MINING APPROXIMATE FUNCTIONAL DEPENDENCIES AS CONDENSED REPRESENTATIONS OF ASSOCIATION RULES

by

Aravind Krishna Kalavagattu

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

August 2008

MINING APPROXIMATE FUNCTIONAL DEPENDENCIES AS CONDENSED REPRESENTATIONS OF ASSOCIATION RULES

by

Aravind Krishna Kalavagattu

has been approved

July 2008

Graduate Supervisory Committee:

Subbarao Kambhampati, Chair
Yi Chen
Huan Liu

ACCEPTED BY THE GRADUATE COLLEGE

ABSTRACT

Approximate Functional Dependencies (AFD) mined from database relations represent potentially interesting patterns and have proven to be useful for various tasks like feature selection for classification, query optimization and query rewriting. Though the discovery of Functional Dependencies (FDs) from a relational database is a well studied problem, the discovery of AFDs still remains under explored, posing a special set of challenges. Such challenges include defining right interestingness measures for AFDs, employing effective pruning strategies and performing an efficient traversal in the search space of the attribute lattice.

This thesis presents a novel perspective for AFDs as condensed representations of association rules; for example, an AFD (Model *determines* Make) is a condensation of various association rules like, (Model:Accord *determines* Make:Honda), (Model:Camry *determines* Make:Toyota). In this regard, this thesis describes two metrics, namely *Confidence* and *Specificity* analogous to the standard metrics confidence and support used in association rules respectively. This thesis presents an algorithm called *AFDMiner* for efficiently mining high quality AFDs by employing effective pruning strategies. *AFDMiner* performs a bottom-up search in the attribute lattice to find all AFDs and FDs that fall within the given *Confidence* and *Specificity* thresholds. Experiments on real data sets show the effectiveness of the approach both in terms of performance as well as the quality of AFDs generated.

To amma and taata

ACKNOWLEDGMENTS

I would be restating the obvious when I say Prof. Subbarao Kambhampati 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 an opportunity to work on this topic and for persevering with me as my advisor throughout the time it took me to complete this research. I remain highly obliged to Dr. Yi Chen and Dr. Huan Liu for the useful ideas and feedback they gave as part of my thesis committee.

I am thankful to my colleagues at DB-Yochan, Raju Balakrishnan, Garrett Wolf and Bhaumik Chokshi for their constructive criticism, discussions and friendship. I am greatly thankful to Ravi Gummadi for helping me with experiments and draft during the conference submission. I would also like to thank the entire Planning Group at Yochan for their support and friendship. My special thanks to Kartik Talamadupula for being a great friend and roommate.

I would like to thank my managers at Yahoo! Mail group (Yahoo! Inc) and Digital Home Group (Intel Corp.) for providing me industrial exposure to enhance my experience and also financial support to help me fund part of my Master's studies. My special thanks to colleagues at my internship from ASU, Trunal Bhanse and Sushma Dittakavi for their collaboration, friendship and sharing their lunch generously many a times. Also, I wish to thank my friends Bala Krishna Chennupati, Sumanth Kolar, Anil Dutt, Kranthi Nekkalapudi, Keerthi Valathoru, Pranav Vasishta, Srujan Kanumuri, Arvind Ramachander, Srinath Bodala, Snehith Cherukuri, Prashant Kommireddi, Ravi Palla, Lakshmi Gade, Shreyasee Das, Krupa Navalkar for their friendship, and the juniors from my hometown Yasovardhan, Sreevatsava and Hareesha Tamatam for helping me stay responsible and inspired.

I am highly indebted to my uncle and aunt in Phoenix, Lakshminarayan, Sudha and my little sisters Deepika and Archana for making my stay here comfortable and enjoyable and forget the feeling that I am staying away from my family. I am as ever, especially indebted to my parents Muni Nagulu and Naga Jyothi and brother Anil Kishore, and Anu aunty for their love, encouragement and support throughout my life. Special thanks to Pallavi Daggumati for being a great sister. I am grateful to my uncles Dr. Viswanath Poosala (Vichen mama) and Dr. Suresh Poosala (Seenu mama) for constantly inspiring me since childhood and setting examples to follow their path. I am highly indebted to my grand parents Anjaneyulu Poosala and Venkata Ratnam for treating me as if I am their third son, and providing their love and encouragement. With due respect, I would like to dedicate this thesis to them.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Rule Mining from Databases

Knowledge discovery from databases has proven to be a useful process with a variety of applications and has received tremendous attention across the knowledge management, information retrieval and database communities. Rule mining[13, 26, 1, 16] is a knowledge discovery process that aims to discover a small set of rules in the database and use them for various purposes like classification, prediction, market-basket analysis etc. Especially rules discovered from relational databases have proven to be effective in database design, missing-value prediction, query optimization, information-retrieval style answering in relational DBs etc. Rule mining process is guided by a set of interestingness measures to guide the quality of rules required for the task at hand. In addition to the measures, efficient search strategies and algorithms are a primary need for learning the rules in a cost-effective manner.

1.2 Approximate Functional Dependencies

Over the last ten years, a new research direction has emerged involving rules between attributes, namely, Approximate Functional Dependencies (AFDs). Approximate Functional Dependencies [6, 26, 8, 7] (AFDs) are rules that represent approximate determinations at attribute level. AFDs are of the form ($X \rightsquigarrow Y$), where X and Y are sets of attributes. A classic example of an AFD is, (Nationality$\rightsquigarrow$Language), which shows we can approximately determine the language of a person if we know which country he is from. AFDs represent knowledge of the dependencies between columns in a table. For example, paraphrasing from [8]: an AFD in a table of chemical compounds relating various structural attributes to carcinogenicity could provide valuable hints to biochemists for potential causes of cancer. AFDs discovered from relational databases have proven to be effective in database design [3], missing-value prediction [24], query optimization [9], information-retrieval style answering on relational databases etc [17, 23].

Functional Dependencies (FDs) were prevalent in database literature and the discovery of functional dependencies from a relational database is a well studied problem. But, the discovery of AFDs still remains under explored, posing a special set of challenges. AFDs were initially introduced as an extension to FDs (precisely, as FDs that hold true when the error tuples that fail to be satisfy the dependency are removed). But, with advances in data mining, AFDs expanded their reach into applications demanding a data mining lens. The fundamental challenges in this regard are defining right interestingness measures for AFDs and coming up with scalable algorithms to efficiently mine the AFDs. The interestingness measures affect the quality of the task for which AFDs are used, and scalable algorithms are necessary to achieve a good performance in the learning component for AFDs. Current best approach to mine AFDs [8, 7] uses a standard error measure defined for AFDs (namely, g_3) and tries to adapt the FD mining algorithm with modifications for generating AFDs.

The search process for AFDs is exponential in terms of the number of attributes in the dataset, thus facing a combinatoric cost of searching the rule space. FD mining algorithms fundamentally search for a minimal cover for FDs from the data (as they can generate all other FDs that hold true directly the minimal cover, without revisiting the data again). So they could employ effective pruning strategies. For example, if $A \rightarrow B$ is an FD, then, $(A, C \rightarrow B)$ is considered redundant and in fact all FDs of the form (Z$\rightarrow B$) such that (Z$\supset$A) can be eliminated during search. But, a minimal cover for AFDs is not defined. Previous approach [8, 7] substitutes this by generating only minimal dependencies in case of AFDs. For example, if the algorithm discovers that $A \rightarrow B$ is a valid AFD according to

TABLE I
FRAGMENT OF A CAR DATABASE

ID	Make	Model	Year	Body Style
1	Honda	Accord	2001	Sedan
2	Honda	Accord	2002	Sedan
3	Honda	Accord	2005	Coupe
4	Honda	Civic	2003	Coupe
5	Honda	Civic	1999	Sedan
6	Toyota	Sequoia	2007	SUV
7	Toyota	Camry	2001	Sedan
8	Toyota	Camry	2002	Sedan

the allowed error threshold, it does not check for AFDs of the form (Z$\rightsquigarrow B$) where (Z$\supset$A). But, such AFDs may be interesting for the application and we may prefer them instead of $A \rightsquigarrow B$. In fact, our experience with using AFDs for AIMQ [17], QPIAD [24], QUIC [23] have shown that those rules are more useful than just considering minimal dependencies.[1] Thus, now the search process becomes much more costlier compared to that for FDs and we need additional pruning techniques for AFDs.

