Mining Relaxed Functional Dependencies from Data

Loredana Caruccio · Vincenzo Deufemia · Giuseppe Polese

Received: 07 February 2019

Abstract Relaxed functional dependencies (RFDs) are properties expressing important relationships among data. Thanks to the introduction of approximations in data comparison and/or validity, they can capture constraints useful for several purposes, such as the identification of data inconsistencies or patterns of semantically related data. Nevertheless, RFDs can provide benefits only if they can be automatically discovered from data. In this paper we present an RFD discovery algorithm relying on a lattice structured search space, previously used for FD discovery, new pruning strategies, and a new candidate RFD validation method. An experimental evaluation demonstrates the discovery performances of the proposed algorithm on real datasets, also providing a comparison with other algorithms.

Keywords Functional dependency · Discovery algorithm · Approximate match · Constraints Mining

1 Introduction

Functional dependencies (FDs) were originally used to verify database design and assess schema quality. In the last decades, they have been used for several new purposes, such as data profiling (Abedjan et al., 2015), query relaxation (Nambiar and Kambhampati, 2004), data cleansing (Bohannon et al., 2007), record matching (Fan et al., 2011), tensor decomposition (Kim and Candan, 2016), and so forth. To this end, their definition has been often extended in order to express constraints in these and other emerging application domains. For instance, FDs were extended for fuzzy (Raju and Majumdar, 1988), XML (Arenas and Libkin, 2004; Lee et al., 2002), multimedia (Chang et al., 2007), and temporal databases (Vianu, 1987).

L. Caruccio · V. Deufemia · G. Polese
University of Salerno, via Giovanni Paolo II n.132, 84084 Fisciano (SA)
E-mail: {lcaruccio, deufemia, gpolese}@unisa.it

Recent literature refers to extended FD definitions with the term *relaxed functional dependencies* (RFDs) (Caruccio et al., 2016b), and shows that they can be classified into three categories, depending on the constraints they relax with respect to the canonical definition of FD (Caruccio et al., 2016b). In particular, there exist RFDs relaxing on the data comparison, which compare tuples by using data similarity rather than equality, those relaxing on the *extent*, which admit the possibility for the RFD to hold only on a subset of data, and finally, hybrid ones, which relax on both criteria. Thresholds might be used in all such categories, either to specify the similarity degree or to indicate the minimum percentage of tuples on which the RFD should hold. With respect to the FD hierarchy proposed in (Szlichta et al., 2015), RFDs strictly generalize MDDs by allowing the definition of the extent relaxation criteria.

Although FDs are properties of the database schema to be specified at design time, in order to exploit them in practical domains several algorithms to discover them from data have been proposed (Abedjan et al., 2014; Berti-Équille et al., 2018; Lopes et al., 2000; Novelli and Cicchetti, 2001; Wyss et al., 2001; Yao et al., 2002). However, RFDs are much more complex to specify at design time, since they also require the specification of thresholds for evaluating the similarity between attribute values and/or for deriving the minimum extent. The automatic extraction of RFDs not only reduces the human effort to identify them at design time, but it also allows to easily adapt them to evolution of the application domain. Thus, the interest for the automatic extraction of RFDs from data has tremendously grown in the last years, especially from the machine learning and knowledge discovery research communities (Liu et al., 2012), also due to the diffusion of big data applications, which demand for improved data quality (Golab et al., 2008; Song et al., 2018; Yao and Hamilton, 2007). Nevertheless, few discovery algorithms have been proposed for RFDs, and they are specific of the RFD definition for which they were conceived.

