2019 16th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)
Mexico City, Mexico. September 11-13, 2019

Inferring Functional Dependencies through Similarity Functions in a Crime Database

Zelzin M. Márquez-Navarrete
Computer Science Department
CINVESTAV
Mexico City, Mexico
zmarquez@computacion.cs.cinvestav.mx

Guillermo B. Morales-Luna
Computer Science Department
CINVESTAV
Mexico City, Mexico
gmorales@cs.cinvestav.mx

***Abstract*—We describe a methodology to find functional dependencies using the TANE algorithm. We also modify the TANE, so we can use a similarity function to find dependencies ignored due to precision of the exact method. These methods let us generate automatic knowledge discovery in appropriate datasets. In datasets where many-to-many relationships exists it is useful for data cleaning process and database design.**

***Index Terms*—crime, functional dependencies, database, similarity functions, data cleaning**

I. Introduction

Functional dependencies are a common tool used for mining information in datasets. Such dependencies might be used to establish relationships among the attributes of a database and contribute to automatic knowledge discovery. Thus we propose a methodology that could help us to find those relationships by seeking functional dependencies in sociodemographic datasets.

A functional dependency establishes a relationship between two sets of attributes in a relation, in such way that the value of the first set determines the value of the second. These kind of relationships are useful to perform automatic knowledge discovery, data mining, machine learning, normalization, etc.

However, in order to establish a functional relation between two attributes, the inference process assumes that the data is free from anomalies that could prevent such relationships from being found, e.g., errors during the collection process. In some cases, the very nature of the data makes difficult for the inference process to establish a functional relation, e.g., small differences for the value of an attribute.

The rest of the paper is structured as follows: in Section II we review the theory of relational databases and survey the existing algorithms used to infer functional dependencies. Section III presents previous works about the use of similarity functions in the inference of functional dependencies. In Section IV we introduce the process of data cleaning. We present the methodology used to analyze a dataset of crime occurrences in Section V, in particular we show the transformations made on the data so the calculation of FDs can be performed. Section VI contains our results. First we show the results of using the TANE algorithm to obtain exact FDs in the crimes dataset and present the relevant benchmarks. Next, we analyze the results of applying our modified TANE algorithm to the crimes dataset. By using this algorithm it is possible to uncover relationships in the dataset that would otherwise remain undetected. Finally, Section VII contains our conclusions.

II. Functional dependencies in relational datasets

Let $\mathcal{U}$ be a finite set of attributes denoted by $\{A_1, \ldots, A_n\}$. A relation r over $\mathcal{U}$ is a set of tuples $\{h_1, \ldots h_m\}$, where each h_i is given by:

$$h_i : \mathcal{U} \to \bigcup_{A \in \mathcal{U}} dom(A), \quad h_i(A) \in dom(A)$$

where $dom(A)$ is a map that defines the domain of each attribute $A \in \mathcal{U}$. A functional dependency (FD) is a relation given by the expression [1]:

$$X \to A, \quad X \subseteq \mathcal{U}, \quad A \in \mathcal{U} \tag{1}$$

where:

- $X \to A$ means that X functionally determines A, i.e., given the value of X it is possible to uniquely determine the value of A.
- A FD $X \to A$ holds in a relation r if for any pair of tuples $h_i, h_j \in r$, $h_i(B) = h_j(B), \forall B \in X \implies h_i(A) = h_j(A)$, $i \neq j$.
- A FD is minimal if A does not functionally depend on any proper subset of X, i.e., if removing an attribute from X makes the dependency invalid.
- A FD $X \to A$ is trivial if $A \in X$.
- A FD defined as in Equation 1 is also called exact functional dependency.

Given these definitions the following problem can be stated: find all minimal and nontrivial exact functional dependencies that hold on a relation r.

This problem can be solved using a brute force approach using the following algorithm:

We thank Mexican CONACYT for the given support.