Before algorithms for discovering AFDs can be developed, interestingness measures for AFDs must be defined. Consider a sample task of using AFDs as a feature selection step for predicting values for an attribute. We expect the AFD used for this to give us a high prediction accuracy. Also, as there are several AFDs possible with the same attribute on the RHS (dependent attribute), we prefer choosing a best one among them for prediction. We want the interestingness measure for the AFDs to denote this so that we can pick the AFD for which the value is high. One choice for us is to use the error measure as the interestingness measure. But, the problem with just choosing this metric is that, AFDs

[1]Another implication of restricting to only minimal dependencies is that, though FDs are special cases of AFDs (with error zero), we need separate runs to generate AFDs and FDs.

with keys or approximate keys (attributes that become keys when a small fraction of the tuples are removed) as their LHS (determining set) now qualify as good rules (because they require less number of tuples to removed to become FDs). For example, though (VIN$\rightsquigarrow$Make) looks like a better AFD compared to (Model$\rightsquigarrow$Make) using the error metric as the interestingness measure, intuitively (as well as practically) (Model$\rightsquigarrow$Make) is a better AFD. Thus, the distribution of values for the determining set is also important. As another example, consider AFDs being used to rewrite queries based on the values of the attributes in the determining set to retrieve tuples. (QPIAD [24] system uses this approach to retrieve tuples with missing values on the dependent attribute, as autonomous databases do not allow to directly retrieve tuples with null values on specified attributes.) To optimize querying costs, we expect a good throughput per each query posed. Among AFDs with close values for their approximation measure, we prefer ones with lesser number of distinct values for its determining set compared to the ones with more number of distinct values. Thus, the distribution of values in the LHS are considerably important. A performance implication of this is to eliminate such rules early during the search process to make the algorithm run faster.[2]

1.2.1 Overview of the challenges in AFD mining

In the context of the problem of AFD mining, three broad challenges (sub-problems) are identified.

1.2.1.1 Approximation measure: The problem of defining an FD approximation measure is interesting and difficult. The determination of the reasonability of such a mea-

[2] In applications [17], [24] this is partly done as a post-processing step after generating AFDs using TANE, which makes the learning of AFDs even slower

sure is subjective. (like the determination of the interestingness of rules in KDD) The issue is how to measure the degree to which a functional dependency (FD) is approximate. Before algorithms can be developed, measures must be defined quantifying the interestingness of the rule for the application in which they are intended to be used.

1.2.1.2 Pruning Strategies: Pruning involves cutting down the cost of visiting the space of uninteresting rules, based on the rules already visited during search. In case of FDs, minimal dependencies help us pruning a lot of the search space. But, pruning in case of AFDs is not straightforward unless we could possibly exploit the monotonicity of the interestingness measures used.

1.2.1.3 Efficient Traversal of the search space: Rule mining processes usually adopt either a top-down or a bottom-up approach in the lattice[26] of possible combinations. In practise, traversal is tied up with the type of pruning rules applicable and also the cost involved in computing the measures. Especially in the case of AFDs the challenge of deciding whether Bottom-up is better or Top-down is better is tricky, because all the pruning strategies applicable for FDs do not work for AFDs.

1.3 Overview of the approach

Association rules share similar motivations and can be considered as approximate determinations at value-level. For example, (Nationality:USA$\rightsquigarrow$Language:English), (Make:Honda$\rightsquigarrow$Model:Accord) etc are association rules. Association rules have well-defined interestingness measures namely $Confidence$ and $Support$ to solve the above challenges. $Support$ of an itemset denotes the fraction of transactions it occurs (which in our case is nothing but the fraction of database tuples where the set of attribute-value pairs occurs in). $Confidence$ of an association rule is a measure that denotes how likely the head of a rule occurs, given its body. Rules with reasonable values for $Confidence$ and *Support* together perform better compared to the rules with high value for only one of them.

The notion of an AFD is to provide us valuable hints at an attribute level to pick the right attributes so that it works well for a dominant number of underlying values of these attributes. Thus treating the AFDs as condensed roll-ups of many such association rules corresponding to each value combination in the attribute sets will solve our problems. For example, an AFD (Model⇝Make) is a condensation of various association rules like, (Model:Accord⇝Make:Honda), (Model:Camry⇝ Make:Toyota). Analogous to association rules, for an AFD, we need *Confidence* to denote how well we can predict most if not all the values of the dependent attribute, given values of the attributes in the determining set. In addition to this, we define a metric called *Specificity* to denote how good the distribution of values for the attributes forming the determining set is. We also show how confidence is related to the standard error measures of the AFDs, namely g_2 and g_3 which helps us in adapting the existing efficient algorithm to compute them for calculating confidence. We present an algorithm $AFDMiner$ that takes user-specified thresholds on the minimum value for the *Confidence* and maximum value for the *Specificity* metric. Also, through experimental observations, we comment on setting a good threshold value for *Specificity* metric.

Assumption: Our aim is to learn AFDs of the type (X⇝A), where X is a set of attributes and A is a single attribute.

1.4 Thesis Contributions

This thesis presents an approach for mining AFDs from a relational database table. The major contributions of this thesis are as follows:

- Insight into the way of treating AFDs roll-ups of association rules, supported by derivation of two standard metrics g_2 and g_3 from association rules
- Two-metric notion for AFDs *Confidence* and *Specificity* with derivations
- An approach, AFDMiner, to compute high quality AFDs efficiently from a relational table
- Empirical observations on choosing a good threshold value for *Specificity*

1.5 Outline

Related work is given in Chapter 2. Basic notations, necessary definitions plus certain conventions used in the rest of the paper are presented in Chapter 3. In Chapter 4 we layout the intuition behind treating AFDs as condensed representations of association rules, explain and derive the metrics for the AFDs. The types of pruning strategies applicable, principles of traversing the search space and the detailed *AFDMiner* algorithm are presented in Chapter 5. Chapter 6 reports the empirical evaluation of *AFDMiner* in terms of performance and quality, and a discussion on comparing it to TANE algorithm. We conclude with a brief discussion on Conditional Functional Dependencies (CFD) in Chapter 7.

2 RELATED WORK

As AFDs represent knowledge of dependencies between columns in a table, their discovery is potentially valuable to domain experts and knowledge engineers. CORDS [9] is an efficient and scalable tool for automatically discovering statistical correlations and soft functional dependencies (soft FDs) between columns. These dependency statistics are used by the optimizer to compute correct selectivity estimates. CORDS system only mines rules of the form C1⇒C2, where C1 and C2 are attributes, where as our system aims to detect rules with multiple attributes in the determining set. Additionally, CORDS works from a sample, and thus can never assert with complete certainty that discovered soft FD is a functional dependency or not. QPIAD [24] uses AFDs for query rewriting to retrieve tuples with missing values on the queried attribute, and in predicting the values for these attributes in those tuples. QUIC [23], a system to handle imprecision and incompleteness together over autonomous databases, also uses AFDs to rewrite (expand) an input user query into a set of queries to get retrieve relevant tuples. AIMQ is a domain and user independent approach for answering imprecise queries over autonomous Web databases. It uses AFDs to determine the importance of an attribute to decide the relaxation order of attributes in sending more selection queries to the database. All the three systems use TANE's algorithm for detecting approximate dependencies with additional post-processing to remove approximate keys.

FDs have a well-defined minimal cover, and the research on algorithms to mine the FDs concentrated on learning this minimal cover faster. [26] states that existing algorithms can be classified into three categories: the candidate generate-and-test approach, the minimal cover approach [14, 25], and the formal concept analysis approach. In [15], the authors conclude that the qualitative comparison between these approaches is difficult because the approaches widely differ. The size of the search space is exponential to the number of variables in the data. One main issue in the discovery of FDs or AFDs is to prune the search space as much as possible. As we do not want to restrict ourselves to minimal dependencies, and also the *Specificity* metric is most effective for a level-wise search, we opted for a BFS strategy that falls under the category of candidate-generate-and-test approach. 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. [7] and [18] both are examples of such levelwise methods. A theory on pruning strategies is well presented in [19].

[20] introduces a problem of discovering functional determinations that result from rolling up the data to a higher abstraction level, which they call roll-up dependencies (RUDs). RUDs allow attributes to be compared for equality at different levels (similar to OLAP). But, they do not consider the problem of defining quality measures for AFDs, or treating them as condensed association rules as we do.

Fast algorithms for mining association rules from transactions of itemsets are presented in [1]. The pruning we use based on the monotonicity of *Specificity* can be considered similar to the property in Support for frequent item sets. In the latter, the apriori property says no itemset can be frequent if any of its subset is infrequent. In our case, if an attribute-set is a failAttrSet all of its supersets are failAttrSets.

There has been some work on grouping [2] and clustering [12] association rules. But, neither of them considers the task of grouping a set of association rules as AFDs. Kivinen and Mannila [10] defined several measures for the error of a dependency. [6] examines the issue of how to measure the degree to which the functional dependency (FD) is approximate. They define a measure based on information theory and compare the behavior with two other standard measures, g3 and Tau. Though they show theoretical differences between these measures, in practice these differences do not bear themselves out. Also, they do not devise any algorithm that uses the measure to generate AFDs. We feel either of these measures can be treated as alternatives to our confidence measure, but we still need the *Specificity* in addition to it to get good quality AFDs.

To the best of our knowledge, ours is the first approach that both defines measures for AFD (derived from association rules) and also presents the algorithm on generating AFDs according to those measures.

3 DEFINITIONS AND CONVENTIONS

$\mathbf{R}(A_1, A_2, \cdots, A_n)$ be a relation schema, and $\mathbf{r}$ is a relation (single relational table) over $\mathbf{R}$, where n is the number of attributes. t, u denote rows in $\mathbf{r}$ ($t, u \in \mathbf{r}$). X, Y, Z are sets of attributes. A, B, C are individual attributes of $\mathbf{R}$. v_{ik} denotes the k^{th} value of an attribute A_i in $\mathbf{r}$, and $1 \leq k \leq N_i$, where N_i is the number of distinct values for the attribute A_i in $\mathbf{r}$. Thus, A_i:v_{ik} indicates an attribute:value pair.