In this paper, we propose DiMε, an algorithm for mining many different types of RFDs, relaxing on the tuple comparison and/or on the extent by using several tuple comparison methods and coverage measures (Caruccio et al., 2016b). The discovery technique underlying DiMε generates candidate RFDs based on Difference Matrices created from input thresholds, and on a lattice structure, also used within FD discovery algorithms (Abedjan et al., 2014; Huhtala et al., 1999; Novelli and Cicchetti, 2001; Yao et al., 2002), in order to model the search space. Moreover, DiMε provides new pruning strategies, in order to guarantee the generation of minimal candidate RFDs, and a new validation technique. With respect to the preliminary version presented in (Caruccio et al., 2016a), the current paper (*i*) provides a better formalization of the whole approach, (*ii*) an extended version of the algorithm to include the discovery of RFDs relaxing on the extent, hence also hybrid ones, and (*iii*) more an in-depth experimental evaluation, by comparing the DiMε performances with those of other algorithms on real datasets.

The paper is organized as follows. Section 2 reviews the RFD discovery algorithms existing in the literature. Section 3 provides some background defi-

nitions concerning RFDs. Section 4 formulates the RFD discovery problem. Section 5 presents the DIMε algorithm, whereas Section 6 shows the experimental results. Finally, summary and concluding remarks are included in Section 7.

2 Related Work

In this section, we review methods proposed in the literature for both FD and RFD discovery.

The automatic discovery of FDs is accomplished either through column-based or row-based methods (Abedjan et al., 2015). The former start generating candidate FDs based on an attribute lattice, verifying their validity, and then using holding FDs to prune the search space for candidate FDs yet to be verified; the latter compare attribute values for each pair of tuples, in order to generate two different sets of attribute subsets, namely *agree-sets* and *difference-sets*, from which candidate FDs are derived.

Column-based approaches include the algorithms TANE (Huhtala et al., 1999), FD_Mine (Yao et al., 2002), FUN (Novelli and Cicchetti, 2001), and DFD (Abedjan et al., 2014), whereas row-based approaches include DepMiner (Lopes et al., 2000), FastFD (Wyss et al., 2001), and FDep (Flach and Savnik, 1999). Empirical studies show that column-based algorithms outperform row-based ones on datasets with many rows and few columns, whereas row-based algorithms perform better on datasets with few rows and many columns (Papenbrock et al., 2015b). Moreover, DFD, TANE, and FDep are the ones best performing on a reasonable mix of rows (up to 300K) and columns (up to 25 columns for DFD, from 25 to 65 columns for TANE, and over 65 columns for FDep). Finally, the hybrid algorithm proposed in (Papenbrock and Naumann, 2016) combines row- and column-efficient discovery techniques by managing two separated phases, one calculating FDs on a small, randomly selected, subset of tuples (column-efficiency), and the other validating the discovered FDs on the entire dataset, trying to perform refinements when they do not hold.

With respect to these algorithms, DIMε discovers a broader class of dependencies beyond FDs, which entails much more a complex validation phase. In fact, although the size of the search space is the same, the validation of FDs merely requires to perform computations on the cardinalities of equivalence classes, whereas the validation of RFDs requires more complex computations on intersecting classes induced by similarity functions. For this reason, in spite of the many different RFD definitions proposed in the literature (Caruccio et al., 2016b), few algorithms have been provided for discovering them from data (Liu et al., 2012). In what follows, we review the RFDs for which a discovery algorithm has been provided.

Approximate functional dependency (AFD) are canonical FDs that must hold for 'almost' all the tuples of a relation r (Kivinen and Mannila, 1995). In other words, a limited number of the tuples of r can violate the AFD. Several approaches have been proposed to calculate the AFD extent (Giannella and Robertson, 2004; Sánchez et al., 2008), i.e., the satisfaction degree, among

which the *g3-error* measure is the most frequently used (Kivinen and Mannila, 1995). Most of the approaches for AFD discovery rely on sampling (Ilyas et al., 2004; Kivinen and Mannila, 1995). They verify whether a candidate AFD holds on an instance r by validating it on a sample of tuples $s \subset r$, which means that the AFD holding on s can also holds on r with a given probability. The method proposed in (King and Legendre, 2003) exploits the error measure of super keys to determine the extent of AFDs.