978-1-7281-4840-3/19/$31.00 ©2019 IEEE

INPUT: A relation r.
OUTPUT: A set F containing all FDs in r.

```
1: procedure FIND ALL FDS(r)
2:     F ← ∅
3:     for X ∈ P(U) do
4:         for A ∈ U\X do
5:             if r ⊨ X → A then
6:                 F ← F ∪ {X → A}
7:             end if
8:         end for
9:     end for
10: end procedure
```

This algorithm simply analyzes all elements in the power set of $\mathcal{U}$ (trivial FDs are not considered). Each $X \in \mathcal{P}(\mathcal{U})$ represents a candidate FD for some $A \in \mathcal{U}$, so the algorithm verifies if such dependency holds in r. The complexity of such algorithm is $O(n^2 2^n |r| \log |r|)$ [2], which is unacceptable for most use cases. Due to the wide range of applications of functional dependencies several algorithms have been developed to solve this problem [3]. Such algorithms can be classified as follows:

A. Lattice traversal algorithms

Just like the brute force approach, these algorithms define the search space of the problem as the power set of all attributes in r. However, instead of exploring exhaustively this search space, these algorithms employ a lattice where each node corresponds to an element of the power set. To prune the search space, the lattice is traversed in a bottom-up fashion following only previously visited elements and candidate FDs generated from stripped partitions. TANE [4], FD_mine [5] [6] and DFD [7] are examples of lattice traversal algorithms.

B. Difference and agree sets algorithms

These algorithms start by calculating the set of attributes where a pair of tuples have the same value (agree sets), thus the search space for these algorithms is given by the Cartesian product of all tuples. Using this information it is possible to calculate the maximal sets for any attribute A. These sets are the largest possible sets of attributes that do not functionally determine A. Finally, using the complements of the maximal sets it is possible to derive all minimal functional dependencies on r. Examples of this class of algorithms are FUN and Dep-Miner [8].

It is worth noting that these algorithms are still exponential in the number of attributes in $\mathcal{U}$, despite numerous efforts no known algorithm solves the inference of exact functional dependencies efficiently [9]. Moreover lattice traversal algorithms are sensible to the number of attributes in $\mathcal{U}$, whereas difference and agree sets algorithms are sensible to the number of tuples in r [10]. In real world applications this means that lattice traversal algorithms can not handle data sets with a large number of attributes, and difference and agree sets algorithms can not handle data sets with a large number of tuples [3]. When using any of these algorithms in a real world application it is necessary to take into account the features of the data in order to select an appropriate algorithm.

III. Functional Dependencies via similarity functions

Exact functional dependencies are not always appropriate to determine if the attributes in a database are related in any meaningful way. According to section II for a FD to hold, it is necessary that an equivalence relation exists between two sets of attributes. However, in real world datasets such criteria is rarely met. As will be shown in Section VI sometimes even the nature of the data is responsible for such outcome. One way to overcome such limitation, that has been extensively studied, is to relax the requirement for equivalency using a similarity function.

A similarity function $K : X \times X \to \mathbb{R}^+$ associates every pair of elements in X, a real number that express the similarity between the elements of X (data objects), it is higher when two elements are more alike. In the same way it is possible to define the dual concept of dissimilarity (or distance) function as a function that express the difference between two objects. They are both called proximity measures. Similarity functions have the following properties:

- Non-negativity: $\forall x, y \in X, \sigma(x, y) \geq 0$
- Maximality: $\forall x, y \in X, \sigma(x, x) \geq \sigma(x, y)$
- Symmetry: $\forall x, y \in X, \sigma(x, y) = \sigma(y, x)$

where $\sigma(x, y)$ is the similarity between the data x and y.

They can be used to identify [11]:

- Duplicate data that may have differences due to typos.
- Equivalent instances from different data sets, e.g., names and/or addresses that are the same but have misspellings.
- Groups of data that are very close (clusters).

One of the earliest works that use similarity functions in relational databases appears in [12]. Here the authors analyze the problem of defining relations among the attributes of a dataset when missing data is found. A tuple missing a value for some attribute effectively prevents a FD from being found. To solve this problem, the authors define a relation between any two attributes in the dataset using similarity functions. In this work, a similarity function must be understood as a reflexive and symmetric map between any pair of elements belonging to a set of attributes, and the interval $[0, 1]$. The results show that structuring a dataset using proper similarity functions for each attribute allows to perform queries that determine if a property holds within a certain set of attributes.

