Semi-automatic support for evolving functional dependencies

Mirjana Mazuran
DEIB – Politecnico di Milano
mirjana.mazuran@polimi.it

Elisa Quintarelli
DEIB – Politecnico di Milano
elisa.quintarelli@polimi.it

Letizia Tanca
DEIB – Politecnico di Milano
letizia.tanca@polimi.it

Stefania Ugolini
Department of Mathematics – Milan University
stefania.ugolini@unimi.it

ABSTRACT

During the life of a database, systematic and frequent violations of a given constraint may suggest that the represented reality is changing and thus the constraint should evolve with it. In this paper we propose a method and a tool to (i) find the functional dependencies that are violated by the current data, and (ii) support their evolution when it is necessary to update them. The method relies on the use of *confidence*, as a measure that is associated with each dependency and allows us to understand "how far" the dependency is from correctly describing the current data; and of *goodness*, as a measure of *balance* between the data satisfying the antecedent of the dependency and those satisfying its consequent. Our method compares favorably with literature that approaches the same problem in a different way, and performs effectively and efficiently as shown by our tests on both real and synthetic databases.

1. INTRODUCTION

The information related to a certain reality of interest is represented in a database by means of data, a vital resource on which decision-making processes are based. The data stored in a database must satisfy the semantic conditions expressed by the schema plus integrity constraints. Specifying and enforcing constraints grants us better data quality, maintenance, query optimization, view updating and database integration and exchange; in particular, *functional dependencies* have been widely applied to these aims and, from the 70s on, their knowledge has been used to support database design and management, reverse engineering and query optimization. The number of application scenarios they are used in has grown over time including, more recently, automated DB analysis such as knowledge discovery and data mining. A functional dependency (FD) is a constraint between two sets X and Y of attributes in a relation R of a database: R is said to satisfy the functional dependency X → Y if each X value is associated with precisely one Y value.

Nowadays, huge amounts of data are generated daily and may come from different applications and sources where constraints are not equally enforced, thus the original datasets may be inconsistent with each other or even by themselves [1, 2]. Once a database designer or administrator is able to understand that a constraint no longer holds, he/she can decide what to do. Most systems that deal with discrepancies of this kind re-establish consistency by changing the data that violate the constraints. In our work we allow for a different interpretation. Indeed, from our point of view, changes in the data might also mean that their semantics is evolving for some reason, like for instance law or policy changes. Therefore, once the DB administrator has ascertained that this is the case, s/he will be able to re-establish consistency by appropriately modifying the violated constraints. Note that the new constraints capture a succinct representation of the new semantics of the data, thus, this kind of analysis is interesting for knowledge discovery purposes.

This paper's main goal is to propose a method to modify functional dependencies so as to adjust them to the evolutions of the modeled reality that may occur during database life. The method we propose provides a way to understand which FDs are violated and, if needed, to modify them by adding, to their antecedent, a minimal set of attributes that makes them consistent with the data.

Running example *Consider the relation Places in Figure* 1 *and assume that the following FDs be defined on it:*

$F_1 : [District, Region] \rightarrow [AreaCode]$

$F_2 : [Zip] \rightarrow [City, State]$

$F_3 : [PhNo, Zip] \rightarrow [Street]$

All the tuples in Places violate F_1*; tuples* t_1, t_2 *and* t_3 *violate* F_2 *and tuples* t_{10} *and* t_{11} *violate* F_3*. Thus, these three FDs are not satisfied by the data. If the DBMS is able to detect that (e.g. by means of periodic or continuous checks of FDs validity) then it can present it to the designer. Suppose the designer realizes that an FD not being satisfied by the data is not a mistake but a symptom of a real-world situation which is no more reflected by the semantics of the FDs, at this point s/he can decide to integrate some of the candidate*

tid	District	Region	Municipal	AreaCode	PhNo	Street	Zip	City	State
t_1	Brookside	Granville	Glendale	613	974-2345	Boxwood	10211	NY	NY
t_2	Brookside	Granville	Glendale	613	974-2345	Boxwood	10211	NY	NY
t_3	Brookside	Granville	Glendale	613	299-1010	Westlane	10211	NY	MA
t_4	Brookside	Granville	Guildwood	515	220-1200	Squire	02215	Boston	MA
t_5	Brookside	Granville	Guildwood	515	220-1200	Squire	02215	Boston	MA
t_6	Alexandria	Moore Park	NapaHill	415	220-1200	Napa	60415	Chicago	IL
t_7	Alexandria	Moore Park	NapaHill	415	930-2525	Main	60415	Chicago	IL
t_8	Alexandria	Moore Park	NapaHill	415	555-1234	Tower	60415	Chester	IL
t_9	Alexandria	Moore Park	QueenAnne	517	888-5152	Main	60415	Chicago	IL
t_{10}	Alexandria	Moore Park	QueenAnne	517	888-5152	Main	60601	Chicago	IL
t_{11}	Alexandria	Moore Park	QueenAnne	517	888-5152	Bay	60601	Chicago	IL

Figure 1: Running example: relation *Places*

changes into the database. Thus, the idea of the paper is to find a way to change the constraints instead of the data and make them valid again. How can we repair an FD? First of all, without loss of generality we can assume that all FDs are decomposed so that their consequent contains a single attribute. In this way it is easy to see that modifying the consequent is a no-issue. What we can do instead is acting upon the antecedent: of course deleting attributes from it cannot repair the FD but adding attributes might. Thus, our aim is to identify the FDs violated by the data, find possible repairs and present them to the designer to be evaluated. The method can be used periodically on the data in order to keep them consistent with the constrains.

The paper is organized as follows: in Section 2 we review the state of the art, Section 3 introduces the technical notions and the considerations at the basis of our proposal, while Section 4 presents our proposal to evolve violated FDs. In Section 5 we formally compare our approach with a proposal which has the same aim as ours, but is based on the notion of entropy to measure how much an FD is far from being exact, while for the same aim we use the (simpler) notion of *confidence*. In Section 6 we explain the experimental results we have obtained on both real and synthetic datasets and finally in Section 7 we draw the conclusions of our work.

2. RELATED WORK

In the last years, research involving FDs has taken a different turn with respect to the past; today, automated data analysis has become fundamental for knowledge discovery and data mining and FDs in particular provide invaluable intensional knowledge on the relation instances. As a consequence, the concept has been used in a wide range of application scenarios, which in turn originated many extensions and variants [3], including: Conditional FDs (CFDs) [4], Approximate FDs (AFDs) [5], Approximate Conditional FDs (ACFDs) [6], Temporal FDs (TFDs) [7], Approximate Temporal FDs (ATFDs) [8], and others.

CFDs [4], have been introduced that specify FDs that do not hold on the whole relation but just on a subset of its data.

Both FDs and CFDs are exact, i.e., they hold for all the instances in the relation (in the case of FDs) or for certain subsets of it (for CFDs). These rules may easily break in the sense that even small errors or minor changes to the relation instance may cause that the constraint no longer holds. However, as time passes by and the lifecycle of a database goes on, reality may change and so should the constraints defined on the data. It is thus appropriate to monitor data evolution as a mirror of reality in order to get useful insights about the way data are evolving in time. To this aim, allowing some exceptions in the table makes it possible to obtain a better understanding of the data. In fact, a few rows might contain errors due to various noise factors or simply be an exception to the rule, and being able to detect the presence of unexpected exceptions may inform us that something has changed in the data semantics. To deal with this scenario, AFDs have been introduced, that is, FDs that are associated with some degree of approximation. AFDs are FDs that allow some rows to contain "errors" as exceptions to the rules; the more errors they allow, the more approximate they are, that is, their "approximation degree" changes. Thus, AFDs "bend but do not break" and coherently ACFDs have been introduced as an extension of CFDs.

The problem of constraint violation has been faced in the literature in different ways. Works such as [9, 10, 11, 12, 13, 14] propose strategies to query inconsistent databases trying to re-establish consistency by changing the facts that violate the constraints. Thus, the problem of inconsistent databases is considered from a query-answering point of view, that is, the data that can produce inconsistency with respect to the integrity constraints imposed at design time are discarded. By contrast, we aim at modifying the integrity constraints so the semantics of the database will adhere as much as possible to the changing reality. Therefore, the data that violate the constraints are not considered as abnormal facts but are used to update obsolete constraints. A similar approach was developed in [15] whose authors used data mining techniques to repair tuple constraints. In this work we extend their methodology to deal with functional dependencies.

