a, b, c, d)$</td>
<td>0</td>
<td>$x &lt; a \text{ and } x &gt; d$</td>
</tr>
<tr>
<td>$(x - a) / (b - a)$</td>
<td>1</td>
<td>$a \leq x &lt; b$</td>
</tr>
<tr>
<td></td>
<td>$b &lt; x &lt; c$</td>
<td></td>
</tr>
<tr>
<td>$(d - x) / (d - c)$</td>
<td>$c \leq x \leq d$</td>
<td></td>
</tr>
</tbody>
</table>

| Tall(x)             | 0                 | height(x) $\leq 191$ |
|                     | (height (x)-191)/5| 191 < height(x) < 195|
|                     | 1                 | > 195 |

| Medium(x)           | 0                 | height(x) < 177 and height(x) >194 |
|                     | (height (x)-177)/3| 177 $\leq$ height(x) $< 180$ |
|                     | 1                 | 180 $\leq$ height (x) $\leq$190 |
|                     | 194-height(x)/3   | 191 $\leq$ height(x) $\leq$ 194 |

| Short(x)            | 0                 | height(x) < 160 and height(x) >178 |
|                     | (height (x)-176)/3| 175 $\leq$ height(x) $\leq$178 |
|                     | 1                 | height (x) $\leq$ 178 |

The projection of the actual relational table over the attribute height, fuzzified using the membership function is shown in Table 4.5. The
membership threshold $\theta$ to qualify data values as identical can be fixed as any value greater than 0.5.

**Table 4.5 Projection of Relational Table Shown in Table 4.3 on Linguistic Variables Short, Medium and Tall**

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Sports_Name</th>
<th>Height</th>
<th>$\mu_{\text{tall}}$ (Height)</th>
<th>$\mu_{\text{medium}}$ (Height)</th>
<th>$\mu_{\text{short}}$ (Height)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Tennis</td>
<td>208 cm</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td>Weight lifting</td>
<td>178 cm</td>
<td>0</td>
<td>0.33</td>
<td>0.66</td>
</tr>
<tr>
<td>T3</td>
<td>Swimming</td>
<td>194 cm</td>
<td>0.75</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T4</td>
<td>Tennis</td>
<td>206 cm</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T5</td>
<td>Ice Hockey</td>
<td>180 cm</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T6</td>
<td>Tennis</td>
<td>198 cm</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T7</td>
<td>Rugby league</td>
<td>173 cm</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T8</td>
<td>Weight lifting</td>
<td>165 cm</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T9</td>
<td>Tennis</td>
<td>196 cm</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T10</td>
<td>Swimming</td>
<td>201 cm</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T11</td>
<td>Weight lifting</td>
<td>170 cm</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T12</td>
<td>Ice Hockey</td>
<td>180 cm</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T13</td>
<td>Rugby league</td>
<td>174 cm</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T14</td>
<td>Weight lifting</td>
<td>168 cm</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T15</td>
<td>Tennis</td>
<td>198 cm</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T16</td>
<td>Rugby league</td>
<td>175 cm</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T17</td>
<td>Ice Hockey</td>
<td>183 cm</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T18</td>
<td>Rugby league</td>
<td>178 cm</td>
<td>0</td>
<td>0.33</td>
<td>0.66</td>
</tr>
</tbody>
</table>

The Table is partitioned into equivalence classes that include tuple IDs of those tuples that qualify as equal along different linguistic variables associated with the attribute. The relational table is also partitioned based on crisp data values over the crisp attribute Sports_Name and with each linguistic dimension separately. Partitioning of tuples are done by sequentially checking the tuples, but by using hash table. Usage of hash table helps in
getting the frequency count of every distinct value in a particular column faster using which entropy of the column is computed.

\[
\Pi_{(Sports\_Name)}(PLAYERS) = \{ \{ T1, T4, T6, T9, T15 \}, \{ T2, T8, T11, T14 \}, \\
\{ T3, T10 \}, \{ T5, T12, T17 \}, \{ T7, T13, T16, T18 \} \}
\]

\[
\Pi_{(Sports\_Name, short(Height(x)))}(PLAYERS) = \{ \{ T2, T8, T11, T14 \}, \{ T7, T13, T16, T18 \} \}
\]

\[
\Pi_{(Sports\_Name, Medium(Height(x)))}(PLAYERS) = \{ \{ T5, T12, T17 \} \}
\]

\[
\Pi_{(Sports\_Name, Tall(Height(x)))}(PLAYERS) = \{ \{ T1, T4, T6, T9, T15 \}, \{ T3, T10 \} \}
\]

Consider the relational Table PLAYERS and the projected relational table given in Table 4.3 and Table 4.5 respectively. Let us take \( \theta = 0.6 \). The fuzzy attribute entropy and joint entropy of the attributes are computed using Equation 4.3 and 4.4 respectively.