Association rules were initially introduced in the context of itemsets in transactions. Tuples in a database relation can be viewed as transactions where itemsets are nothing but the attribute-value pairs. Thus, in our case, *Support* of an itemset denotes the fraction of database tuples where the set of attribute-value pairs occurs in. And, $Confidence$ of an association rule has the same interpretation that denotes how likely the head of a rule occurs, given its body. For example, applying the definitions on the database snapshot showed in Table I, $Confidence$(Model:Accord⇝Make:Honda) $= \frac{3}{3} = 1$, where as $Confidence$(Make:Honda⇝Model:Accord) $= \frac{3}{5}$, and the itemset (Make:Honda, Model:Accord) has a support of $\frac{3}{8}$.

Association Rule: ($\alpha \rightsquigarrow \beta$) denotes an association rule such that, α denotes a set of such attribute value pairs, such that each pair is from a different attribute (i.e., no two pairs have the same attribute). And consider β to be a singleton set of an attribute:value pair $A_j : v_{j_y}$ such that they are all different from ones in α.

Association Rule Mining Problem: Given a relation $\mathbf{r}$, the problem of mining association rules [1] is to generate all association rules of the form ($\alpha \rightsquigarrow \beta$) that have support and confidence greater than the user-specified minimum support (called *minsupp*) and the minimum confidence (called *minconf*) respectively.

Determining Set and Dependent Attribute: In an AFD (X⇝ A), X is called the determining set and A is called the dependent attribute.

$\varphi(X \rightsquigarrow A)$ denotes the error measure for the dependency $(X \rightsquigarrow A)$ in a relation. Three such standard measures defined in [10] are as follows:

A pair (u, v) of tuples of r violates the dependency, or is a violating pair for it, if u[X] = v[X] but u[Y] $\neq$ v[Y]. Hence, a dependency holds in a relation if the relation does not contain violating pairs for it. A single tuple 'u' is called violating, if it is a component in some violating pair. G_1 denotes the number of violation pairs of tuples in the relation, G_2 denotes the number of violating tuples in the relation, and G_3 denotes the number of tuples one has to delete to obtain a relation that satisfies the dependency.

The corresponding normalized measures are denoted by g_1, g_2 and g_3 so that the error lies between 0 and 1.

$$\begin{aligned}
G_1(X \rightarrow Y, r) &= |\{(u,v) \mid u, v \in r, u[X] = v[X], \\
&\quad u[Y] \neq v[Y]\}| \\
g_1(X \rightarrow Y, r) &= G_1(X \rightarrow Y)/|r|^2 \\
G_2(X \rightarrow Y, r) &= |\{u \mid u \in r, \exists v \in r : u[X] = v[X], \\
&\quad u[Y] \neq v[Y]\}| \\
g_2(X \rightarrow Y, r) &= G_2(X \rightarrow Y)/|r| \\
G_3(X \rightarrow Y, r) &= |r| - max\{|s| \mid s \subseteq r, s \models X \rightarrow Y\} \\
g_3(X \rightarrow Y, r) &= G_3(X \rightarrow Y)/|r|
\end{aligned}$$

In terms of error measure, Approximate Dependency can be defined as follows:

Approximate Dependency: According to [7], given an error threshold $\varepsilon, 0 \leq \varepsilon \leq 1$, we say that $X \rightsquigarrow A$ is an approximate dependency if and only if $\varphi(X \rightsquigarrow A)$ is at most ε.

Minimal Pass Rule: $X \rightsquigarrow C$ is a minimal pass rule (written minP) provided that $\varphi(X \rightsquigarrow C) \leq \varepsilon$ and for all Y, $Y \subset X, \varphi(Y \rightsquigarrow C) \leq \varepsilon$ does not hold.

4 CONDENSING ASSOCIATION RULES INTO AFDS

In this section, we show how AFDs mined from a relational table are condensed representations of association rules, define the metrics and explain them.

Make	Model	
Honda	Accord	$\alpha_1 \rightsquigarrow \beta_1$
Honda	Accord	
Honda	Accord	
Honda	Civic	$\alpha_1 \rightsquigarrow \beta_2$
Honda	Civic	
Toyota	Sequoia	$\alpha_2 \rightsquigarrow \beta_3$
Toyota	Camry	$\alpha_2 \rightsquigarrow \beta_4$
Toyota	Camry	

Fig. 1. A simple logistics example

Using the notations defined in Chapter 3 ($\{A_{i_1}, A_{i_2} \ldots A_{i_m}\} \rightsquigarrow A_{i_{m+1}}$) denotes an AFD, where $1 \leq i_1, i_2 \ldots i_{m+1} \leq n$. For simplicity, let us first consider singleton set as the determining set of the AFD. It would be of the form $\{A_i\} \rightsquigarrow A_j$.

Consider the attribute-value pairs corresponding to this AFD. Let $A_i{:}v_{ik}$ correspond to attribute-value pairs of attribute A_i, and $A_j{:}v_{jl}$ correspond to attribute-value pairs of attribute A_j, where $1 \leq k \leq N_i$ and $1 \leq l \leq N_j$.

Association rules derived from a database relation are a set of attribute-value pairs determining another; ($A_i{:}v_{ik} \rightsquigarrow A_j{:}v_{jl}$) denotes an association rule. For example, (Make:Honda$\rightsquigarrow$Model:Accord) is an association rule.

For simplicity, let us denote the body of the association rule to be α_x and the head of the rule to be β_y (here x varies from 1 to N_i, where as y varies from 1 to N_j. An example of this is shown in the Figure 1 by applying the above principle on the car database fragment in table I. In Figure 1, $\alpha_1 = Honda$, $\alpha_2 = Toyota$, and values of β_1, β_2, β_3 and β_4 are 'Accord', 'Civic', 'Sequoia' and 'Camry' respectively.

The AFD ($\{A_i\} \rightsquigarrow A_j$) thus is a condensation of many such association rules. For example, $Make \rightsquigarrow Model$ is the AFD condensed from all association rules of the type ($Make$: *valueOfMake*$\rightsquigarrow$$Model$: *valueOfModel*), where *valueOfMake* and *valueOfModel* are possible values for Make and Model respectively. (Attribute A_i is *Make* and the attribute A_j is *Model*). Different association rules ($\alpha_1 \rightsquigarrow \beta_1$), ($\alpha_1 \rightsquigarrow \beta_2$) $\ldots$ ($\alpha_1 \rightsquigarrow \beta_{N_j}$, ($\alpha_2 \rightsquigarrow \beta_1$), ($\alpha_2 \rightsquigarrow \beta_2$) $\ldots$($\alpha_2 \rightsquigarrow \beta_{N_j}$), $\ldots$ ($\alpha_{N_i} \rightsquigarrow \beta_1$), ($\alpha_{N_i} \rightsquigarrow \beta_2$) $\ldots$($\alpha_{N_i} \rightsquigarrow \beta_{N_j}$) make up the AFD ($\{A_i\} \rightsquigarrow A_j$).

This notion and similar notation can be extended even for AFDs with more than one attribute in the determining set of the rule. In case of multiple attributes in the determining set, α_x denotes each distinct combination of the corresponding attribute-value pairs on the LHS, and β_y stays the same.

4.1 *Confidence*

If an Association rule is of the form ($\alpha \rightsquigarrow \beta$), it means that if we find all of α in a row, then we have a good chance of finding β. The probability of finding β for us to accept this rule is called the confidence of the rule. Confidence denotes the conditional probability of head given the body of the rule. To define confidence of an AFD, we would expect a similar definition, that it should denote the chance of finding the value for the dependent attribute, given the values of the attributes in the determining set. Thus, we define Confidence of an AFD, *Confidence* in terms of the confidences of the underlying association rules. Specifically, we define it in terms of picking the best association rule for every distinct value-combination of the body of the association rules. For example, if there are two association rules (Honda $\rightsquigarrow$ Accord) and (Honda $\rightsquigarrow$ Civic), given Honda, the probability of occurrence of Accord is greater than the probability of occurrence of Civic. Thus, (Honda $\rightsquigarrow$ Accord) is the best association rule, for (Make = Honda) as the body.

$$\begin{aligned} \mathit{Confidence}\ (X{\rightsquigarrow}A_j) \ &= \ \sum_{x}^{N'} \arg\max_{y\in[1,N_j]} (\mathit{support}(\alpha_x) \times \\ &\quad confidence(\alpha_x \rightsquigarrow \beta_y)) \end{aligned} \tag{1}$$

Here, N' denotes the number of distinct values for the determining set X in the relation r.

The above equation can also be written as,

$$\mathit{Confidence}\ (X{\rightsquigarrow}A_j) \ = \ \sum_{x}^{N'} \arg\max_{y\in[1,N_j]} (\mathit{support}(\alpha_x \rightsquigarrow \beta_y))$$

Example: For the database relation displayed in Table I, Confidence of the AFD $(Make \rightsquigarrow Model)$ = Support $(Make : Honda \rightsquigarrow Model : Accord)$ + Support $(Make : Toyota \rightsquigarrow Model : Camry) = \frac{3}{8} + \frac{2}{8} = \frac{5}{8}$.