FDs have been used as means to enforce data quality through data profiling and cleaning. In [16] the authors propose an algorithm for *discovering* Denial Constraints (which include functional dependencies) without supposing that any constraint has been specified on the database at design time. In order to apply the proposal in [16] to update the constraints on a given database when they are not up to date, one has: (i) first to discover all the possible constraints from data, then (ii) relax the constraints, considered as if they

have been specified at design time on the database schema, that do not hold on the current instance. This approach is rather impractical when the FDs, though obsolete, have been originally defined by a designer, first because of efficiency reasons, and second because, as we have noticed testing the on-line algorithm of [16] the inferred constraints not always include extensions of the ones specified by the designer.

Not many works have been proposed in the literature that, in the case of inconsistency, try to change the constraints instead of the data. The authors of [17, 18] have for the first time introduced a model that considers functional dependency repair. They illustrate a method to extend, by adding one attribute, the body of a violated FD in order to obtain a new dependency that is not violated anymore. In particular, given a functional dependency $F : X \rightarrow Y$ over a relation R, each attribute of R (other than X and Y) is evaluated as a candidate using the notion of *entropy* for comparing clusterings of tuples. As a result, a ranked list of candidate attributes is given to the designer. Given an FD that is violated by the data, the strategy in [17]: i) first computes a ground truth clustering of the data, based on the attributes in the FD; ii) then, for each attribute A, not present in the FD, computes a clustering of the data based on A and finally iii) computes the relative entropy (see Section 3) between this clustering and the ground truth clustering, which means comparing all its clusters with those in the ground truth clustering. The metric used in the work, which is the information variation requires frequent clustering of tuples in order to understand how good a candidate attribute is. We propose a very simple approach based on confidence and a measure of goodness that only require to count tuples. In Section 5 we explain the details of [17] and give a theoretical comparison between this work and ours. An experimental comparison between the two approaches was unfortunately impossible due to the unavailability of the tool presented in [17].

3. BASIC NOTIONS

Let $\mathcal{U}$ be a finite set of attribute names; we denote attributes by capital letters from the beginning of the alphabet (e.g., A, B, C, A_1, etc.), while capital letters from the end of the alphabet (e.g., U, X, Y, Z, X_1, etc.) are used to denote sets of attributes. Let $\mathcal{D}$ be a finite set of domains, each containing atomic values for the attributes; the domain D_j contains the possible values for the attribute A_j. A *relation schema* $R(A_1, A_2, ..., A_n)$ describes the structure of a relation whose name is R and whose set of attributes is $A_1, A_2, ..., A_n$. A relation instance r of relation R, is a finite set of tuples $t_1, t_2, \ldots, t_m$ of the form $t_h = (v_1, v_2, ..., v_n)$, where each value v_k, $1 \leq k \leq n$, is an element of D_k. $t[A_i]$ denotes the value assumed by the attribute A_i in the tuple t (i.e., v_i). Given an instance r of a relation R, we denote by $|r|$ the number of tuples in r and $|R|$ the number of attributes in R. Moreover, $\pi_X(r)$ is the projection of r on the attributes of X. The structure of a functional dependency is defined as follows:

Definition 1 (Syntax) *Given a relation schema R, a functional dependency F over R has the form $F : X \rightarrow Y$ where X and Y are sets of attributes in R.*

We use XY to denote the union of X and Y, moreover $|F| = |XY|$ is the number of attributes in the FD and, given two FDs F_1 and F_2, $|F_1 \cap F_2|$ is the number of attributes common to F_1 and F_2. An instance r of a relation schema R can either satisfy an FD or not, according to Definition 2.

Definition 2 (Semantics) *Given an instance r of a relation R, r satisfies F if, for every pair of tuples t_1, t_2 in r, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$.*

We say that an instance r is inconsistent with respect to F if it does not satisfy Definition 2.

Let us introduce a useful characterization of the notion of FD with the introduction of *confidence* and *goodness*.

Definition 3 *Let r be an instance of a relation R defined over a set of attributes S. Let X and Y be subsets of S and $F : X \rightarrow Y$ a functional dependency over R. The* confidence *of F w.r.t. r is:*

$$c_{F,r} = \frac{|\pi_X(r)|}{|\pi_{XY}(r)|}$$

while the goodness *of F w.r.t. r is:*

$$g_{F,r} = |\pi_X(r)| - |\pi_Y(r)|$$

Moreover, based on the value of the confidence of an FD, we have the following definition:

Definition 4 *Given a relation R, an instance r of R, a functional dependency F over R and its confidence $c_{F,r}$, we say that F is an* exact functional dependency *iff $c_{F,r} = 1$, otherwise it is an* approximate functional dependency*.*

All three FDs from Example 1 are approximate and have the following confidence and goodness values: $c_{F_1,Places} = 0.5$ and $g_{F_1,Places} = -2$; $c_{F_2,Places} = 0.667$ and $g_{F_2,Places} = -1$; $c_{F_3,Places} = 0.889$ and $g_{F_3,Places} = 1$. Note that the confidence of an FD reflects the average number of values of Y that are associated with each value of X in r. When the confidence is 1 it means that, for each distinct value of X, exactly one value of Y is associated with X, thus it is easy to see that exact FDs are the classical FDs of Definition 1:

The notion of functional dependency can be also formalized as a function between clusters of tuples.

Definition 5 (Clustering) *Given an instance r of a relation R and a set X of attributes of R, we call X-clustering a partition $\mathcal{C}_X = \{C_1, C_2, ..., C_K\}$ of r into mutually disjoint subsets C_i, with $i \in \{1, \ldots, K\}$, called classes (or clusters), such that each class C_i contains all the tuples of r that have the same value for the attributes in X.*

Given $F : X \rightarrow Y$, there are two clusterings naturally generated by F: $\mathcal{C}_X$ and $\mathcal{C}_Y$. Intuitively, if each cluster in $\mathcal{C}_X$ is associated with only one cluster in $\mathcal{C}_Y$ (that is, if there is a function between classes in $\mathcal{C}_X$ and classes in $\mathcal{C}_Y$) then F is satisfied, otherwise it is not. Consider as an example

$$F_1 : [District, Region] \rightarrow [AreaCode]$$

The two clusterings $\mathcal{C}_{District,Region}$ and $\mathcal{C}_{AreaCode}$ are shown in Figure 2a. As we can see, the relation between the two clusterings is not a function because there are some tuples, having the same value of $District, Region$ that are associated with sets having different values of $AreaCode$. In fact, F_1 is violated by the data.

To understand whether F is satisfied or not, we consider the two clusterings $\mathcal{C}_X$ and $\mathcal{C}_{XY}$. Since $\mathcal{C}_{XY}$ is finer-grained than $\mathcal{C}_X$ we always have: $|\mathcal{C}_{XY}| \geq |\mathcal{C}_X|$. When $|\mathcal{C}_{XY}| > |\mathcal{C}_X|$,