Another alternative is given by approximate functional dependencies [13]. Here, an error measure that determines how far is a candidate FD from being valid is used. Therefore, a low error indicates a possible dependency between two sets of attributes. Commonly the error measure is defined using the number of tuples that prevent a FD from being valid. Approximate dependencies are used to verify the correctness and consistency of data during the process of data cleaning. The main advantage of approximate dependencies is that to calculate them, it is only necessary to modify an existing algorithm that infers FDs. The main disadvantage of approximate

dependencies is to determine how big the error measure can be before declaring a dependency between two sets of attributes. A solution to this issue is given in [14], where the authors use mutual information to define error estimators.

Matching dependencies [15] [16] are another extension to FDs that intend to overcome the limitations of equivalence relations. A matching dependency specifies a restriction that must be satisfied in order to establish a functional dependency between two sets of attributes. Consider a dataset with attributes X and Y. If X functionally determines Y, then this means that any pair of tuples in X have at least a similarity α, and that any pair of tuples in Y have a similarity of at least β. Here $\alpha, \beta \in [0, 1]$ are obtained using an appropriate similarity function, e.g., the Minkowski distance for numerical data, or the Levenshtein distance for text data.

Another example of dependencies based on similarity functions are metric functional dependencies [17]. Given two sets of attributes X and Y, if for every set of tuples having the same value in X, corresponds a set of tuples with similar values in Y then there is a metric functional dependency between X and Y. Mathematically we can define a metric functional dependency as follows: given a similarity function d fulfilling the triangle inequality, two sets of attributes X and Y define a metric functional dependency if the set of all tuples with value x in X, when projected into Y, lies within a ball of diameter δ. Note that in contrast to matching dependencies, metric functional dependencies only use similarity functions in the right hand side of the relation. This kind of relation depends on interpreting data as points in a highly dimensional metric space, so it is particularly fit for numerical and text data.

As we have seen exact functional dependencies do not always represent adequately the relationships of a dataset. As stated previously, functional dependencies are important for a great number of disciplines, so there exists great interest in designing algorithms to calculate these relations. It is important to remember that this is a hard problem, exponential in the number of attributes in a dataset.

Of course, non-exact functional dependencies require additional processing to calculate the similarity of any two tuples of a set of attributes. Thus, it is desirable to have algorithms that can handle the large number of data in most databases. In the present article we present an extension to a well known algorithm for the inference of exact functional dependencies that allows to calculate functional dependencies based on similarity functions. Our approach has the advantage that it is easy to implement, and also adapts well to scenarios where previously defined works do not apply.

IV. Data Cleaning

While data is being collected it is prone to errors either because it is manually collected or there is no standardized process for filling out the registers. Some of this sources of error may include misspellings, an erroneous entry, an entry into a wrong field, inconsistent timing, missing values and duplicates. Since the inference of functional dependencies depends on matching two or more tuples according to one or more attributes, it is necessary to guarantee the quality of the data before any processing can be done. In particular the data set must fulfill the following requirements [18]:

- Accuracy: The dataset agrees with real data.
- Timeliness: All data is up to date.
- Completeness: All values of an attribute have been recorded.
- Consistency: Data representation is uniform.
- Uniqueness: No duplicates.

We say that there is an anomaly if any of these properties does not hold. So we need a process that ensures that these anomalies are corrected, such process is called data cleaning.

We can classify these anomalies as:

- Syntactic, such as lexical and domain format errors (values that does not conform the expected format for an attribute) and certain irregularities.
- Semantic, that include integrity constraints, contradictions, duplicates an invalid tuples.
- Coverage, as they are missing values and missing tuples.

Due to the amount of data in large datasets, it is necessary to automate this process. The following are some tasks that should be included in any data cleaning process [19]:

- Determine the anomaly type.
- Find all anomalies in the dataset.
- Correct all found anomalies.
- Verify that all anomalies have been corrected.

V. The proposed methodology

The focus of the present work is a dataset henceforth called *crimes*, that contains crime occurrences in the municipality of Nezahualcóyotl, México between 2016 and 2017. The dataset is in a CSV file with $n = 15$ and $|r| = 16425$, where n is the number of attributes and $|r|$ is the number of tuples. The occurrence of a crime is described using information such as:

- Date and time. The date and time at which the crime occurred.
- Address. The name and number of the street where the crime occurred.
- Location. The (X, Y) coordinate where the crime occurred.
- Type. The type of crime that was committed.

In the following sections we detail the transformations and processing of this dataset that allows us to obtain the FDs in the data.