It is interesting to see that, this turns out to be equal to (1 - g_3), where g_3 is one of the standard error measures for defining AFDs, and also used by previous algorithms (cite TANE, HLS etc). The g_3 error measure has a natural interpretation as the fraction of tuples with exceptions or errors affecting the dependency. Applying this on the same example table displayed in Table I, the error tuples are the ones numbered 4, 5 and 6. Thus, $g_3 = \frac{3}{8}$, and (1 - g_3) = $\frac{5}{8}$.

Another measure similar to g_3 is g_2, which defines error tuples as all the violating pairs of rows in **r**. Interestingly, if we define AFD's confidence measure in terms of picking the association rules strictly deterministic (confidence of 1), then *Confidence* turns out to be equals to (1-g_2).

$$\begin{aligned} \textit{Confidence } (X \rightsquigarrow A) &= \sum_{x}^{N'} (\textit{support}(\alpha_x) \times \\ & \quad confidence(\alpha_x \rightsquigarrow \beta_y)) \end{aligned} \tag{2}$$

such that, $confidence(\{\alpha_x\} \rightsquigarrow \beta_y) = 1$

$$\textit{Confidence } (X \rightsquigarrow A) = (1 - g_2(T)) \tag{3}$$

Thus, *Confidence* for an AFD is defined in terms of the confidences and supports of the underlying association rules. And we also showed the connections to other standard measures for AFDs.

4.2 *Specificity*

In this section, we show how in addition to confidence, the supports of underlying association rules forming the AFD play an important role in determining the qualify of an AFD and define the *Specificity* metric. In the definition of support for the underlying association rules, we consider only left-hand sides of them (this way, the support and confidence are independent). For an association rule $\alpha \rightsquigarrow \beta$, confidence denotes the conditional probability that two distinct tuples are β-equivalent provided that they are already α-equivalent, where as the support is the probability that the conditioning event occurs [20].

Even if the confidence value is high, low support for an association rule makes it less interesting. Presence of a lot of association rules with low supports makes the AFD bad. For example, in using AFDs as feature selection for classification, if the support is less, it means the probability of the conditioning event is less. This makes it less likely for the value to occur in the test set, thus affecting the prediction accuracy. Also, less support means, the number of tuples returned when we query on this value is less. Thus, throughput (number of tuples returned) per query is less and we need more queries to achieve a high recall. Thus, in addition to confidence, the *distribution* of supports plays an important role in the quality of an AFD.

We illustrate this further by analyzing some cases how supports of the association rules forming the AFD can have different distributions.

1. $Model \rightsquigarrow Make$
   - Few Branches - Uniform Distribution
   - Good, and might hold good universally
2. $VIN \rightsquigarrow Make$
   - Many Branches - Uniform Distribution
   - Bad - Confidence of these rules are high, but the L.H.S attribute sets are approximate keys.
3. $Model, Price \rightsquigarrow Color$
   - Many Branches - Skewed Distribution
   - Good, and probable case for dominating association rules[1]

We need a measure to denote how supports of the bodies of individual association rules are distributed into various branches. For this, we take ideas from the way information gain

[1]It means the AFD is composed of a few association rules with high confidence and many association rules with low confidence

ratio is computed in decision trees. In classification using decision trees, a metric called $SplitInfo$ is used to reflect the distribution of probabilities among the branches while defining the information gain ratio [22]. Both the problems share similar motivations; in decision trees, the idea of using *SplitInfo* is to avoid splitting the tree with an attribute that leads to too many branches. This is because, then, the chance for the values in the training set to repeat in the testing set is less. (For example, if the split attribute is SSN of a person, though the classification is accurate, no SSN in the training set will be seen in the testing set later, because it is unique.) Similarly, in our case, more branches for an AFD formation-tree indicates that the supports are less for may of the association rules. Such AFDs are not of high interest, as motivated in the Chapter 1.

$$\textit{Specificity}\ (X \rightsquigarrow A) = \frac{\sum_{x}^{N'} support(\alpha_x) \times \log_2(support(\alpha_x))}{\log_2(Numtuples_r)}$$

The value is normalized with the maximum possible value for *Specificity* which is nothing but the case when the determining set is a key, i.e., there are there are $Numtuples_r$ distinct values for the attribute combination. The *Specificity* measure captures our intuition about the three types of AFDs explained above.

Monotonicity of *Specificity* : For any attribute sets X and Y such that (Y⊃X), *Specificity* (Y) $\geq$ *Specificity* (X). This property is well exploited in pruning AFDs with bad determining sets. This is explained while describing all pruning strategies during AFD search process in Chapter 5.

5 AFDMINER: DESCRIPTION OF THE MINING ALGORITHM

Having defined the necessary metrics for AFDs in the Chapter 4 above, next we describe the algorithmic details of mining the set of AFDs from a relational table. Formally, the problem of mining the set of AFDs is defined as follows:

Definition 5.0.1 (AFD Mining Problem). *Given a database relation* $\boldsymbol{r}$, *and user-specified thresholds* minConf *(minimum confidence) and* max*Specificity (maximum Specificity), generate all the Approximate Functional Dependencies (AFDs) of the form* $(X \rightsquigarrow A)$ *from* r *for which Confidence* $(X \rightsquigarrow A) \geq minConf$ *and Specificity* $(X) \leq$ max*Specificity*

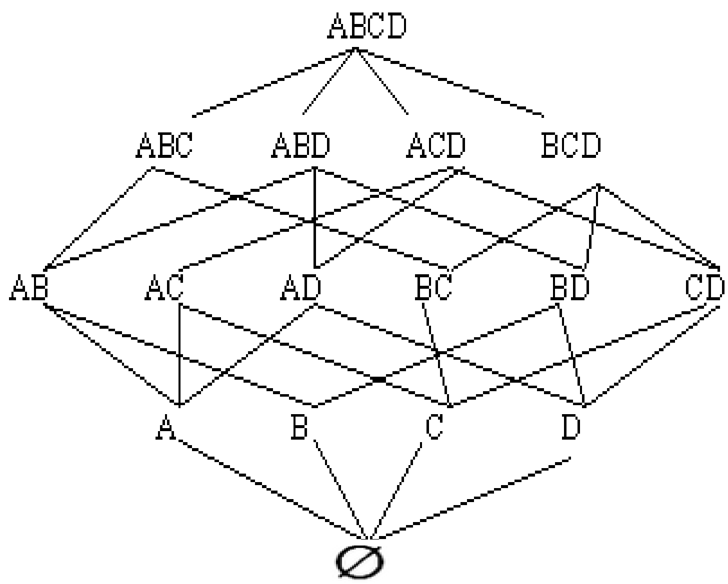

Fig. 2. Set containment lattice of 4 attributes

To find all dependencies according to the definition above, we search through the space of non-trivial dependencies and test the validity of each dependency. Algorithms for mining AFDs face two costs: the combinatoric cost of searching the rule space and the cost of visiting the data to calculate the required metrics for the rules. The first dominant factor is the combinatoric complexity of searching a space related to the power set lattice of the set of attributes. An example for such a lattice for a dataset with four attributes is shown in Figure 2.

We follow a breadth first search strategy and perform a level-wise search in the lattice for all the required AFDs. Bottom-up search in the lattice starts with singleton sets and proceeds upwards level-wise in the lattice, searching bigger sets. Top-down search starts with the biggest possible set and moves down the lattice in a level-wise manner. The choice of doing a bottom-up or top-down search in the lattice depends on what types of pruning strategies are applicable in case of searching the rules we are interested in. In case of AFDs, the level-wise bottom-up algorithm has a powerful mechanism for pruning the search space.

Search starts from singleton sets of attributes and works its way to larger attribute sets through the set containment lattice level by level. When the algorithm is processing a set X, it tests AFDs of the form $X \setminus A \rightsquigarrow A$, where A $\in$ X. In case of FDs, the small-to-large direction of the algorithm can be used to guarantee that only minimal dependencies are generated.

To test minimality of a dependency $X \setminus A \rightarrow A$, we need to know whether $Y \setminus A \rightarrow A$ holds for some proper subset Y of X. We store this information in the set $\mathcal{C}(Y)$ of right-hand side candidates of Y. If A$\in$ $\mathcal{C}(X)$ for a given set X, then A has not been found to depend on any proper subset of X. To find minimal dependencies, it suffices to test dependencies $X \setminus A \rightarrow A$, where A$\in$X and A$\in\mathcal{C}(X \setminus B)$ for all B$\in$X. While the initial rhs candidates are sufficient to guarantee the minimality of discovered dependencies, we will use improved RHS+ candidates $\mathcal{C}^{+}(X)$ that prune the search space more effectively:

$\mathcal{C}^{+}(X) = \{A \in$R$|\forall B \in$X:X$\setminus\{A, B\} \rightarrow\{B\}$ doesnot hold $\}$

The search strategy helps in effective pruning, and also helps in improving the efficiency of the level-wise algorithm. The algorithm calculates the $\mathcal{C}^+(Y)$ values from all its subsets X (X⊂Y). Thus, the information from previous levels is used in reducing the computations at the higher levels and efficiency is achieved. Also, when the algorithm is processing the level of the lattice that contains a set X, we can with one stroke cut off all supersets of X from the lattice simply by deleting X. If some property of X tells us that no superset of is interesting to us, we just delete X. We illustrate some such properties below for effectively pruning the search space:

5.1 Pruning the search space

5.1.1 Pruning based on *Specificity*

Definition 5.1.1 (failAttrbSet). *An attribute set X is called a* failAttrbSet *if Specificity (X)* $>$ max$Specificity$ *where* max*Specificity is the user-specified threshold on the value of Specificity*

An AFD with a *failAttrbSet* as its determining set is not of interest to us. Also, for any attribute set Y, such that $Y \supset X$, *Specificity* (Y) $\geq$ *Specificity* (X). And, if *Specificity* (X) $>$ *maxSpecificity* , then *Specificity* (Y) $>$ *maxSpecificity* which means Y becomes a *failAttrbSet*. Thus, at any point in the search, if we come across a *failAttrbSet* X, we can eliminate all AFDs having X or any superset of X as their determining set.