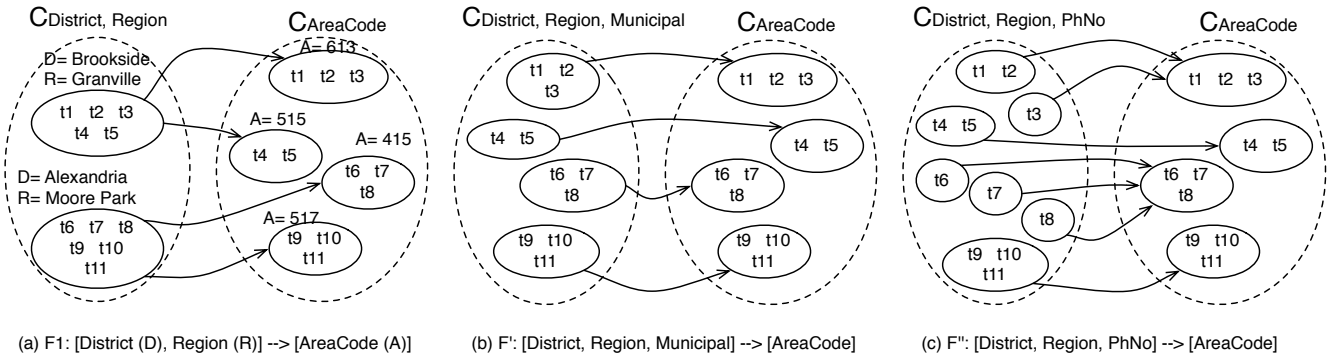

Figure 2: FDs clusterings

it means that there exists at least one class C_x in $\mathcal{C}_X$ whose tuples form more than one class in $\mathcal{C}_{XY}$, thus, the relation between $\mathcal{C}_X$ and $\mathcal{C}_Y$ is not a function. Only when $|\mathcal{C}_{XY}| = |\mathcal{C}_X|$, we have that each class in $\mathcal{C}_X$ forms one class in $\mathcal{C}_Y$ and therefore F is satisfied. If there is a function between $\mathcal{C}_X$ and $\mathcal{C}_Y$, such function is surjective, because each tuple in the database contains values for both X and Y[1], thus each value of Y is necessarily associated with at least one value of X. Now, the function could be injective (and thus bijective) or not. In particular, we introduce the following definition:

Definition 6 (Proper association) *Given two clusterings $\mathcal{C}_X$ and $\mathcal{C}_Y$, we say that a class C_x in $\mathcal{C}_X$ is* properly associated *with a class C_y in $\mathcal{C}_Y$, when C_y is the unique class in $\mathcal{C}_Y$ such that $C_x \subseteq C_y$.*

The concept of proper association is crucial for defining an FD in terms of a function between two given clusterings. Given $F : X \rightarrow Y$, when for every class C_x in $\mathcal{C}_X$ there is a class C_y in $\mathcal{C}_Y$ which is properly associated with C_x (i.e. C_y is the unique class that contains C_x), one usually says that the clustering $\mathcal{C}_X$ is *h*omogeneous with respect to $\mathcal{C}_Y$. Accordingly, we say that there is *a well-defined function* between the classes in $\mathcal{C}_X$ and those in $\mathcal{C}_Y$. Intuitively, a well-defined function is bijective.

From our point of view, FDs that allow us to obtain a well-defined function are to be preferred to other FDs. Consider $F' : [District, Region, Municipal] \rightarrow [AreaCode]$ and $F'' : [District, Region, PhNo] \rightarrow [AreaCode]$; the clusterings $\mathcal{C}_X$ and $\mathcal{C}_Y$ generated by the two FDs are shown in Figure 2b and Figure 2c respectively. As we can see, in both cases there is a function between $\mathcal{C}_X$ and $\mathcal{C}_Y$, however, F' allows us to obtain a well-defined function while F'' does not. Intuitively speaking, the municipality is "a better choice" than the phone number.

To explain this intuition, consider again that a non-satisfied $F : X \rightarrow Y$ generates two clusterings $\mathcal{C}_X$ and $\mathcal{C}_Y$ where the number of clusters in $\mathcal{C}_X$ is smaller than the number of clusters in $\mathcal{C}_Y$. To obtain a function between the two clusterings, we need to "further fragment" $\mathcal{C}_X$ so that there is the same number of clusters as in $\mathcal{C}_Y$, or more: in our method we do this by adding attributes to X.

Note that the number of clusters in $\mathcal{C}_X$ gives us an idea of how "specific" is a set of attributes X, that is, if the number of clusters is 1 all the tuples in the relation are put in the same cluster, while, in the extreme case where each tuple is assigned to a distinct cluster, X is actually a candidate key.

This second case will surely happen if, while repairing F, we add to its antecedent an attribute that has the property of being UNIQUE in the relation. Adding a UNIQUE attribute allows to repair any FD because this attribute alone determines Y thus it practically makes X "useless" in the FD. Something very close to this case happens with the two dependencies F' and F'' above: F' is "better" than F'' because adding the municipality to the antecedent *allows to generate two clusterings that are associated by a function such that the specificity of the domain is as much as possibile similar to the specificity of the codomain*, while, due to the high specificity of the phone number, adding it would make the antecedent too specific (also making the other attributes "almost useless"). We will see that our repairing method discourages the addition of such attributes (they are penalized through the *goodness* coefficient) and instead prefers those attributes that fragment $\mathcal{C}_X$ just as much as it is necessary to reach the same specificity as $\mathcal{C}_Y$.

Notice the relationship between these concepts and the confidence and goodness of an FD F:

- F is *exact* when there is a well-defined function from $\mathcal{C}_X$ to $\mathcal{C}_Y$, that is, the *confidence coefficient* of an FD somehow measures the "degree of being a function" ($c_{F,r} \leq 1$).

- When this pleasant fact happens ($c_{F,r} = 1$), then the *goodness coefficient* measures how far our function is from being injective. In fact it is injective when $g_{F,r} = 0$. In any case, when $c_{F,r}$ is different from 1, the *goodness coefficient* measures the distance of our approximate FD from having the domain with the same cardinality of the co-domain.

Therefore the case $\{c_{F,r} = 1, g_{F,r} = 0\}$ is such that to each class in $\mathcal{C}_X$ is associated one and only one class in $\mathcal{C}_Y$: the corresponding FD allows to obtain a bijective function between the two clusterings and so it is the best function which cannot be further improved. In fact, from a well-known result on functions [2], since the correspondence F is surjective and $|\pi_X(r)| = |\pi_Y(r)|$, it is therefore bijective. Finally we

[1]Note that attributes involved in FDs do not contain NULL values

[2]Let $f : X \longrightarrow Y$ be a function. If $|X| = |Y|$, then f is injective if and only if f is surjective.

notice that the *goodness coefficient* can be positive or negative. It is positive when the domain cardinality is higher than the co-domain one and negative instead when the cardinality of the domain is smaller.

As a final remark, notice that, if the DB schema is in a higher normal form, the only non-trivial FDs are those determining candidate keys. However, we believe this to be a strong assumption, especially nowadays, since NoSQL, semi-structured and other types of poorly organized data are widely used. Thus, we do not rely on the assumption that a database is in a higher normal form and obtaining some insight about the data becomes very important in such a scenario.

4. EVOLVING FUNCTIONAL DEPENDENCIES

The goal of our method is to: (i) understand which FDs are violated and (ii) repair these FDs by adding attributes to the antecedent of the dependency.

Objective *Given an FD $F : X \rightarrow Y$, not satisfied by the data, our aim is to find a minimal set of attributes U such that the new FD $F^U : XU \rightarrow Y$ is satisfied by the data.*

Given a relation schema R, an instance r of R, and all the FDs defined on it, for each FD, $F : X \rightarrow Y$, we compute its confidence. If the result is lower than 1 then the FD is not satisfied and, to repair it, we look for a set of attributes U in $R \setminus XY$ such that, if added to the antecedent of F, generate a new dependency $F^U : XU \rightarrow Y$ whose confidence is 1.

4.1 FD ordering

If an instance r of a relation R violates more than one constraint we need to decide the order in which they should be repaired. To do this, similarly to [17], for each FD F we compute a rank that is the average of two indicators:

1. $ic_{F,r}$: the "degree of inconsistency" of F with respect to the relation r: $ic_{F,r} = 1 - c_{F,r}$
2. cf_F: the "conflict score" of F with respect to the other FDs defined on R, which is independent of any specific instance and depends on the number of attributes that F has in common with these FDs:

$$cf_F = \frac{\sum_{F' \in \mathcal{F}} \frac{|F \cap F'|}{max(|F|,|F'|)}}{|\mathcal{F}|}$$

where $\mathcal{F}$ is the set of FDs defined on R.

The final rank is the average of these two indicators:

$$O_F = \frac{ic_{F,r} + cf_F}{2}$$

Given a set $\mathcal{F}$ of FDs, we sort $\mathcal{F}$ according to the rank O_F of each $F \in \mathcal{F}$ and then follow this order to repair the FDs. Consider the relation in Figure 1 and the three FDs F_1, F_2 and F_3 in Example 1. The constraints should be examined in the following order: F_1 (0.25), F_2 (0.167), F_3 (0.056).