A. Data cleaning

Most of the data that comprises sociodemographic datasets comes from field studies that collect information about urban areas [20]. Since the data is collected manually is particularly prone to anomalies such as lexical errors, missing values, and domain format errors. These anomalies make the discovery of any meaningful relationship a very difficult task, to remove them we performed the following transformations on the data using the Pandas Python library.

TABLE I: Domain format errors in the dataset.

Tuple	DATE	TIME
2	1/1/2016	6:40
3519	14/05/16	17:00 A 20:15
15130	1-Jul	14:00

TABLE II: Domain errors corrected.

Tuple	DATE	START_TIME	END_TIME
2	1/1/2016	6:40	6:40
3519	14/05/16	17:00	20:15
15130	1-Jul	14:00	14:00

1) Missing errors. The first transformation applied to the data was to remove missing fields from the dataset, e.g., in some cases it was not possible to collect the street number where a crime was committed. In these cases a dummy value consisting of the string NO_ATTRIBUTE_NAME was used instead. ATTRIBUTE_NAME is the name of the corresponding attribute, e.g., STREET.
2) Lexical errors. As en example of this type of errors consider the attribute STREET_NUMBER which represents the particular address where a crime occurred. An example of an entry for this particular value is the string AV, PANTITLAN. Since our dataset is in CSV format no entry for an attribute may contain a comma character. A simple replacement of a comma with another suitable character, e.g, a period is enough to solve this anomaly.
3) Domain format errors. The prime example of this kind of anomalies occurs in the attributes TIME and DATE. Consider the tuples and their corresponding attributes shown in Table I. Here we can observe that there are multiple formats in use for the attribute DATE. It is not possible however, to simply parse each string to produce a uniform format. To solve this problem it was necessary to use a regular expression suitable for each of the formats, then extract each field of the date and finally produced a single format DAY/MONTH/YEAR for all tuples.

   The case of the TIME attribute, Table I shows that some tuple contain a single value representing the time at which the crime occurred. However, entry 3519 contains two time values that represent an estimate or time interval for the crime. For this scenario we decided to split the time attribute in two: a START_TIME representing the starting time of the crime, and an END_TIME attribute that represents the end time of the crime. This transformation has the advantage that it is possible to operate on the values of the attributes to obtain a time interval if necessary. Table II shows the result of this processing.

We also create a new attribute named COORDINATE made of attributes X and Y, separated by a space in order to use Minkowski distance.

TABLE III: Results of the experiments for the crimes dataset.

Dataset	n	$\|r\|$	N	Time (s)	Memory
crimes	15	16425	915	10.0896	322.0 MB
chess	7	28056	1	0.5353	60.8 MB
abalone	9	4177	137	0.19446	27.2 MB
nursery	9	12960	1	0.88912	90.8 MB

B. Inference of functional dependencies

As stated in Section II it is critical to consider the features of a dataset before selecting a particular algorithm for the inference of FDs. Comparative studies of the performance of several algorithms for this problem can be found in [3] and [10]. The results show that for all datasets tested, the TANE algorithm commonly outperforms other lattice traversal algorithms. For reference FD_mine fails for datasets with $n > 17$, and DFD fails for $n > 27$, where n is the number of attributes in the relation. Moreover, when compared to difference and agree sets algorithms, TANE is only outperformed significantly in some datasets (for $n > 30$, TANE exceeds a 100 GB memory limit). Taking this information into account we have selected TANE to calculate the FDs of the crimes dataset.

VI. Results

A. Benchmarks

In this section we report the results of using the TANE [21] algorithm to obtain the FDs of the crimes dataset. A C++ implementation of the TANE algorithm compiled with the GCC g++ compiler version 7.3 was used. Experiments were performed on a desktop PC running at 2.9 GHz with 8GB of RAM. To obtain an accurate estimate of the running time of the TANE algorithm for the crimes dataset, 100 runs of the algorithm were performed and their running time was averaged. The time of each individual execution was measured using the UNIX `time` command. Table III shows the results of these experiments, the attribute N represents the number of FDs found. For comparison similar experiments were performed using well known data sets taken from the UCI Machine Learning Repository [22].