5.1.2 Pruning applicable to FDs

In case of FDs, generating minimal cover is enough because using Armstrong axioms we can generate the entire set of FDs with out visiting the data explicitly to check if they hold good or not. Thus, it is enough to generate only minimal pass rules in case of FDs [7].

In case of FDs, empty RHS+ candidate set has a similar property as that of *Specificity* metric. That is, if $\mathcal{C}^+(X)=\emptyset$, then $\mathcal{C}^+(Y)=\emptyset$ for all supersets Y of X, and no dependency of the form $Y \rightsquigarrow$A can be minimal FD. Thus the set Y need not be processed at all. Thus we can prune all the supersets if we encounter an empty RHS+ candidate set for an AFD.

5.1.3 Pruning keys and superkeys

An attribute set X is a superkey if no two tuples agree on X. X is a key if it is superkey and no proper subset of it is a superkey.

Since no two tuples agree on X, when we consider an AFD (X$\rightsquigarrow$ A), the confidence is '1'. This is because for each distinct value of X, there is only one value of A. Thus, there are N association rules each with a confidence of '1', and support 1/N. By the definition of confidence for AFD, adding them up will give us '1' as the confidence of the AFD. If X is a superkey then (X$\rightarrow$ A) is always valid and we do not need to check for any of the supersets of X exclusively. Hence, we can prune all keys[1] and their supersets.

5.2 Traversal in the search space

We are interested in looking for AFDs with a confidence greater than the given threshold, such that the *Specificity* value of the determining set of the AFD is within the constrained value. Consider that we are analyzing an AFD (X$\rightsquigarrow$A) for computing its metrics. The confidence value can only increase or stays the same if the determining set is expanded. If

[1]This pruning is subsumed by *Specificity* because keys have the highest *Specificity* value possible. But we use this in addition to the *Specificity* pruning to handle the case when the user chooses not to input any *maxSpecificity* value for the algorithm

we find that confidence of an AFD is less than the given threshold, we have to move up in the lattice because we are interested in not just minimal pass rules, but all AFDs with good confidence values. Thus, unless the confidence is '1' (which means the AFD becomes an FD), we cannot stop expanding the determining set of the AFD for the same dependent attribute. For example, if the confidence threshold is 0.7 (say) and Confidence(X$\rightsquigarrow$ A) is 0.75, we still need to look for rules of the form (Y$\rightsquigarrow$A) where Y$\supset$X, because the confidence value can only increase in an expanded determining set.

The only stopping criteria at a node for the level-wise search are:

1. The AFD confidence becomes 1, and thus it is an FD. In this case, it is trivial that all AFDs $Y \rightsquigarrow A$ where $Y \supset X$ are FDs, and can be generated without visiting the data again.

2. The *Specificity* value of the X is greater than the max value given.

5.3 Computing Confidence

As explained in the derivation of Confidence in Chapter 4, Confidence turns out to be equal to (1-g_3). TANE presented a partition based approach to efficiently compute the minimum number of error tuples to be removed for the approximate dependency to become an FD. We reuse the same error function to compute the confidence as (1 - e(X$\rightsquigarrow$A)). We briefly explain the idea below (The detailed algorithm 5 is presented below).

Any attribute set X partitions the tuples of the relation into equivalence classes. We denote the equivalence class of a tuple $t \in r$ with respect to a given set $X \subseteq$R by $[t]_X$, i.e. $[t]_X = \{u \in r \mid t[A] = u[A]$ for all A$\in X\}$. The set π_X= $\{[t]_X | t \in r\}$ of equivalence classes is a partition of r under X. That is, π_X is a collection of disjoint sets (equivalence classes) of tuples, such that each set has a unique value for the attribute set X and the union of the sets equals the relation r.

Example: For the carDB example in Table I,

$$\pi_{Make} = \{\{1,2,3,4,5\},\{6,7,8\}\}$$

$$\pi_{Model} = \{\{1,2,3\},\{4,5\},\{6\},\{7,8\}\}$$

A functional dependency (X→A) holds if and only if $|\pi_X|$=$|\pi_{X\cup A}|$. For AFDs,

$$\begin{aligned} G_3(X \rightsquigarrow A) &= 1 - \sum_{c\in\pi_X} max\{|c^{'}| \mid c^{'} \in \pi_{X\cup A} and c^{'} \subseteq c\} \\ g_3(X \rightsquigarrow A)) &= G_3(X \rightsquigarrow A)/|r| \end{aligned}$$

5.4 Algorithms

Algorithm 1 presents the main $AFDMiner$ algorithm.

Algorithm 1 AFDMiner: Levelwise search of dependencies

```
L_0 := {∅}
C⁺(∅) := R
L_1 := {{A} | A ∈ R}
ℓ := 1
while L_ℓ ≠ ∅ do
    ComputeDependenciesAtALevel(L_ℓ)
    PRUNE(L_ℓ)
    L_{ℓ+1} := GenerateNextLevel(L_ℓ)
    ℓ := ℓ + 1
```

GenerateNextLevel computes the level $L_{\ell+1}$ from L_ℓ. The level $L_{\ell+1}$ will contain only those attribute sets of size $\ell+1$ which have their subsets of size ℓ in L_ℓ, i.e.

$L_{\ell+1} = \{X \mid |X| = \ell + 1$ and for all Y with Y ⊂ X and $|Y| = \ell$ we have Y $\in L_\ell$ }.

Algorithm 2 (ComputeDependenciesAtALevel(L_ℓ)) computes all the AFDs and FDs that hold true at the given level of the lattice. It also updates the RHS+ candidate sets. Line

Algorithm 2 ComputeDependenciesAtALevel(L_ℓ)

1: **for all** $X \in L_\ell$ **do**
2: $\quad \mathcal{C}^+(X) := \bigcap_{A \in X} \mathcal{C}^+(X \backslash \{A\})$
3: **for all** $X \in L_\ell$ **do**
4: $\quad$ **for all** $A \in X \cap \mathcal{C}^+(X)$ **do**
5: $\quad\quad$ **if** $Confidence(X \backslash \{A\} \rightarrow A) \geq minConf$ **then**
6: $\quad\quad\quad$ **if** $(X \backslash \{A\} \rightarrow A)$ holds exactly **then**
7: $\quad\quad\quad\quad$ output $X \rightarrow A$ as an FD
8: $\quad\quad\quad\quad$ remove A from $\mathcal{C}^+(X)$
9: $\quad\quad\quad\quad$ remove all B in $R \backslash X$ from $\mathcal{C}^+(X)$
10: $\quad\quad\quad$ **else**
11: $\quad\quad\quad\quad$ output $X \rightsquigarrow A$ as an AFD with $\mathit{Confidence}(X \rightsquigarrow A) = (1 - g_3(X \rightsquigarrow A))$

6 checks if the dependency is an FD and removes the candidates A and all B (B∈R\ X) from $\mathcal{C}^+(X)$.

Algorithm 3 PRUNE(L_ℓ): Pruning dependencies

1: **for all** $X \in L_\ell$ **do**
2: $\quad$ **if** $\mathcal{C}^+(X) = \emptyset$ **then**
3: $\quad\quad$ delete X from L_ℓ
4: $\quad$ **if** $CalculateSpecificity\ (X) \geq \mathit{maxSpecificity}$ **then**
5: $\quad\quad$ delete X from L_ℓ

Algorithm 3 implements the different pruning strategies explained above. Lines 2 and 3 correspond to the pruning of empty RHS+ candidate sets. Pruning based on *maxSpecificity* threshold is reflected by the lines 4 and 5.

Algorithm 4 Calculating *Specificity* for a given attribute set X

sumInfoX := 0
for $i = 1$ to $numPartitionsX$ **do**
$\quad$ sumInfoX := sumInfoX + $(|\pi_{X_i}| \times log_2(|\pi_{X_i}|)$
InfoSuppX := sumInfoX / (log_2N)

In the Algorithm 4, π_{X_i} corresponds to the i^{th} the i^{th} equivalence class for the partition of tuples corresponding to the attribute set X.