4.2 Candidate-repair ordering for an FD

Given an instance r of a relation R, an FD $F : X \rightarrow Y$ and an attribute A which is a candidate to extend the antecedent of FD, we first compute the confidence of the candidate FD $F^A : XA \rightarrow Y$ as:

$$c_{F^A,r} = \frac{|\pi_{XA}(r)|}{|\pi_{XAY}(r)|}$$

and then provide an ordered list of candidate attributes sorted in descending order of $c_{F^A,r}$. However, this ranking does not allow us to distinguish which attribute is better when the FDs they produce have the same confidence. Consider $F_1 : [District, Region] \rightarrow [AreaCode]$ defined on table *Places*. Intuitively, if we add the attribute $Municipal$ to the antecedent of F_1 we obtain an exact FD and, similarly, if we instead add the attribute $PhNo$ we also obtain an exact FD. Now, is it more reasonable to add the municipality or the phone number as attribute that is part of a functional dependency? Which one is better and why?

To provide a better ranking, following the intuition explained in Section 3 we use the goodness of F^A, which depends on the number of distinct values A assumes, as shown in Definition 3:

$$g_{F^A,r} = |\pi_{XA}(r)| - |\pi_Y(r)|$$

Once we have computed both the confidence and the goodness of each attribute in $R \setminus XY$, we produce, for each violated FD, a ranked list of candidate attributes, sorted first according to $c_{F^A,r}$ and then, as secondary sorting key, according to $g_{F^A,r}$. Consider $F_1 : X \rightarrow Y$ with $X = [District, Region]$ and $Y = [AreaCode]$. The confidence and goodness of F_1 are:

$$c_{F_1,Places} = \frac{|\pi_{District,Region}(Places)|}{|\pi_{District,Region,AreaCode}(Places)|} = \frac{2}{4} = 0.5$$

$$g_{F_1,Places} = |\pi_{District,Region}(Places)| - |\pi_{AreaCode}(Places)| = \\ = 2 - 4 = -2$$

F_1 is not satisfied by the tuples in *Places*, thus, for each candidate attribute A in relation *Places* we compute the two parameters $c_{F_1^A,Places}$ and $g_{F_1^A,Places}$ (Table 1 shows the results). The two candidate attributes $Z_1 = [Municipal]$

A	$c_{F_1^A,Places}$	$g_{F_1^A,Places}$
Municipal	$4/4 = 1$	0
PhNo	$7/7 = 1$	3
Street	$7/8 = 0.875$	3
Zip	$4/5 = 0.8$	0
City	$4/5 = 0.8$	0
State	$3/5 = 0.6$	-1

Table 1: Evolving $F_1 : [District, Region] \rightarrow [AreaCode]$

and $Z_2 = [PhNo]$ allow us to obtain new *exact* FDs, while every other attribute does not. Moreover, attribute Z_1 has a better rank because it allows to discriminate the distinct values of $District$, $Region$, $AreaCode$ in a homogeneous way.

4.3 When more than one attribute is needed

Until now we have assumed that we repair an FD by adding only one attribute to its antecedent. Of course, it might happen that adding only one attribute is not enough to obtain an exact new FD. In this case, we can either stop or try to find a "more specific" FD by adding more attributes to

its antecedent. We handle this scenario as an iterative process where, at each step of iteration, the method presented so far is applied. Thus, at each step we have to choose the next attribute to be added to the antecedent and we do so by adding the attribute that produces the candidate FD with the highest rank. Consider $F_4 : X \rightarrow Y$ with $X = [District]$ and $Y = [PhNo]$, whose confidence and goodness are:

$$c_{F_4,Places} = \frac{|\pi_{District}(Places)|}{|\pi_{District,PhNo}(Places)|} = \frac{2}{7} = 0.29$$

$$g_{F_4,Places} = |\pi_{District}(Places)| - |\pi_{PhNo}(Places)| = \\ = 2 - 6 = -4$$

Thus, F_4 is not satisfied, and Table 2 shows the ranking of the attributes which are candidates to extend it.

A	$c_{F_4^A,Places}$	$g_{F_4^A,Places}$
Street	0.875	1
Municipal	0.571	-2
AreaCode	0.571	-2
City	0.571	-2
Zip	0.5	-2
State	0.429	-3
Region	0.286	-4

Table 2: Evolving $F_4 : [District] \rightarrow [PhNo]$

As we can see, there is no attribute that allows us to obtain an exact new FD thus we proceed by adding the attribute that generates the candidate FD with the highest rank. We obtain a new FD which is still not exact and then apply our method again to look for attributes that can be added to its antecedent. Therefore, we add attribute *Street* to the antecedent of F_4 and obtain $F_4^{Street} : [District, Street] \rightarrow [PhNo]$. Table 3 shows the ranking of the attributes which are the candidates to extend F_4^{Street}.

B	$c_{F_4^{Street,B},Places}$	$g_{F_4^{Street,B},Places}$
Municipal	1	4
AreaCode	1	4
Zip	0.889	4
City	0.875	4
State	0.875	3

Table 3: Evolving $F_4^{Street} : [District, Street] \rightarrow [PhNo]$

We have found two attributes, *Municipal* and *AreaCode*, that allow to extend $F_{4'}$ and obtain an exact new FD. Therefore, the two pairs *Street*, *Municipal* and *Street*, *AreaCode* allow to extend the antecedent of F_4 and obtain an exact new FD. They score the same value also for the goodness thus they are actually equivalent w.r.t. our aim. In such a case, it is for the designer to choose which one is more significant w.r.t. the application scenario.

4.4 Algorithm

Our approach is formalized by Algorithm 1 which receives as input a relation R (both instance and schema) and the set $\mathcal{F}$ of FDs defined over R. First of all, the function **OrderFDs** (line 2) orders all FDs according to the rank introduced in Section 4.1. Then, for each functional dependency F, the algorithm computes its confidence $c_{F,r}$ (line 4) in order to understand whether the FD is satisfied or not. If it is not satisfied, it calls the **ExtendByOne** function that computes the confidence and goodness (w.r.t. F) of all attributes in R other than those that are already in F (lines 3 and 4 in Algorithm 2) and returns the set of all candidates sorted according to their rank. Finally, if the considered attribute allows to obtain an exact new FD, it is added to the set of exact new FDs (line 9 in Algorithm 1).

Algorithm 1 `FindFDRepairs` (pseudocode).

Input: R the schema of a relation
r the instances of R
$\mathcal{F}$ the functional dependecies defined on R
Output: *Exact* the set of exact FDs obtained

```
1: Exact = ⟨⟩; Cand = ⟨⟩
2: FDs = OrderFDs(F)
3: for all F : X → Y ∈ F do
4:    c_{F,r} = |π_X(r)| / |π_XY(r)|
5:    if c_{F,r} < 1 then
6:       Cand = ExtendByOne(F, R, r)
7:       for all ⟨F', c_{F',r}, g_{F',r}⟩ ∈ Cand do
8:          if c_{F',r} = 1 then
9:             Exact =addInOrder(Exact, ⟨F', c_{F',r}, g_{F',r}⟩)
10:         end if
11:      end for
12:   end if
13: end for
14: return Exact
```

Notice that the computation of confidence and goodness can be implemented using SQL queries. In fact, the values $|\pi_X(r)|$, $|\pi_{XY}(r)|$,$|\pi_{XA}(r)|$, $|\pi_{XAY}(r)|$, $|\pi_{XY}(r)|$, $|\pi_A(r)|$ used in the algorithm are computed by counting the number of distinct tuples over the set of attributes involved in the antecedent and consequent of the FD. For example, the confidence of F_1, is the ratio between the results of:

Q_1:
```
select count(distinct District, Region)
from Places
```

Q_2:
```
select count(distinct District, Region, AreaCode)
from Places
```

The computation of these queries heavily depends on the query plan implemented by the DBMS and on the presence of supporting data structure such as indices. What we can say is that, considering the worst case scenario where no optimization techniques are implemented, we have a $O(n \log n)$ complexity because counting the distinct values corresponds to a sorting ($O(n \log n)$) followed by counting ($O(n)$). Moreover, looking for a single attribute at a time to extend the antecedent of an FD is linear with respect to the number of attributes in the relation.