As expected the running time for the crimes dataset is longer than the time observed in the other datasets. As stated in Section II, TANE is an exponential algorithm sensible to the number of attributes in a relation. However, it is important to note that factors such as the number of tuples and the number of FDs in the dataset also play an important role in determining the running time of the algorithm.

B. Analysis of FDs in the crimes dataset

The output of the algorithm is a set (in lexicographical order) of all valid, minimal and nontrivial FDs in the crimes dataset. An exhaustive analysis of the 917 dependencies is of course beyond the scope of this article. Instead consider the following sample of the results:

1) END_TIME, MONTH, X, Y $\rightarrow$ STREET
2) END_TIME, X, Y $\rightarrow$ CRIME_TYPE

3) START_TIME, MONTH, Y → CRIME_TYPE

Dependency 1 is a typical example of the dependencies found in the crimes dataset. It simply shows that as the number of attributes increases in the left hand side of the dependency it is more likely to uniquely determine the value of the attribute in the right hand side.

Dependencies 2 and 3 represent the type of relationship that might be interesting if conditions that lead to a crime are desirable. It would be too risky however to state that by looking at a time and (x, y) location it would be possible to determine the type of crime that would occur. Analysis of the data set reveals that the only reason this dependency exists is because each crime was committed at a different time and location, so it is obvious that a dependency would be found. We conclude that while these results might be used for database design, e.g., to normalize a database, they cannot be used to construct a predictive model of crime occurrences.

One of the reasons that leads to these results is that FDs are determined through an equivalence relation. For an attribute A to functionally determine attribute B it is necessary that every equivalence class in A to uniquely determine a value in B. If a single tuple in an equivalence class does not meet this criteria then we say that A does not functionally determine B. Nevertheless, meaningful relationships might exists in a dataset even if some tuples fail to meet the requirements of an exact FD.

C. Similarity functions

A possible solution to this problem is as follows, instead of requiring that two different tuples have the same value in order to belong to the same equivalence class, we determine membership trough a similarity function. The effect of this change is that close enough values of an attribute will belong to the same equivalence class. It is expected then that a functional dependency $X \rightarrow A$ holds even if two different tuples of X have slightly different values for attribute A.

To clarify this idea consider Table IV, which shows a small subset of the iris dataset. This dataset contains taxonomic data, in which the width and length of the petal or sepal are traits, that could determine each one of the three different species of iris plant. It is well known that the iris dataset has only four FDs $X \rightarrow A$, with $|X| = 3$, i.e., it is necessary to know at least three of this features in order to determine the species of the plant. It is worth noting that in Table IV the tuple $(4.8, 1.8)$ in rows 3 and 5 prevents a functional dependency $(\texttt{Petal length}, \texttt{Petal width}) \rightarrow \texttt{Class}$ from being valid. Since it would be unreasonable to expect that a measure such as the length or width of a petal be the same in every individual of the species, it is clear that a dependency must be defined taking into account such a small differences.

To determine a FD, the TANE algorithm uses the concept of partition. A partition is a collection of equivalence classes, i.e., sets of tuples that have the same value on some set of attributes X. If $t(A)$ denotes the value of attribute A for tuple t, then two tuples t_1 and t_2 are equivalent on attribute set X if $t_1(A) = t_2(A), \forall A \in X$. An equivalence class is a set of tuples that are all equivalent on attribute set X. The set of all equivalence classes for X is a partition denoted by π_X. From [4] it is known that a FD $X \rightarrow A$ is valid in some relation r if and only if $|\pi_X| = |\pi_{X \cup A}|$, where $|\pi_X|$ denotes the number of equivalence classes in partition π_X.

TABLE IV: A subset of the iris dataset.

Sepal length	Sepal width	Petal length	Petal width	Class
5.1	3.5	1.4	0.2	Setosa
4.7	3.2	1.3	0.2	Setosa
5.9	3.2	4.8	1.8	Versicolor
6.1	2.8	4.0	1.3	Versicolor
6	3	4.8	1.8	Virginica
6.7	3.1	5.6	2.4	Virginica

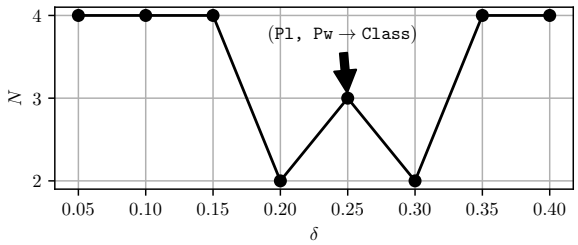

Fig. 1: The FD that relates the petal length (Pl) and petal width (Pw) holds for a value of $\delta = 0.25$.