Algorithm 5 Calculating *Confidence* for an AFD (X$\rightsquigarrow$A)

Require: stripped partitions $\widehat{\pi_X}$ and $\widehat{\pi_{X \bigcup \{A\}}}$

1: $e := 0$
2: **for all** $c \in \widehat{\pi_{X \cup \{A\}}}$ **do**
3: choose (arbitrary) $t \in c$
4: $T[t] := |c|$
5: **for all** $c \in \widehat{\pi_X}$ **do**
6: $m := 1$
7: **for all** $t \in c$ do $m := max\{m, T[t]\}$ **do**
8: $e := e + |c| - m$
9: **for all** $c \in \widehat{\pi_{X \cup \{A\}}}$ **do**
10: do $t \in c$ (same t as on line 3)
11: $T[t] := 0$
12: return (1-$e/|r|$)

Algorithm 6 PRUNE(L_ℓ): Pruning dependencies in TANE

1: **for all** $X \in L_\ell$ **do**
2: **if** $\mathcal{C}^+(X) = \emptyset$ **then**
3: delete X from L_ℓ
4: **if** X is a (super)key **then**
5: **for all** $A \in C^+(X) \setminus X$ **do**
6: **if** $A \in \bigcap_{B \in X} C^+(X \bigcup \{A\} \setminus \{B\}$ **then**
7: output $X \rightarrow A$
8: delete X from L_ℓ

Algorithm 6 is the pruning used by TANE. It uses pruning based on empty RHS+ candidate set and pruning based on Superkey (which AFDMiner doesnot need because it is subsumesd by the pruning based on *Specificity*). Next, we describe the evaluation of our algorithm and the metrics.

6 EMPIRICAL EVALUATION

In this chapter, we describe the implementation and empirical evaluation of AFDMiner for efficiency as well as quality of the rules generated. By taking a sample application for using AFDs – feature selection for classification, we show how the quality of AFDs generated by AFDMiner are better. By varying *Specificity* and analyzing the behavior of quality and time taken, we demonstrate what could be a reasonable value for *maxSpecificity* threshold. We present a discussion on comparing AFDMiner with TANE as well as an extended approach to TANE called TANENoMinP (by removing the constraint on stopping at only minimal pass rules). We evaluate the effectiveness of using *Specificity* metric along with confidence through performance experiments by varying several parameters of the algorithm.

6.1 Experimental Settings

The algorithms were implemented in C with necessary optimizations. The programs were compiled with GNU C compiler. All experiments were run on the same 2.67 GHz Pentium 4 PC with 1 GB of memory running Linux operating system. The following are the input parameters for the AFD mining algorithm:

1. number of tuples in the data set
2. number of attributes in the data set
3. *minConf*
4. *maxSpecificity*
5. length of the determining set of each AFD

We performed evaluations over two data sets. The datasets are CensusDB and MushroomDB, and were taken from UCI Machine Learning repository. The description of each dataset is given below:

1. CensusDB (US Census data 1990) has a total of 199523 tuples, and 30 attributes
2. CarDB has 5526 tuples and 9 attributes
3. MushroomDB has 8124 tuples, and 23 attributes

NoSpecificity : To show the effectiveness of *Specificity* we compare $AFDMiner$ with an approach NoSpecificity which is a modified version of AFDMiner. NoSpecificity varies from AFDMiner, in using only Confidence but not *Specificity* for AFDs. Thus, it generates all AFDs (X$\rightsquigarrow$A) with (Confidence(X$\rightsquigarrow$A)$>minConf$).

6.2 Evaluation of Quality

To evaluate and compare the quality of the AFDs generated, we choose a sample task of using AFDs. AFDs were proven to be very effective in a system called QPIAD [24], a query processing framework over incomplete autonomous databases. AFDs are used to rewrite queries using the values of the attributes in the determining set to retrieve tuples with missing values for the dependent attribute. Also, predicting the missing value for the attribute is treated as a classification problem, where AFDs are used as feature selectors (this technique is termed as AFD-enhanced-classification).

For example, if a query is Q(Bodystyle='convt'), and value for the attribute *Bodystyle* is missing in some tuples, an AFD (Model$\rightsquigarrow$ Bodystyle) is used in retrieving the tuples – using values of *Model*. Then, using Model as a feature, coupled with a classifier, values for the attribute *Bodystyle* are predicted.

BestAFD: Since there are many AFDs with the same dependent attribute, we pick a single best AFD for the attribute (say, A) whose values are being predicted. Thus, termed BestAFD is the AFD having the highest confidence among all the AFDs having the attribute A as the dependent attribute.

Once BestAFD is chosen, a classifier is run using the attributes in the determining set of the BestAFD as the set of features for classification. We used 10-fold cross-validation and computed the classification accuracy of the classifier. For various classifiers, we used the implementation provided in Weka [21], a machine learning tool-kit for our experiments. We experimented over the CensusDB described above in the previous section. The number of tuples and number of attributes, are taken to be 10000 and 25 respectively. BestAFD for each attribute in the table as the dependent attribute was picked by running through all the AFDs generated by each approach AFDMiner and NoSpecificity separately .

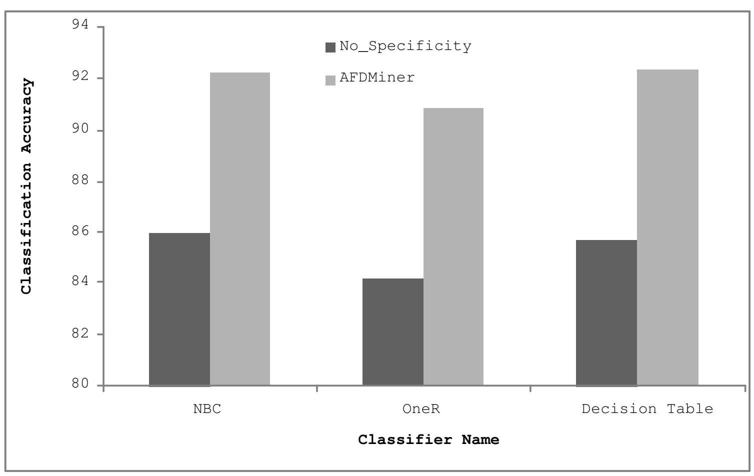

Fig. 3. CensusDB: Comparing quality of AFDs between AFDMiner and NoSpecificity

We considered average classification accuracy as the quality measure, and defined it as the average of all accuracies of classification, using each AFD in the BestAFDs set for feature selection. Figure 7 shows the comparison between the average classification accuracies of using the BestAFDs by AFDMiner and NoSpecificity for feature selection followed by classification. We performed the experiment over various classifiers, Naive Bayes Classifier (NBC), *OneR* and *DecisionTable* provided in Weka. These experiments show that the quality of AFDs generated by AFDMiner is better than those by NoSpecificity approach, which shows that *Specificity* is effective in generating better quality AFDs. The *minConf* threshold in these experiments was set to 0.8, where as for the *maxSpecificity* value, we set a threshold of 0.4, based on the following experiment.

The average classification accuracy by AFDMiner is 93% which signify that they are of very good quality.

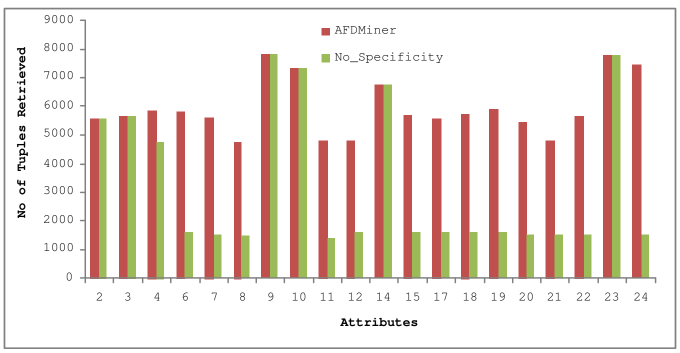

Fig. 4. CensusDB: Comparing the query throughput of top-10 queries using BestAFD (AFDMiner Vs NoSpecificity

6.2.1 Evaluating Query Throughput

We evaluated the query throughput by sending the top-10 queries to the censusDB table. These queries were based on values in the determining set of the BestAFD used. (the same AFD used in the previous experiments). This experiment demonstrates that the number of tuples returned by the queries based on BestAFD of AFDMiner outperforms those by the queries based on BestAFD of NoSpecificity. Thus, our approach gives not only better classification accuracy, but also better query throughput.