However, when looking for repairs that contain more than one attribute, things get more complicated because the number of candidate repairs grows exponentialy with respect to the number of attributes. To limit this problem we use a greedy algorithm that chooses the FD candidates according first to the number of attributes in their antecedent and then to their rank. To this aim, we modify Algorithm 1 by introducing a queue that contains candidate repairs sorted by increasing cardinality of the antecedent and decreasing

Algorithm 2 `ExtendByOne` (F, R, r) pseudocode.

1: $Cand = \langle\rangle$
2: **for all** $A \in R \setminus XY$ **do**
3: $\quad c_{F^A,r} = \frac{|\pi_{XA}(r)|}{|\pi_{XAY}(r)|}$
4: $\quad g_{F^A,r} = |\pi_{XA}(r)| - |\pi_Y(r)|$
5: $\quad$ **if** $c_{F^A,r} = 1$ **then**
6: $\quad\quad Cand = \text{addInOrder}(Cand, \langle F^A, c_{F^A,r}, g_{F^A,r}\rangle)$
7: $\quad$ **end if**
8: **end for**
9: **return** $Cand$

rank: given an FD F that needs to be evolved, Algorithm 3 first generates all the candidates obtained by adding one attribute to the antecedent of F (line 1) and inserts them into a queue ordered by decreasing rank (line 2). Then, one at the time, it removes the first candidate in the queue (line 4); if its confidence is 1, it adds it to the set of candidates that allow to find an exact FD (line 6), otherwise it generates all the candidates obtained by adding one attribute to the antecedent of the FD (line 8) and inserts them into the queue, again sorted first by increasing cardinality of their antecedent and then by decreasing rank (line 9). The process is repeated while there are still candidates left in the queue and, at the end of the process, the algorithm returns the set of all candidates that allow to obtain an exact FD (line 12). Notice two things: (i) the stop condition of the algorithm can be easily changed to end when the first repair is found (in this way the algorithm does not need to explore the whole search space); (ii) since the candidates in the queue are ordered first according to the number of attributes in their antecedent (and then according to their rank), the first repair found is also a minimal one, that is, it contains the minimum number of attributes that need to be added to the antecedent of the considered FD to repair it.

Algorithm 3 `Extend` (F, R, r) (pseudocode).

1: $Cand = \textbf{ExtendByOne}(F, R, r)$
2: $addInOrder(Queue_F, Cand)$
3: **while** $Queue_F$ *not empty* **do**
4: $\quad \langle F', c_{F',r}, g_{F',r}\rangle = removeFirst(Queue_F)$
5: $\quad$ **if** $c_{F',r}$=1 **then**
6: $\quad\quad$ addInOrder($Exact$, $\langle F', c_{F',r}, g_{F',r}\rangle$)
7: $\quad$ **else**
8: $\quad\quad Cand = \textbf{ExtendByOne}(F', R, r)$
9: $\quad\quad addInOrder(Queue_F, Cand)$
10: $\quad$ **end if**
11: **end while**
12: **return** $Exact$

The complexity of managing the queue corresponds to the complexity of a sorting algorithm. Of course, the number of candidates inserted into the queue is exponential with respect to the number of attributes in the relation. In order to find all candidates that allow to obtain an exact FD we need to explore the whole search space. However, this does not happen if we decide to stop when we find the first candidate instead. In the latter case, given the ordering we use to explore the search space, we can ensure that we reach our objective, as stated at the beginning of Section 4, that is, that we are able to find a minimal repair for a given FD.

Notice that a minimal repair might not always be the best choice. Suppose we are trying to repair a given FD $F : X \rightarrow Y$ and suppose there are two ways to do it: i) we can add attribute A which has the property of being UNIQUE and ii) we can add the two attributes B and C, neither of whom is UNIQUE. Of course, since we privilege shorter repairs, the algorithm will privilege the first repair which unfortunately goes against what we have discussed in Section 3. To address this drawback we are currently investigating the use of a user-specified maximum goodness threshold. The idea is to use it to privilege those repairs whose goodness is lower than the threshold. In particular, we are currently considering combining such a threshold with our confidence and goodness measures in order to provide an objective function that guides our repair strategy.

5. THEORETICAL COMPARISON WITH THE ENTROPY-BASED APPROACH

The technique presented in [17] finds attributes that are good candidates to extend the antecedent of an FD, based on its variation of information, which in its turn is based on the entropy measure. In this section we dub this Entropy-Based method EB, and compare it with ours (CB) which is based mainly on the Confidence measure.

Even though the aim of the EB method is the same as ours, there are some differences between them: the first difference is that CB easily supports the evolution of an FD by adding more than one attribute in its antecedent; second, and more important, to understand if an attribute is a good candidate to extend the FD, we only compute its confidence, while with EB more complex computations are needed. To discuss this, we need to introduce formally the notion of *Entropy*.

Given two clusterings $\mathcal{C}$ and $\mathcal{C}'$, we can compute the *Variation of Information* (VI) [19] between them as the sum of the two conditional entropies:

$$VI(\mathcal{C}, \mathcal{C}') = H(\mathcal{C}|\mathcal{C}') + H(\mathcal{C}'|\mathcal{C})$$

where the conditional entropy of $\mathcal{C}$ given $\mathcal{C}'$ is defined as:

$$H(\mathcal{C}|\mathcal{C}') = -\sum_{k=1}^{K}\sum_{k'=1}^{K'} P(k,k')logP(k|k')$$

where: $P(k,k') = \frac{|C_k \cap C'_{k'}|}{n}$ is the joint probability distribution associated to the pair $(\mathcal{C}, \mathcal{C}')$,

$$P(k|k') = \frac{P(k,k')}{P(k')} = \frac{|C_k \cap C'_{k'}|}{|C'_{k'}|}$$

is the conditional probability distribution associated to $\mathcal{C}$ given $\mathcal{C}'$, and $P(k') = \frac{|C_{k'}|}{n}$ the marginal probability distribution associated to $\mathcal{C}'$. Note that $VI(\mathcal{C}, \mathcal{C}')$ is symmetric with respect to the two clusterings.

Given $F : X \rightarrow Y$, the EB method creates a ground truth clustering $\mathcal{C}_{XY}$ and then looks for an attribute A that, when added to the antecedent of F, allows to obtain a clustering $\mathcal{C}_{XA}$ that is either homogeneous or, preferably, homogeneous and *complete* w.r.t. $\mathcal{C}_{XY}$ (i.e., with VI equal to zero).

Note that, with EB, for each FD, the algorithm computes a ground truth clustering that is obtained by scanning all tuples and grouping them according to the attributes in the FD; then, for each attribute A in the relation, the algorithm computes the clustering $\mathcal{C}_A$ of the relation in the same manner; and finally the two clusterings must be compared by

computing the intersections of all pairs of clusters in order to determine the variance of information. This last action requires, for each cluster in the ground truth clustering, to scan all clusters in $\mathcal{C}_A$. Thus, the EB method requires to store the tuples in order to be able to perform the intersections between clusters while with the CB technique we do not keep trace of all tuples in the groups but only of their amount.

We now show that the *confidence* and *goodness* parameters introduced in Section 3 can be successfully used instead of the conditional entropies and that these two simple parameters give rise to a measure which is equivalent to the VI measure.

Given $F : X \to Y$, the EB method chooses $\mathcal{C}_{XY}$ as the ground truth clustering and, for each attribute A, considers the clustering $\mathcal{C}_A$ in order to understand how well it matches $\mathcal{C}_{XY}$. This is done by taking advantage of the two conditional entropies involved in the definition of VI, but *not symmetrically*. In fact, a modified version of the VI is introduced that considers first the conditional entropy of $\mathcal{C}_{XY}$ given $\mathcal{C}_{XA}$, i.e. $H(\mathcal{C}_{XY}|\mathcal{C}_{XA})$, and then the conditional entropy of $\mathcal{C}_A$ given $\mathcal{C}_{XY}$, i.e. $H(\mathcal{C}_A|\mathcal{C}_{XY})$. Then, the EB approach selects the attribute A that has the lowest value of $H(\mathcal{C}_{XY}|\mathcal{C}_{XA})$ and, in the case of a tie, the attribute A with the lowest value of $H(\mathcal{C}_A|\mathcal{C}_{XY})$.