So if we intend that TANE takes into account small differences in the value of a tuple for some attribute A, it is only necessary to determine membership of a tuple to some equivalence class according to a similarity function. Therefore, two tuples belong to the same equivalence class if $d(t_1(A), t_2(A)) \leq \delta$, where d is a distance function and δ is a threshold parameter.

According to these ideas, it is possible to construct equivalence classes for each attribute of a relation using appropriate similarity functions. For example, in the crimes dataset we could use a Levenshtein distance for attributes that contain textual information (STREET), a Minkowsky distance for numerical attributes (X and Y), the difference between two dates for the attribute DATE, or any other that fits the data.

We executed our implementation of the TANE algorithm with these modifications [23], using the Minkowski distance as a similarity function to construct the equivalence classes of the attribute "Petal Length". For this experiment, Figure 1 shows that for a value of $\delta = 0.25$ the FD $(\texttt{Petal length}, \texttt{Petal width}) \rightarrow \texttt{Class}$ holds in the relation. This small example illustrates that for datasets where attributes have domains that make difficult to infer exact FDs, our version of the TANE algorithm correctly infers relations in the data that are useful for machine learning or automatic knowledge discovery.

Due to the nature of the attributes in the crimes dataset, which has many-to-many relationships, it is very difficult to perform a task such as knowledge discovery. Consider for example, the attributes CRIME_TYPE and DATE that

represent the kind of crime that was committed and the date of the occurrence of the crime respectively. According to the definition of functional dependency, for these two attributes to be related it would be necessary that for every date recorded a single type of crime have been committed or vice versa, which is, of course, highly improbable. Nevertheless, in cases such as this, our methodology is still useful to find anomalies in the data that could have been omitted by the data cleaning process. For instance, the attribute `MONTH` can be deduced from the attribute `DATE`, i.e., `DATE` $\rightarrow$ `MONTH`. In the same way, it should be easy to deduce the `SECTOR` attribute from `QUADRANT` attribute since every quadrant has exactly one sector. Nonetheless, this FD is not discovered by the TANE algorithm for exact FDs, even when in the data cleaning step leading and trailing spaces as well as unfilled fields were corrected. However, when we used our modified algorithm together with a Levenshtein similarity function ($\delta = 1$), the dependency is found.

VII. Conclusions and future work

We have shown that using exact functional dependencies to analyze the crimes dataset does not yield any useful relation between the traits that define the occurrence of a crime and the kind of crime committed. Due to the spatial nature of this dataset we only found dependencies of such nature that could be deemed as trivial. However, by using similarity functions it was possible to find additional functional dependencies that while obvious, highlight problems in the data cleaning process and thus can be used to improve it.

It must be noted that in its present form our methodology is not adequate to perform knowledge discover on the crimes database. However, it might be possible to consider further modifications to our version of the TANE algorithm in order to calculate approximate functional dependencies based on similarity functions. By doing this we can discard a certain number of tuples that prevent a dependency from being valid, and also take into account data that is not clean enough, or data that contains tuples with values that are similar.

References

[1] J. Demetrovics, L. Libkin, and I. B. Muchnik, "Functional dependencies in relational databases: A lattice point of view," *Discrete Applied Mathematics*, vol. 40, no. 2, pp. 155 – 185, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X92900289

[2] H. Mannila and K.-J. Räihä, "On the complexity of inferring functional dependencies," *Discrete Applied Mathematics*, vol. 40, no. 2, pp. 237 – 243, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X92900315

[3] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann, "Functional dependency discovery: An experimental evaluation of seven algorithms," in *Proceedings of the VLDB Endowment*, vol. 8. Kohala Coast,Hawaii: VLDB Endowment Inc., 2015, pp. 1082–1093. [Online]. Available: http://www.vldb.org/pvldb/vol8/p1082-papenbrock.pdf

[4] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "Discovering functional and approximate dependencies," *The Computer Journal*, vol. 42, pp. 100–111, 1999.