\[
H(\text{Sports\_Name}) = 2.257
\]

\[
H(\text{Sports\_Name} , \text{Tall(Height(x))}) = 0.863
\]

\[
H(\text{Sports\_Name} , \text{Medium(Height(x))}) = 0.430
\]

\[
H(\text{Sports\_Name} , \text{Short(Height(x))}) = 0.964
\]

\[
H(\text{Sports\_Name} , \text{Height(x)}) = 0.863 + 0.430 + 0.964 = 2.257.
\]

It is seen that \( H(\text{Sports\_Name}) \) is equal to \( H(\text{Sports\_Name}, \text{Height(x)}) \) and this equality in entropy values indicate that \( \text{Sports\_Name} \rightarrow_{\theta} \text{Height} \) with a degree of 0.6. The following fuzzy rules could be derived from this fuzzified functional dependency.

**Fuzzy Rule 1:** All successful tennis and swimming players are tall.

**Fuzzy Rule 2:** All successful Weightlifting and Rugby league players are short.

**Fuzzy Rule 3:** All successful Ice Hockey players are medium in height.
4.3.3 Experimental Results

The algorithms ITFFD and DDFFD were implemented using java and tested on the sports Table, which includes values collected from Wikipedia. The sports Table shown in Table 4.3 is extended with the attributes like Age, Weight, and Country etc. to create a dataset of large arity. There are about 1K records in the Table. The data records are duplicated to create data sets of large size. The experiments were run on Intel® core™ 2 Duo CPU at 2.54 GHz speed with 2 GB RAM.

Figure 4.9 shows the precision of the results produced by the ITFFD algorithm. Precision increases as the number of records increases, because more number of records contributes to the qualifying FFDs. The experiment is repeated by varying the membership threshold between 0.6 and 0.9.

![Figure 4.9 Precision Vs Number of Tuples](image)

When the membership threshold increases, the accuracy of the detected rules also increases. When the membership threshold is kept at 0.9, the fuzzified attribute is almost equivalent to crisp attribute and hence contributes to higher accuracy of the results. Figure 4.10 shows the numbers of FFDs extracted from datasets of different sizes. As the size of the dataset
increases, the number of FFDs decreases. This is due to the fact that, the new tuples inserted may invalidate the FFDs discovered. When the membership threshold increases, the number of FFDs discovered decreases.

![Figure 4.10 Number of FFDs Vs Number of Tuples](image)

**Figure 4.10 Number of FFDs Vs Number of Tuples**

Figure 4.11 shows the execution time in milliseconds taken by the algorithm for both fuzzification and for extracting fuzzy functional dependencies.

![Figure 4.11 Execution Time Vs Number of Tuples](image)

**Figure 4.11 Execution Time Vs Number of Tuples**

The execution time of the algorithm decreases, when the threshold increases. The number of FFDs to be verified decreases as the membership threshold increases and hence the algorithm takes lesser time than that with lower threshold values. The execution time taken by the proposed ITFFD
It is seen from Figure 4.12 that the time taken by the proposed approach is 40% less on average compared to the time taken by DDFFD method. Computing equivalence classes and their refinement is not required by the proposed approach and hence takes much lesser time to discover FFDs.

### 4.4 MATCHING DEPENDENCIES

Data are not represented in a consistent way across different data sources. Duplication of entities occurs in integrated data when multiple data sources are merged to provide a uniform view of data and negatively impacts data quality. Identifying matching entities becomes essential to provide a consistent view of data. Functional dependencies, conventionally used for schema design and integrity constraints are in recent times revisited for improving data quality (Fan et al 2007). However, functional dependencies based on equality function, often fall short in entity matching applications, due to a variety of information representations and formats, particularly in the Web data. Several attempts are made to replace equality function of traditional dependencies with similarity metrics. Recently, Matching
dependencies (MDs) are used for data quality applications, such as entity matching (Song & Chen 2009). In order to be tolerant to different information formats, MDs target on dependencies with respect to similarity metrics, as an alternative of equality functions in conventional dependency. MDs are a recent proposal for declarative duplicate resolution.

An MD expresses, in the form of a rule, that if the values of certain attributes in a pair of tuples are similar, then the values of other attributes in those tuples should be matched (or merged) into a common value. For example, the MD $R_1[X_1] \approx R_2[X_2] \rightarrow R_1[Y_1] = R_2[Y_2]$ says that if the tuples $R_1$ and $R_2$ have similar values for attributes $X_1, X_2$, then their values for $Y_1, Y_2$ should be made equal.