The first conditional entropy $H(\mathcal{C}_{XY}|\mathcal{C}_{XA})$ measures the *non homogeneity* property of $\mathcal{C}_{XA}$ with respect to $\mathcal{C}_{XY}$. In fact, it is easy to see that when a class $C_{xa} \in \mathcal{C}_{XA}$ is such that $C_{xa} \subseteq C_{xy}$ for some $C_{xy} \in \mathcal{C}_{XY}$, then

$$logP(C_{xy}|C_{xa}) = log\frac{P(C_{xy} \cap C_{xa})}{P(C_{xa})} = log\frac{P(C_{xa})}{P(C_{xa})} = 0$$

Thus, when $\mathcal{C}_{XA}$ is *homogeneous* with respect to $\mathcal{C}_{XY}$, the relative conditional entropy $H(\mathcal{C}_{XY}|\mathcal{C}_{XA})$ is zero. Similarly, the second conditional entropy $H(\mathcal{C}_A|\mathcal{C}_{XY})$ is zero when every class $C_{xy} \in \mathcal{C}_{XY}$ is such that $C_{xy} \subseteq C_a$ for some $C_a \in \mathcal{C}_A$. When this happens, the completeness property for $\mathcal{C}_A$ versus $\mathcal{C}_{XY}$ is verified.

The best attribute found by the EB technique is both *homogeneous* and *complete* implying that the VI is zero.

In the following we propose a slight variation of the EB approach of [17], based on the original definition on VI as in [19], i.e.

$$VI(\mathcal{C}_{XY}, \mathcal{C}_{XA}) = H(\mathcal{C}_{XY}|\mathcal{C}_{XA}) + H(\mathcal{C}_{XA}|\mathcal{C}_{XY})$$

This allows us to make the comparison clearer, while not affecting the results. Note that, given $F : X \to Y$, VI can be seen as a measure, denoted by $\varepsilon_{VI} := VI$, on all FDs $F^A : XA \to Y$, where A is, as introduced in Section 4.2, a candidate attribute to repair F. This measure is equal to zero when there is homogeneity and completeness between the two clusterings $\mathcal{C}_{XA}$ and $\mathcal{C}_{XY}$. Let us also introduce the measure ε_{CB}, based on our *confidence* and *goodness* coefficients of Definition 3:

$$\varepsilon_{CB} := ic_{F^A} + \hat{g}_{F^A}$$

where $ic_{F^A} := 1 - c_{F^A}$ is the "degree of inconsistency" introduced in Section 4.1 and $\hat{g}_{F^A} := |g_{F^A}|$ is the absolute value of the goodness coefficient. This measure is equal to zero when F^A allows to generate a bijective function between the classes of $\mathcal{C}_{XA}$ and those of $\mathcal{C}_Y$. We can state that the two measures are equivalent, i.e. they have the same null sets (the sets where they assume the null value) and, consequently, the same support sets (the sets where the measures assume a strictly positive value):

Theorem 1 *Let R be a relation schema and $F^Z : XZ \to Y$ a functional dependency defined on R as above. Then the measures ε_{CB} and ε_{VI} are equivalent.*

Proof 1 *First we observe that the CB best case $\{c_{FZ} = 1, g_{FZ} = 0\}$ corresponds to $\{\varepsilon_{CB} = 0\}$ and the EB best case $\{VI = 0\}$ corresponds to $\{\varepsilon_{VI} = 0\}$.*

Let us recall that given two measures P and Q, the measure P is absolutely continuous w.r.t. the measure Q (or P is dominated by Q) if for all A such that $Q(A) = 0$ one has that $P(A) = 0$, i.e. when the null sets of Q are also null sets of P. Moreover in this case by Radon-Nikodym theorem (see [20]) there exists a (positive) density f such that in differential form one has that $dP = fdQ$. If in addition Q is also absolutely continuous w.r.t. P, then the two measures are called equivalent. In particular in this case one can show that $dQ = f^{-1}dP$ ([20]).

We prove that ε_{VI} is absolutely continuous w.r.t. ε_{CB}. We put $B = F^Z : XZ \to Y$ and suppose that $\varepsilon_{CB}(B) = 0$. Then $\{c_{FZ} = 1\}$ and $\{g_{FZ} = 0\}$. When the confidence is equal to one we also have the homogeneity property of $\mathcal{C}_{XZ}$ versus $\mathcal{C}_{XY}$. In fact when $\{c_{FZ} = 1\}$ there is a proper association for every $C_{xz} \in \mathcal{C}_{XZ}$, that is:

$$\forall C_{xz} \in \mathcal{C}_{XZ} \quad \exists! \quad C_y \in \mathcal{C}_Y \quad s.t. \quad C_{xz} \subseteq C_y$$

Thus, $\mathcal{C}_{XZ}$ is homogeneous with respect to $\mathcal{C}_Y$ and it follows that $\mathcal{C}_{XZ}$ is homogeneous also with respect to $\mathcal{C}_{XY}$, i.e.:

$$\forall C_{xz} \in \mathcal{C}_{XZ} \quad \exists! \quad C_{xy} \in \mathcal{C}_{XY} \quad s.t. \quad C_{xz} \subseteq C_{xy}$$

In fact if there exists a C_{xz} whose tuples are contained in more than one class C_{xy}, it would be $|\mathcal{C}_{XZ}| < |\mathcal{C}_{XY}| \leq |\mathcal{C}_{XZY}|$. But this cannot happen, since

$$\{c_{FZ} = 1\} \Leftrightarrow |\mathcal{C}_{XZ}| = |\mathcal{C}_{XZY}|$$

Therefore, also the confidence coefficient of the CB method is a measure of the homogeneity property of $\mathcal{C}_{XZ}$ versus $\mathcal{C}_{XY}$: there is homogeneity when the confidence coefficient is one.

Moreover, when in addition also the goodness coefficient is zero, the clustering $\mathcal{C}_{XZ}$ has also the completeness property versus $\mathcal{C}_{XY}$, in the sense that $H(\mathcal{C}_{XZ}|\mathcal{C}_{XY}) = 0$. In fact, we recall that a clustering has the cited completeness property when

$$\forall C_{xy} \in \mathcal{C}_{XY} \quad \exists! \quad C_{xz} \in \mathcal{C}_{XZ} \quad s.t. \quad C_{xy} \subseteq C_{xz}$$

If there exists a class C_{xy} whose tuples are in part contained in some C_{xz} and in part in some other $\hat{C}_{xz}$, then it would be $|\mathcal{C}_Y| \leq |\mathcal{C}_{XY}| < |\mathcal{C}_{XZ}|$. But this cannot happen, since

$$\{g_{FZ,r} = 0\} \Leftrightarrow |\mathcal{C}_{XZ}| = |\mathcal{C}_Y|$$

Since the homegeneity plus completeness properties between the two clusterings implies $\epsilon_{VI} = 0$ we have proved that the measure ε_{VI} is dominated by the measure ε_{CB}.

We show that ε_{CB} is dominated by ε_{VI}. Let us suppose that $\varepsilon_{VI}(B) = 0$. Then $VI = 0$, which implies that the homogeneity and the completeness properties hold. As a conseuence the two clusterings are exactly equal, i.e. every single class in correspondence contains the same subset of tuples. In particular this means that: a) $|\mathcal{C}_{XY}| = |\mathcal{C}_{XZ}|$

b) $\forall y \quad \exists! \quad (x, z)$ and therefore $|\mathcal{C}_{XZ}| = |\mathcal{C}_Y|$

c) $\forall(x,z) \quad \exists! \quad y = z$ *and therefore* $|\mathcal{C}_{XZY}| = |\mathcal{C}_{XZ}|$ *and this implies* $ic_{F^z} = 0$ *and* $g_{F^z} = 0$*, i.e.* $\varepsilon_{CB}(B) = 0$.

Finally the two measures are equivalent: they have the same null sets and, consequently, the same support sets (that is the sets where the measures assume strictly positive values). Moreover by Radon-Nikodym theorem there exists a positive density f, such that

$$d\varepsilon_{CB}(B) = f(B)d\varepsilon_{VI}(B), \quad d\varepsilon_{VI}(B) = f^{-1}(B)d\varepsilon_{CB}(B).$$

The proof of Theorem 1 shows that the two measures have the same support sets (i.e. the sets where a measure assumes strictly positive values).