[5] H. Yao and H. J. Hamilton, "Mining functional dependencies from data," *Data Mining and Knowledge Discovery*, vol. 16, no. 2, pp. 197–219, Apr. 2008. [Online]. Available: http://link.springer.com/10.1007/s10618-007-0083-9

[6] H. Yao, H. Hamilton, and C. Butz, "FD_mine: discovering functional dependencies in a database using equivalences," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* Maebashi City, Japan: IEEE Comput. Soc, 2002, pp. 729–732. [Online]. Available: http://ieeexplore.ieee.org/document/1184040/

[7] Z. Abedjan, P. Schulze, and F. Naumann, "DFD: Efficient Functional Dependency Discovery," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management - CIKM '14*. Shanghai, China: ACM Press, 2014, pp. 949–958. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2661829.2661884

[8] A. A. Chavan and V. K. Verma, "Mining Functional Dependency in Relational Databases using FUN and Dep-Miner: A Comparative Study," *International Journal of Computer Applications*, vol. 78, no. 15, pp. 34–36, Sep. 2013. [Online]. Available: https://research.ijcaonline.org/volume78/number15/pxc3891377.pdf

[9] T. Bläsius, T. Friedrich, and M. Schirneck, "The Parameterized Complexity of Dependency Detection in Relational Databases," in *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Guo and D. Hermelin, Eds., vol. 63. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 6:1–6:13. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/6920

[10] N. Asghar and A. Ghenai, "Automatic Discovery of Functional Dependencies and Conditional Functional Dependencies: A Comparative Study," Apr. 2015, university of Waterloo. [Online]. Available: https://cs.uwaterloo.ca/ nasghar/848.pdf

[11] L. Rhodes, "Similarity and Dissimilarity," Dec. 2018, juniata College. [Online]. Available: http://jcsites.juniata.edu/faculty/rhodes/ml/simdissim.htm

[12] B. P. Buckles and F. E. Petry, "A fuzzy representation of data for relational databases," *Fuzzy Sets and Systems*, vol. 7, no. 3, pp. 213–226, May 1982. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0165011482900525

[13] J. Kivinen and H. Mannila, "Approximate inference of functional dependencies from relations," *Theoretical Computer Science*, vol. 149, no. 1, pp. 129–149, Sep. 1995. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/030439759500028U

[14] P. Mandros, M. Boley, and J. Vreeken, "Discovering Reliable Approximate Functional Dependencies," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*. Halifax, NS, Canada: ACM Press, 2017, pp. 355–363. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3097983.3098062

[15] W. Fan, "Dependencies revisited for improving data quality," in *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '08*. Vancouver, Canada: ACM Press, 2008, p. 159. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1376916.1376940

[16] S. Song and L. Chen, "Efficient discovery of similarity constraints for matching dependencies," *Data & Knowledge Engineering*, vol. 87, pp. 146–166, Sep. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169023X13000700

[17] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian, "Metric Functional Dependencies," in *2009 IEEE 25th International Conference on Data Engineering*. Shanghai, China: IEEE, Mar. 2009, pp. 1275–1278. [Online]. Available: http://ieeexplore.ieee.org/document/4812519/

[18] L. Li, "Data quality and data cleaning in database applications," Ph.D. dissertation, Edinburgh Napier University, Edinburgh, Scotland, Sep. 2012.

[19] J. I. Maletic and A. Marcus, "Data Cleansing: A Prelude to Knowledge Discovery," in *Data Mining and Knowledge Discovery Handbook*, 2nd ed., O. Maimon and L. Rokach, Eds. Boston, MA: Springer US, 2009, pp. 19–32. [Online]. Available: http://link.springer.com/10.1007/978-0-387-09823-4_2

[20] C. Hernández-Alavez and R. Murcio-Villanueva, "Estudio de Indicadores ONU-HÁBITAT para los Observatorios Urbanos Locales de las Ciudades Mexicanas," SEDESOL, ONU-HÁBITAT México, Tech. Rep., 2004.

[21] Y. Zhou, "DependencyMiner," 2017. [Online]. Available: https://github.com/CalciferZh/DependencyMiner

[22] D. Dua and C. Graff, "UCI Machine Learning Repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[23] Z. M. Marquez-Navarrete, "DependencyMiner," 2019. [Online]. Available: https://github.com/TsukiZombina/DependencyMiner