6.2.2 Choosing a good *maxSpecificity* threshold

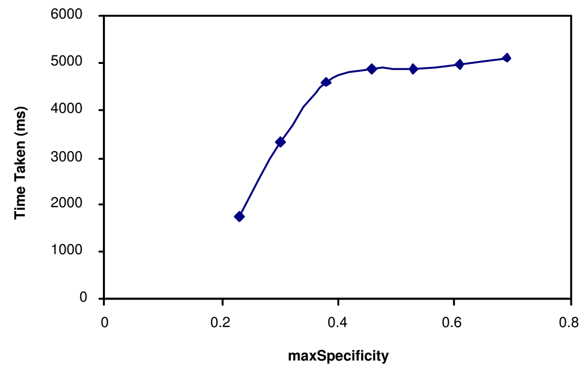

Fig. 5. CensusDB: Time taken with varying *maxSpecificity* thresholds

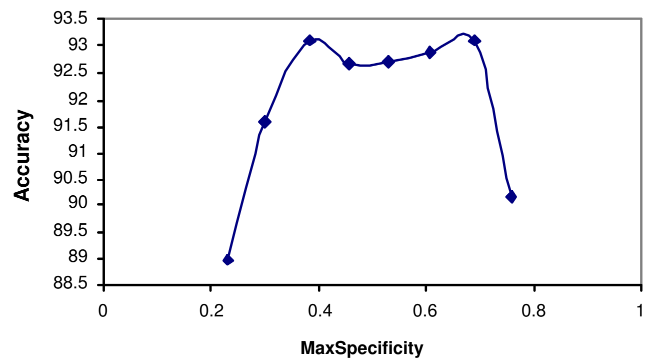

Fig. 6. CensusDB: Quality of AFDs with varying *maxSpecificity* thresholds

We plotted the graphs showed in Figures 5 and 6 by varying the *maxSpecificity* thresholds for various runs of AFDMiner. This is performed over the CensusDB. The curve in Figure 5 shows the time taken by AFDMiner where as the curve in Figure 6 represents the classification accuracy.

This showed that when the threshold is very low, though the algorithm runs faster, it skips many good rules having good confidence values. Thus, the BestAFDs set picked here may not give us a good classification accuracy. Also, when the *maxSpecificity* threshold is very high, rules which we consider bad are not being pruned. Thus the accuracy goes down as well as the time taken to generate them is very high. If we analyze the behavior of the curve in Figure 6 closely, the accuracy keeps increasing till a certain point, and then there is a region in the middle with reasonable *maxSpecificity* values where the classification accuracy does not vary much. This is because, there is not much change in the set of BestAFDs chosen. For example, if the number of attributes is 25, which means the number of BestAFDs is 25, only 2 or 3 of them keep changing for every *maxSpecificity* in this region. Then again, the accuracy starts to drop for higher values of *maxSpecificity* threshold. Thus, the behavior of classification accuracy by varying the *maxSpecificity* approximately forms a double elbow shaped curve.

Increasing the *maxSpecificity* value will increase the time taken for computing AFDs because the number of rules pruned would be less for higher *maxSpecificity* thresholds. Interestingly, though the time taken keeps increasing, the rate of increase beyond a certain interval is less. This approximately corresponds to the first elbow in the plot above for accuracy.

A good value for *maxSpecificity* is the value at the first elbow in the graph on quality, because it is the point for which the quality is very good as well as the time taken is less. Thus, good AFDs are the ones with a reasonable *maxSpecificity* threshold and high confidence.

6.3 Comparison with TANE

In this section we compare the AFDs generated by AFDMiner with those by TANE. TANE is mainly designed to generate the minimal cover for functional dependencies. For

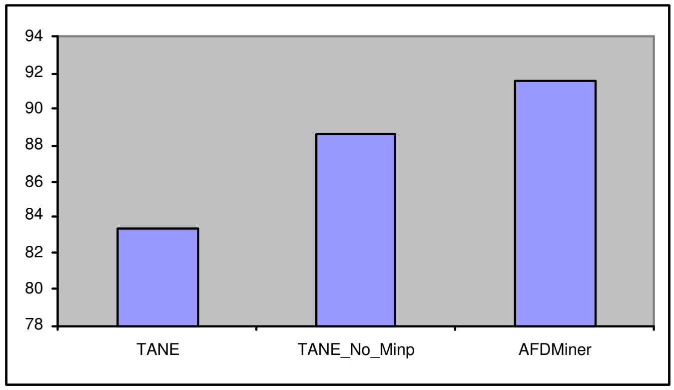

Fig. 7. CensusDB: Comparing quality of AFDs between TANE, TANENoMinP and AFDMiner

approximate dependencies, TANE uses g_3 as the error metric. It performs a breadth first search in the set containment lattice of attributes and generates minimal pass rules.

TANENoMinP : TANENoMinP is a modified version of TANE that generates all AFDs within the error threshold of g_3, and does not just stop with just minimal pass rules.

As the $minConf$ was 0.8, we set the g_3 to be 0.2. For the similar experimental setup for quality as explained above, we compare the set of AFDs generated by TANE with those of TANENoMinP and AFDMiner. Figure 7 shows the comparison. Firstly it shows that AFDs with high confidence and longer determining sets are better than just minimal pass rules. And also it shows that AFDMiner outperforms both the metrics – thus strengthening the argument that AFDs with high confidence and with reasonable *Specificity* are the best. Next, we show the evaluation of the performance of AFDs generated (by using the same *maxSpecificity* value we obtained from the quality curves).

6.4 Evaluation of Performance

We ran experiments varying each parameter of AFDMiner and computed the time taken to generate the AFDs. For all the performance experiments over the CensusDB dataset, the *minConf* was set to be 0.8. The number of tuples and number of attributes, if unspecified, are taken to be 10000 and 25 respectively. The length of the determining set of the AFD is 4 for all experiments except the ones shown in Figures 10 and 13.

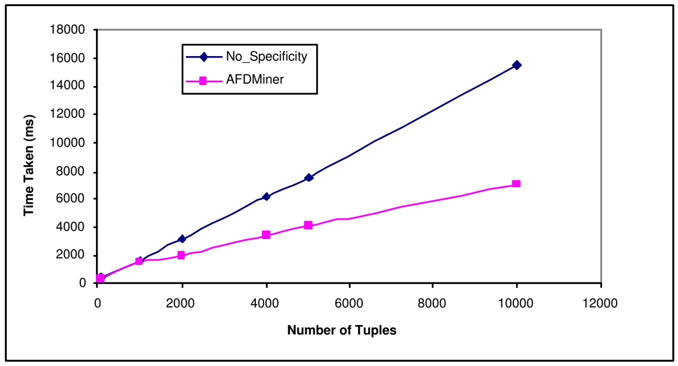

Fig. 8. CensusDB: Varying size of the dataset

6.4.1 Varying number of tuples and attributes

Experiment in Figure 8 shows that time taken by both AFDMiner and NoSpecificity varies linearly with the number of tuples. But AFDMiner takes less time compared to that of NoSpecificity. Similarly, the experiment in Figure 9 shows that – time taken by both the approaches varies exponentially on the number of attributes, but AFDMiner completes much faster than NoSpecificity.

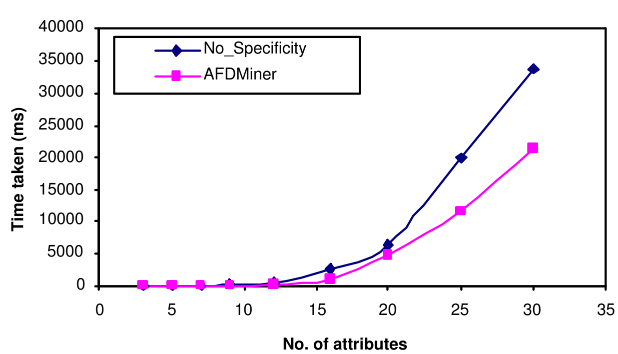

Fig. 9. CensusDB: Varying number of attributes in the data set

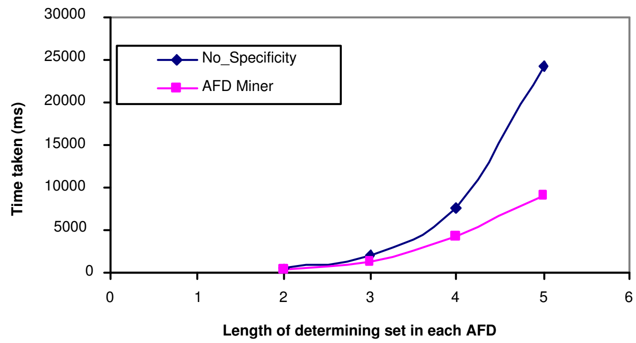

Fig. 10. CensusDB: Varying the length of the determining set

6.4.2 Varying length of the determining set

Since both AFDMiner and NoSpecificity do a level-wise search in a breadth first manner, we evaluated the time taken, by putting a constraint on the level till which they should search for AFDs. This means, a constraint on the maximum length of the determining set of the AFDs is enforced. Through the experiment in Figure 10 we showed that similar behavior as observed in varying the number of attributes is noticed. The time taken during the search varies exponentially on the length of the determining set, but AFDMiner outperforms NoSpecificity by a large margin.

6.4.3 Other performance-related experiments

We ran similar experiments on a different dataset, i.e., MushroomDB mentioned above. The experiments shown in the Figures 11 and 12 indicate that the performance of AFD-Miner compared to NoSpecificity shows similar behavior as it did in case of CensusDB.