4.4.1 Extracting Matching Dependencies

Although functional dependencies and their extension with conditions (CFDs) are very useful in determining data inconsistency and repairing the dirty data, they check the specified attribute value agreement based on accurate match. Obviously, this strict exact match constraint limits usage of FDs and CFDs for entity matching applications, since real-world information often has various representation formats. To make dependencies adapt to this real-world scenario and to be tolerant of various representation formats, matching dependencies that replace exact match of FDs with similarity metric based match are used. MDs are employed as matching rules for data cleaning. Informally, a matching dependency targets on the fuzzy values like text attributes and defines the dependency between two set of attributes according to their matching quality measured by some matching operator. In this study, a hierarchy of data dependencies defined in terms of entropy is used to discover MDs. Matching rules are extracted from discovered MDs and can aid entity matching techniques in identifying the key attributes for matching.
In relational databases, every entity is represented as a record consisting of several fields. Different similarity metrics are used to match different types of fields in the records. String fields are matched by comparing the characters and their sequence in the strings. Edit distance is the most commonly used string similarity measure. For product matching applications, using edit distance to compare product names and descriptions is not adequate. For example, a product named computer, could also be stored as PC or data processor in different data sources. Such string values cannot be matched using exact string matching or edit distance based measures. In the proposed work, synonyms are considered for matching string fields, if exact string matching fails to match. Wordnet ontology is a lexical database and it is used to retrieve synonyms of the string fields to be compared. To compare the semantics of various string fields, ontology specific to the domain under consideration can be used. A dataset like product catalogue may include fuzzy attributes like price, size, weight etc whose values cannot be compared using strict equality. Fuzzy set theory and fuzzy logic proposed by Zadeh (1968) provide mathematical framework to deal with imprecise information.

In practice, MDs are often valid in a subset of tuples and not on all the tuples of a relation. In this study, conditional matching dependencies (CMDs) are used to infer matching rules that are appropriate for product matching (Song et al 2010). CMDs declare matching dependencies on a subset of tuples specified by conditions. For example, the MDs

Manufacturer→Product [Nokia||Cell Phone] and

Manufacturer, Model→Price [Nokia, 5111|| very High]

hold only on a subset of tuples satisfying the condition specified along with the dependency. CMDs have more expressiveness compared to FDs and CFDs and hence they are useful in determining the entity matching rules.
Approximate Functional Dependencies (AFDs) are also generalizations of the classical notion of a hard FD where the value of X completely determines the value of Y not with certainty, but merely with high probability. The MD has properties of CFDs, FFDs and AFDs and could be represented as a combination of all these dependencies and this idea is exploited in the proposed approach to discover MDs from data.

Song et al (2010) have proposed three discovery methods namely Straight-Forward, Tuple-Oriented (TOEM) and Domain Oriented (DOEM) to extract MDs from the given sample data I that includes all possible MDS. The straight forward approach is more primitive and evaluates all candidates in the given relational instance to find all non redundant CMDs. The domain-oriented approach also evaluates the entire instance I and studies the pruning of candidates in a static domain. The tuple-oriented approach incrementally excludes tuple pairs with respect to attribute values of the tuple pairs and introduces dynamic domain according to the currently remaining tuple pairs. But this approach does not scale well with the increase in the number of attributes. Also the fuzzy nature of attributes is not considered by any of the MD discovery methods proposed in the literature.

In this study, an MD discovery approach called ACMD (Approximate Conditional Matching Dependency) algorithm that uses different types of dependencies discovered using information theory measures is proposed. This approach associates support entropy with every attribute participating in a CMD and also associate an approximate measure to detect the presence of dependency between attributes. Two types of approximation is used in the algorithm. One is to compensate for uncertainty in matching similar values using FFDs and the other approximation is to compensate for the fraction of tuples violating FDs using AFDs and CFDs. Experimentally, it is shown that the ACMD algorithm takes less time in discovering MDs from the give sample data.
4.4.2 Information Theory Based MD Extraction Algorithm

Matching entities gathered from multiple heterogeneous data sources place a lot of challenges because the attributes describing the entities may have missing values or the values may be represented using different encodings. The proposed ACMD algorithm also uses a level-wise search method and requires the support ratio, the membership threshold and the approximation measure to compare candidates in the search space. It uses two pruning rules stated below to prune the MD discovery search space.

Pruning Rules

Pruning Rule 1: Support Entropy Based Pruning

Candidates that do not even have one partition with size greater than or equal to k need not be verified for the presence of MDs. MDs with support less than k need not be checked. Supersets of such candidates are also not k-frequent. So, candidates with entropy higher than support entropy can be removed from the search space.

Pruning rule 2: Augmentation Property Based Pruning

The supersets of the LHS candidate of the MDs need not be verified further, once when a CD is satisfied by the candidate. This eliminates redundant MDs from the set of MDs discovered.