We remark that the measures ε_{VI} and ε_{CB} can be considered as acting on a general $F : X \to Y$, with X and Y sets of attributes in R, and in this case they assume respectively the form

$$\varepsilon_{VI}(F) = H(\mathcal{C}_{XY}|\mathcal{C}_Y) + H(\mathcal{C}_Y|\mathcal{C}_{XY})$$

and

$$\varepsilon_{CB}(F) := ic_F + \hat{g}_F.$$

Moreover it can be proved, in the same way as for Theorem 1, that they are equivalent measures.

Usually the equivalence relation between measures is rather weak: they have the same support sets but the values they assume can be very different. Since in our case the number of possible FDs is finite and our measures are finite too, the absolutely continuity property is a sort of continuity property between the two measures; more precisely, one can prove that, in our finite case, the mutual absolutely continuity property of ε_{CB} and ε_{VI} is equivalent to the following $\epsilon - \delta$ property:

$\forall\epsilon \quad \exists\delta \quad s.t. \quad \varepsilon_{CB}(A) < \epsilon$ for each A with $\varepsilon_{VI}(A) < \delta$ (and the same relation holds with ε_{CB} and ε_{VI} interchanged).

As a consequence, our measures are equivalent and assume comparable values in their support sets. Moreover both EB and CB are based on algorithms which are looking for exactly the sets where the two measures assume the value zero.

We want to stress the fact that the CB method is indeed simpler than the EB approach both from the conceptual and computational point of view. First, because CB uses the classical framework of set functions with well-known elementary mathematical concepts. Moreover, with CB we have to perform only a few cardinality computations of the notable clusterings associated to a given FD in order to achieve our ranked list of attributes, i.e. with no need to enter in the detailed structure of the involved clusterings.

We have compared our CB approach only with the EB method in [17]. We think that this comparison is sufficient for the following reasons. In [21] it has been shown that from an axiomatic point of view the best *approximation measure* for FDs is the *information dependency measure*. One can easily prove that the measure introduced in [21] for an arbitrary FD $F : X \to Y$ is a normalized version of the first conditional entropy entering in the VI, that is $H(\mathcal{C}_{XY}|\mathcal{C}_X)$. We observe that also what we call "degree of inconsistency" $ic_F = 1 - c_F$ can be seen as an *approximation measure* of how far the given FD is from generating an exact function and that from the proof of Theorem 1 one can deduce that our measure ic_F is equivalent to the *approximation measure* given by $H(\mathcal{C}_{XY}|\mathcal{C}_X)$. Since in [21] there is also an accurate and complete review of the *approximation measures* in the literature, we finally conclude that the comparison of our CB method with the EB approach proposed in [17] is sufficient. It is now quite clear that the CB method grants results that are fully comparable with those obtained by means of the EB method, with the important difference that the basic concepts and the required computations are much simpler.

6. EXPERIMENTAL RESULTS

We implemented our method in a Java prototype tool. Initially, users connect to a MySQL database and visualize its relations and all FDs defined on each relation; then, they are allowed to add other FDs to the ones that are already defined, and finally they can start the process of FD validation.

We tested the tool on both real and synthetic databases and studied the time needed to find FD repairs. Our study was conducted varying both the FDs and the number of attributes and tuples in the relations.

All the experiments were run on a Core i5 2.6 GHz PC with 4 GB of memory and Windows 8 x64 operating system.

6.1 Synthetic databases

We used DBGEN to independently generate three synthetic databases of different sizes: Table 4 shows the features of the generated relations in terms of numbers of attributes and tuples. For each instance of the three databases, we defined one FD on each relation and run our algorithm to understand how execution time varies depending on the dimension of the dataset. Notice that, as shown in Table 5, all FDs have one attribute in the antecedent and one in the body and by processing time we mean the time it took for the algorithm to find all possible repairs for the given FD.

		100MB	250MB	1GB
Table	arity	card.	card.	card.
customer	8	15 000	30 043	150 249
lineitem	16	601 045	1 196 929	6 005 428
nation	4	25	25	25
orders	9	149 622	301 174	1 493 724
part	9	20 000	40 098	199 756
partsupp	5	80 533	160 611	779 546
region	3	5	5	5
supplier	7	1 000	2 000	10 000

Table 4: TPC-H Databases Overview

Figure 3 shows, for the 1GB synthetic database, how the processing time varies depending on the number of attributes (Figure 3a), number of tuples (Figure 3b) and overall dimension of the table (Figure 3c).

We report only the plots related to the 1GB database because of space limitations. However, we noticed that the time needed to repair the FDs is higher for bigger datasets, but the trend is the same. In fact, the trends for the 100MB and 250MB databases are very similar to the one for the 1GB database, although on a smaller scale. This happens because the structure of the three datasets is the same, thus what changes is only the number of tuples. Synthetic datasets tend to behave in a somehow "uniform" way, thus we performed a study on real datasets, with the aim of understanding in what way the number of attributes and the number

Table	FD	100MB processing time	250MB processing time	1GB processing time
customer	$[name] \rightarrow [address]$	1s 276ms	2s 873ms	20s 657ms
lineitem	$[partkey] \rightarrow [suppkey]$	9m 42s 708ms	21m 20s 599ms	1h 59m 19s 884ms
nation	$[name] \rightarrow [regionkey]$	5ms	5ms	6ms
orders	$[custkey] \rightarrow [orderstatus]$	8s 621ms	19s 726ms	1m 57s 103ms
part	$[name] \rightarrow [mfgr]$	1s 3ms	1s 983ms	18s 561ms
partsupp	$[suppkey] \rightarrow [availqty]$	4s 450ms	10s 570ms	1m 3s 909ms
region	$[name] \rightarrow [comment]$	3ms	3ms	3ms
supplier	$[name] \rightarrow [address]$	74ms	141ms	717ms

Table 5: `FindFDRepairs processing times`

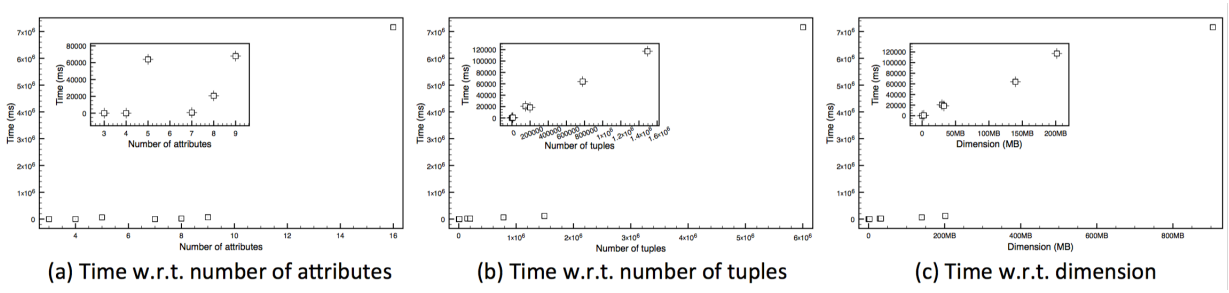

Figure 3: Processing times for the 1GB synthetic database

of tuples in a relation influence the algorithm.

6.2 Real-life databases

We conducted our experiments using the relations listed in Table 6, which also summarizes, for each of them, the number of attributes and tuples. On each relation we defined an FD containing one attribute in the antecedent and one in the consequent and then run our algorithm to find one possible repair (that is, the algorithm stops when it finds the first repair). Table 6 shows the results in terms of the time needed to execute the task.

Table	arity	card.	FD process time
Places	9	10	257ms
Country[3]	15	239	32ms
Rental[4]	7	16044	588ms
Image[5]	14	124768	2m 52s
PageLinks[6]	3	842159	4s 678ms
Veterans[7]	481	95412	29m 45s

Table 6: Real Databases Overview and processing times

We can see that relations with a higher number of attributes take longer time to be processed while the same does not hold in general for relations with high amounts of tuples. In fact, as we will see in the next Section, the time grows exponentially with the number of attributes while tuples do not influence it so much.