Also, the experiment showed in Figure 13 shows that the number of candidates (AFDs) visited during the search for the type of AFDs desired, by AFDMiner is less than those

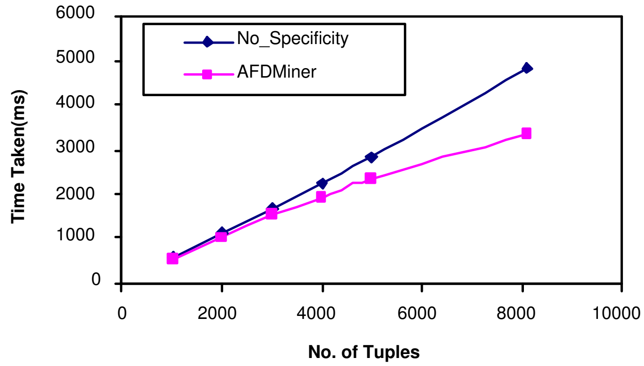

Fig. 11. MushroomDB: Varying the size of the dataset

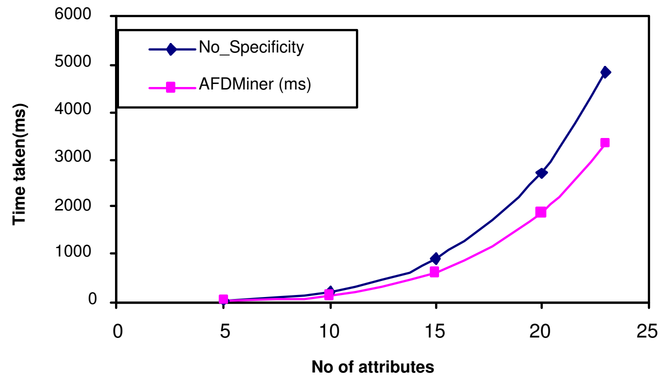

Fig. 12. MushroomDB: Varying the number of attributes

traversed by NoSpecificity . This demonstrates that a good amount of search space containing bad rules is effectively pruned by AFDMiner.

Intuitively, if we decrease the *maxSpecificity* further, the time taken would be less, but at the same time the quality goes down. This is the reason why we fixed it to the reasonable value as explained in the previous section 6.2. Above experiments show that for a good quality of the generated AFDs, the time taken is less and *Specificity* is effective.

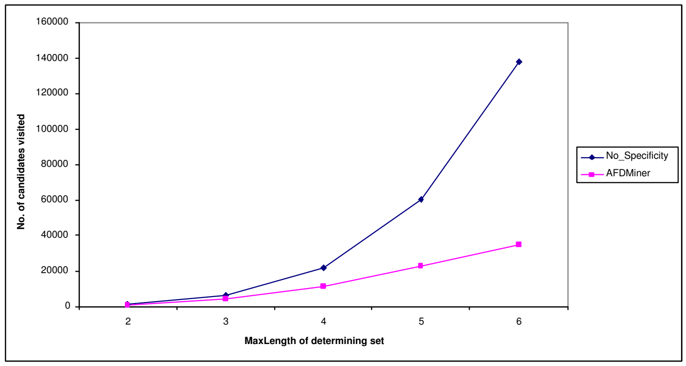

Fig. 13. CensusDB: No. of candidates visited at each level of search

model mileage ~~~> year
model price ~~~> make
make body ~~~> model
year model ~~~> price
year model ~~~> mileage
model price ~~~> body
year model ~~~> certified

Fig. 14. best AFDs from carDB

6.5 Sample AFDs from different domains:

The set of AFDs mined from the carDB and mushDB are displayed in Figures 14 and 15 respectively.

bruises? gill-attachment stalk-surface-above-ring ~~~> cap-shape
cap-shape stalk-surface-above-ring population ~~~> gill-attachment
stalk-color-below-ring ~~~> gill-spacing
bruises? gill-color stalk-surface-above-ring ~~~> gill-size
cap-shape gill-size eil-type ~~~> stalk-surface-below-ring
stalk-surface-above-ring eil-type spore-print-color ~~~> stalk-color-above-ring
stalk-color-below-ring ~~~> ring-number
cap-shape gill-color population ~~~> ring-type

Fig. 15. best AFDs from mushroomDB

7 CONCLUSION

We introduced a novel perspective for Approximate Functional Dependencies (AFDs), treating them as condensed roll-ups of association rules. Based on this view, we devised two metrics for AFDs namely *Confidence* and *Specificity* derived from the metrics $Confidence$ and *Support* of the underlying association rules that form the AFD. We also showed how *Confidence* is connected to two other standard approximation measures for AFDs. We presented an algorithm $AFDMiner$ that generates all AFDs within the user-specified thresholds on the values of the metrics. $AFDMiner$ performs a bottom-up search in a breadth first manner in the set containment lattice of attributes. In addition to pruning techniques applicable for FDs, we showed how the monotonicity of *Specificity* can be exploited to effectively prune the search space of AFDs. Through empirical studies on real datasets, we showed that the AFDs are generated faster by $AFDMiner$ and also that they are of high quality (by using them for a sample classification task). We analyzed the behavior in quality and time by varying the *maxSpecificity*and concluded that the most useful AFDs (and can be generated faster) are the ones with high *Confidence* and a reasonable *Specificity* value.

A possible future extension for the proposed approach of moving from association rules to AFDs is an efficient mining algorithm for Conditional Functional Dependencies (CFD) and Conditional Approximate Functional Dependencies (CAFD). CFDs are dependencies of the form (*Make→Model if country =“England”*). More formally, CFDs are deterministic dependencies holding true only for certain values of one or more of other attributes [4]. CAFD's are the probabilistic counter part of CFDs—CFDs holding with a probability—similar to AFDs are probabilistic counter part of FDs. CFDs and CAFDs are applied in data cleaning and value prediction recently [5], but mining these conditional rules is largely unexplored. Intuitively, CFDs are intermediate rules between association rules (value level) and FD (attribute level). Hence the proposed approach of condensing association rules is likely to enable efficient discovery of interesting CFDs and CAFDs.

[11] suggests determinations as a representation of knowledge that should be easy to understand. It presents ConDet, an algorithm that uses feature selection to construct determinations from training data, augmented by a condensation process that collapses rows to produce simpler structures. This is related to how decision table classifier works. Current work on mining AFDs for all attributes can be considered as—finding the best feature set for each and every attribute in the dataset. (while decision table has a specified attribute as the class attribute). But, AFDMiner doesnot save the probability distributions of the values in the training set and relies on just suggesting the feature set for the AFD-enhanced classifier. An interesting future direction is to combine the two approaches—in saving the association rules generated during the process to learn the value distribution and build a complete AFD-enhanced classifier.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] Aijun An, Shakil Khan, and Xiangji Huang. Objective and subjective algorithms for grouping association rules. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 477, Washington, DC, USA, 2003. IEEE Computer Society.

[3] F. Berzal, J. C. Cubero, F. Cuenca, and J. M. Medina. Relational decomposition through partial functional dependencies. *Data Knowl. Eng.*, 43(2):207–234, 2002.

[4] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755. IEEE, 2007.

[5] Wenfei Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170, 2008.

[6] Chris Giannella and Edward Robertson. On approximation measures for functional dependencies. *Inf. Syst.*, 29(6):483–507, 2004.

[7] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[8] Yka Huhtala, Juha Kinen, Pasi Porkka, and Hannu Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE*, pages 392–401, 1998.

[9] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658, New York, NY, USA, 2004. ACM.

[10] Jyrki Kivinen and Heikki Mannila. Approximate dependency inference from relations. In *ICDT*, pages 86–98, 1992.

[11] Pat Langley. Induction of condensed determinations. In *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96*, pages 327–330. AAAI Press, 1996.

[12] Brian Lent, Arun N. Swami, and Jennifer Widom. Clustering association rules. In *ICDE '97: Proceedings of the Thirteenth International Conference on Data Engineering*, pages 220–231, Washington, DC, USA, 1997. IEEE Computer Society.

[13] Bing Liu. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining*, pages 80–87, 1998.

[14] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *EDBT '00: Proceedings of the 7th International Conference on Extending Database Technology*, pages 350–364, London, UK, 2000. Springer-Verlag.

[15] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Functional and approximate dependency mining: database and fca points of view. *J. Exp. Theor. Artif. Intell.*, 14(2-3):93–114, 2002.

[16] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *AAAI Workshop on Knowledge Discovery in Databases (KDD-94)*, pages 181–192, Seattle, Washington, 1994. AAAI Press.

[17] Ullas Nambiar and Subbarao Kambhampati. Answering imprecise queries over autonomous web databases. In *ICDE*, page 45, 2006.

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

[19] Jeffrey C. Schlimmer. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *International Conference on Machine Learning*, pages 284–290, 1993.

[20] Jef Wijsen, Raymond T. Ng, and Toon Calders. Discovering roll-up dependencies. In *Knowledge Discovery and Data Mining*, pages 213–222, 1999.

[21] I. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. Cunningham. Weka: Practical machine learning tools and techniques with java implementations, 1999.

[22] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2 edition, 2005.

[23] Garrett Wolf, Hemal Khatri, Yi Chen, and Subbarao Kambhampati. Quic: A system for handling imprecision & incompleteness in autonomous databases (demo). In *CIDR*, pages 263–268, 2007.

[24] Garrett Wolf, Hemal Khatri, Bhaumik Chokshi, Jianchun Fan, Yi Chen, and Subbarao Kambhampati. Query processing over incomplete autonomous databases. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 651–662. VLDB Endowment, 2007.

[25] Catharine Wyss, Chris Giannella, and Edward Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances.

[26] Hong Yao and Howard J. Hamilton. Mining functional dependencies from data. *Data Min. Knowl. Discov.*, 16(2):197–219, 2008.