ACMD Discovery Algorithm

The function Fuzzification(fa, f) is used for converting fuzzy attribute fa to a set of fuzzy columns represented as fac based on the membership function f. The procedure Compute_entropy( ) computes the entropy of the crisp attributes and the procedure Compute_fuzzyEntropy( )
computes the entropy of fuzzy columns. The values in the fuzzy columns with membership degree greater than or equal to the membership threshold are treated as equal.

**ACMD Discovery Algorithm**

**Input**: Sample training relational instance \( r(U) \) with \( m \) tuples
- Support ratio- \( k/m \)
- Membership Threshold - \( \theta \)

**Output**: Set of Matching Dependencies. (MD)

**Procedure**

1. Initialize \( MD = \emptyset \) // empty set
2. Calculate \( H_s = - \left( \frac{m_k}{m} \log_2 \left( \frac{m_k}{m} \right) + m_i \left( \frac{1}{m} \log_2 \left( \frac{1}{m} \right) \right) \) 
3. for each fuzzy attribute \( fA_i \) \( i=1,2..j \)
   - \( fAc_i[] \leftarrow \text{Fuzzification}( fA_i, f_i \); 
4. for each crisp attribute \( i= 1,2..m-j \)
   - Compute_entropy();
5. for each fuzzy column added in step one,
   - Compute_fuzzyEntropy( \( \theta \));
6. for each fuzzy attribute \( A \) with fuzzy columns \( fAc_i[] \) and crisp attribute \( B \)
   begin
   \( nf = \text{lengthOf}(fAc_i[]) \); // number of fuzzy columns
   Calculate \( H(fAB) = \sum_{i=1}^{nf} H(fAc_iB) \)
   Check if ( \( H(fAB) \leq H_s \)
   then
   if \( H(fAB) - H(B) \approx 0 \)
   then \( MD = MD \cup (B \rightarrow_{\theta} A) \)
   Apply_Pruningrule2();
   else
   Apply_Pruningrule1();
   end
7. Repeat through step 5 for higher level candidates level by level.
The joint entropy of any fuzzy and crisp attribute pair is computed using Equation 4.3. The joint entropy is compared with support entropy $H_s$ to check whether there is sufficient number of tuples supporting the CMD that is being verified. The time complexity of the ACMD algorithm varies exponentially with respect to the number of attributes and varies linearly with respect to the number of tuples in the dataset. As the probability distribution is represented using a single entropy measure, the set closure and set comparison operations are not required to test the presence of MDs. The support entropy based pruning eliminates most of the candidates from the discovery search space at initial stages and hence reduces the execution time of the ACMD algorithm.

4.4.3 Experiments

Experiments were carried out on a 2.16 GHz processors with a 2GB RAM with Windows XP operating system. The implementation of this project is done using java. WordNet ontology is used for extracting the semantics of the string fields under consideration. JAWS is a Java API for WordNet Searching, that provides Java applications with the ability to retrieve data from the WordNet database. It is a simple and fast API that is compatible with many versions of the WordNet database files and can be used with Java.

Dataset

The product catalogue for electronic goods were collected from 20 web sites and consolidated as a dataset with 150 entities. On average, the maximum number of duplicates for an entity is 10. The data records are randomly duplicated and dataset with 10K to 80K records were created. The product catalog has four fields including product_ID, Product_Name, Manufacturer_Name and Price. The fuzzy membership threshold is set as 0.6 and the support ratio is taken as 60%.
Experimental Results

Experiments are carried out to measure the time taken by the ACMD discovery algorithm in extracting ACMDs from data. The ACMD algorithm and the TOEM and DOEM methods (Song et.al 2010) are implemented using Java and run on the product database.

Figure 4.13 shows the time taken to discover MDs from data using the proposed ACMD algorithm and the existing TOEM and DOEM methods.

![Graph showing execution time vs number of tuples](image)

**Figure 4.13 Execution Time Vs Number of Tuples**

The proposed ACMD method is fast in detecting the matching rules, because it scans the database horizontally and also vertically and the dependencies are discovered in single scan. The tuple and domain oriented approaches require entire set of tuples to be scanned multiple times. The two pruning rules used by the ACMD algorithm ensure that only minimal CMDs that have sufficient number of tuples supporting them are discovered and other candidates are eliminated from the discovery space at earlier stages.
4.5 SUMMARY

This chapter describes the extension of traditional FDs that capture more semantics from data in the form of rules. An information theory based approach is presented, that helps in checking the presence of extended functional dependencies with less number of comparisons. Algorithmic approaches are discussed, that dictates step by step procedures to extract various dependencies from data sets. The experimental results show that the proposed approach in the study discovers Extended FDs very fast, because of using the entropy measure.