[3] `http://dev.mysql.com/doc/index-other.html`
[4] `http://dev.mysql.com/doc/index-other.html`
[5] `http://dumps.wikimedia.org/backup-index.html`
[6] `http://dumps.wikimedia.org/backup-index.html`
[7] `http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html`

There are many factors that influence execution time: these mainly concern the content of the tables (e.g. number of null values, number of candidates at one level that improve confidence, etc ...), whose impact on computation time needs to be studied in details. For example, in the results shown in Table 6 we have that even though Places is a smaller relation than Country (both in terms of attributes and tuples) it took longer to compute the results. This happens because for the first relation, longer repairs were needed, in fact, for table Places, the algorithm added 2 attributes to repair the given FD while for relation Country it added only 1 attribute. Moreover, we can see that even though table PageLinks is the biggest one in terms of tuples, it took less time to repair it than the Image table. This happens because the PageLinks relation has only three attributes, and since the FD defined on it has already two attributes (one in the antecedent and one in the consequent) the algorithm had to consider only the third one in the table. On the other hand, in the Image table, the algorithm had to add 2 attributes to the antecedent of the FD to repair it.

Nr. of tuples	Nr. of attributes 10	20	30
10K	26s	4m16s	17m34s
20K	38s	7m56s	35m1s
30K	57s	11m47s	51m48s
40K	2m13s	15m29s	1h28m12s
50K	2m44s	19m34s	1h48s
60K	3m17s	22m51s	1h56m3s
70K	5m13s	36m36s	2h23m8s

Table 7: Processing times for the Veterans relation – find all repairs

6.2.1 Case study

To better understand how the number of attributes and the number of tuples in a relation influence our algorithm, we performed a case study using the Veterans relation (see Table 6) which has 481 attributes (323 of which do not have null values) and 95412 tuples containing only non-null values. Notice that, since the attributes occurring in an FD are not allowed to contain NULL values, when generating candidate repairs, we only consider the addition of those attributes that do not contain NULLs. We created several instances of this relation, each containing a different number of attributes and tuples. We defined an FD containing one attribute in the antecedent and one in the consequent and then run our algorithm to find: (i) all possible repairs (Table 7 shows the results) and (ii) the first repair (results in Table 8). From these results we can confirm the evidence of the synthetic dataset study, namely, that execution time grows much quicker with the number of attributes in the relation than with the number of tuples. Computation time grows exponentially with the number of attributes while an increase in the number of tuples implies longer query processing times, but does not affect the algorithm processing time so much. Moreover, we can see that processing times are much smaller if the algorithm stops when it finds the first repair instead of exploring the whole search space. However, it might happen that the two times are very similar (e.g. in the 70K tuples DB with 10 attributes) when the algorithm is not able to find a repair for the given FD.

	Nr. of attributes		
Nr. of tuples	10	20	30
10K	8s76ms	53s96ms	2m23s
20K	18s22ms	1m30s	4m10s
30K	27s64ms	2m15s	6m12s
40K	1m25s	3m4s	8m18s
50K	1m47s	3m46s	10m38s
60K	2m10s	4m44s	12m51s
70K	5m23s	5m57s	16m10s

Table 8: Processing times for the Veterans relation – find the first repair

During the experiments we noticed that there are other parameters, generally application-dependent, that influence our method. Just to name a few: (i) the number of distinct values of an attribute: the more distinct values there are, the more time is needed to compute the queries; (ii) the initial confidence of an FD: as can be expected, the smaller the initial confidence, the greater the probability that a longer repair is needed, that is, the more attributes should be added in the antecedent, thus requiring more time; (iii) the average length of the repairs: if an FD needs repairs that add many attributes to the antecedent it will require more computation time. These parameters are related to each other, depend on the domain of application and are very difficult to control and predict.

6.3 Quality of results

We claim that our criterion for choosing the order in which attributes are added to the antecedent of functional dependencies favours the quality of the results we obtain. Indeed, as we have already discussed, our method privileges the addition of attributes that allow us to obtain functions that are "as well defined as possible", by approximating the goodness to 0. This is because we try to construct an FD that resembles as much as possible a bijective function, mapping the clusters generated by the antecedent of the FD to the clusters generated by the consequent of the FD. This choice allows us to:

- discourage the addition of a UNIQUE attribute: indeed, that would make the rest of the antecedent useless, because that attribute alone determines the consequent of the FD;
- along the same line of thought, encourage the addition of attributes that make the "specificity" of the antecedent of the FD is as much as possible similar to the "specificity" of the consequent;
- support indexing and query optimization, because, when the method manages to find "invertible" FDs, not only the antecedent determines the consequent but also vice-versa; thus, an index built on the antecedent of an FD can be used to efficiently access the attributes in the consequent (if we know the correspondence between the clusters in the antecedent and the clusters in the consequent).

7. CONCLUSION

In this paper we proposed a new method for repairing FD violations that works at the intensional level: rather than changing the data, it repairs the FD by adding one or more attributes to its antecedent. To this aim we have used the notions of confidence and goodness of an FD, as measures to estimate if an FD is violated by the data and to what extent. In future we intend to extend the method to other kinds of constraints and to make a more extensive study on the parameters that influence the processing time and on the impact they have when examining a database.

8. ACKNOWLEDGEMENTS

This research has been partially funded by the Italian project SHELL CTN01_00128_111357 and by the IT2Rail project, funded by European Union's Horizon 2020 research and innovation programme under grant agreement No: 636078.

9. REFERENCES

[1] T. Murata and A. Borgida. Handling of irregularities in human centered systems: A unified framework for data and processes. *IEEE Transaction on Software Engineering*, 26(10), 2000.

[2] G. Cugola, E. Di Nitto, A. Fuggetta, and C. Ghezzi. A framework for formalizing inconsistencies and deviations in human-centered systems. *ACM Trans. Software Eng. and Methodology*, 5(3):191–230, 1996.

[3] L. Caruccio, V. Deufemia, and G. Polese. Relaxed functional dependencies - A survey of approaches. *IEEE Trans. Knowl. Data Eng.*, 28(1):147–165, 2016.

[4] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011.

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

[6] H. Nakayama, A. Hoshino, C. Ito, and K. Kanno. Formalization and discovery of approximate conditional functional dependencies. In *DEXA*, pages 118–128, 2013.

[7] J. Wijsen. Temporal dependencies. In *Encyclopedia of Database Systems*, pages 2960–2966. 2009.

[8] C. Combi, P. Parise, P. Sala, and G. Pozzi. Mining approximate temporal functional dependencies based on pure temporal grouping. In *ICDM Workshops*, pages 258–265, 2013.

[9] L. E. Bertossi and J. Chomicki. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2003.

[10] S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Querying and repairing inconsistent xml data. In *Web Information System Engineering*, volume 3806 of *LNCS*, pages 175–188, 2005.

[11] S. Flesca, F. Furfaro, and F. Parisi. Consistent query answers on numerical databases under aggregate constraints. In *DBPL Workshops*, volume 3774 of *LNCS*, pages 279–294, 2005.

[12] J. Chomicki. Consistent query answering: Opportunities and limitations. In *DEXA*, pages 527–531. IEEE Computer Society, 2006.

[13] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005.

[14] J. Chomicki. Consistent query answering: Five easy pieces. In D. Suciu T. Schwentick, editor, *Proceeding of ICDT'07*, volume 4353 of *LNCS*, pages 1–17, 2007.

[15] M. Mazuran, E. Quintarelli, R. Rossato, and L. Tanca. Mining violations to relax relational database constraints. In *DaWaK*, pages 339–353, 2009.

[16] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[17] F. Chiang and R.J. Miller. A unified model for data and constraint repair. In *ICDE 2011*, pages 446 –457, 2011.

[18] J. Segeren, D. Gairola, and F. Chiang. CONDOR: A system for constraint discovery and repair. In *CIKM*, pages 2087–2089, 2014.

[19] M. Meilă. Comparing clusterings—an information based distance. *J. Multivar. Anal.*, 98(5):873–895, 2007.

[20] P. Billingsley. *Ergodic Theory and Information*. Wiley, 1965.

[21] C. Giannella. An axiomatic approach to defining approximation measures for functional dependencies. In *ADBIS*, pages 37–50, 2002.