Conditional functional dependencies (CFDs) specify the extent of a dependency by means of a condition (Bohannon et al., 2007). Among the approaches proposed for discovering them from data, the algorithm proposed in (Chiang and Miller, 2008) exploits the attribute lattice built from the properties of attribute value partitions to derive candidate CFDs; the greedy algorithm proposed in (Golab et al., 2008) derives a close-to-optimal tableau for a CFD, given its underlying FD. The closeness of the discovered tableau to the optimal tableau is measured by means of the support and confidence measures. In addition, three of the above mentioned algorithms for FD discovery, namely FD_Miner, TANE, and FastFD, have been extended to discover CFDs, yielding the three algorithms CFD_Miner, CTANE, and FastCFD, respectively (Fan et al., 2009). With respect to CTANE and FastCFD, CFD_Miner has a limitation, since it cannot discover CFDs containing wildcards, hence it can only discover constant CFDs.

Matching dependencies (MDs) have been defined for object identification, and rely on similarity predicates to cope with errors, and heterogeneous representation formats in unreliable data sources (Fan et al., 2011). Main algorithms for discovering them evaluate the utility of MDs for the given database instance by means of the confidence and support measures, and determine the similarity threshold patterns by analyzing the statistical distribution of data (Song and Chen, 2013). The algorithm relies on pruning strategies to filter out candidate patterns with low support, aiming to achieve best performances.

Differential dependencies (DDs) relax FDs on the method to compare attribute values, by specifying constraints on their value differences (Song and Chen, 2011). A discovery algorithm for DDs relies on reduction algorithms, which seek the set of left-reduced LHS differential functions forming DDs with a fixed RHS differential function, for each attribute in a relation r (Song and Chen, 2011). The pruning strategies proposed to improve the discovery performances are based on the subsumption order of differential functions, the implication of DDs, and the instance exclusion. An alternative algorithm reduces the search space by means of user-specified upper limits of thresholds for the distance intervals of LHS (Kwashie et al., 2014). In particular, the discovery algorithm relies on a distance-based subspace clustering model, and includes further pruning strategies to enable efficient discovery of DDs for high threshold values. A further proposal for DD discovery is an algorithm mining a minimal cover of DDs, based on association rules (Kwashie et al., 2015). In particular, the approach exploits the algorithm presented in (Li, 2006) for mining a class of non-redundant association rules, which are successively transformed into DDs. The latter are pruned in order to generate a minimal cover of in-

teresting DDs. Finally, an algorithm has been proposed for deriving suitable thresholds for a given DD (Song et al., 2014). In particular, given a database instance and a DD holding on it, the algorithm determines the DD distance thresholds maximizing its utility, measured in terms of support, confidence, and dependent quality. The authors also prove that the identified distance thresholds are more effective than randomly selected settings in the context of violation detection.

While these discovery algorithms are each focused on a specific RFD, our goal has been to derive an algorithm for more a generic class of RFDs (Caruccio et al., 2016b), including RFDs relaxing on the tuple comparison method, those relaxing on the extent, and finally, hybrid ones relaxing on both.

Finally, it is worth to mention the following two frameworks: RQL and Metanome. The former provides a declarative pattern mining language to discover logical implications between attributes of the database (Chardin et al., 2017). Such implications generalize the concept of FD, enabling the use of data mining techniques in an SQL-like fashion. The Metanome project[1] provides an experimental framework for data profiling, including algorithms for the discovery of FDs and few RFD types (Papenbrock et al., 2015a). As DiMε, these frameworks go beyond FD discovery, since they are also capable to discover some types of RFDs. Nevertheless, although intersecting, the classes of RFDs they discover are different from those discovered with DiMε. In particular, except for conditional and order dependencies, DiMε is capable of discovering all RFDs relaxing on the extent, the tuple comparison method, or both, and it can be easily extended to deal with order dependencies. The RQL language allows to specify most types of RFDs, but with some limitations. For instance, it does not allow to specify general coverage measures for RFDs relaxing on the extent. Moreover, although it deals with conditional dependencies, it does not discover them from data, rather it retrieves them once the condition has been specified in input.

As for the Metanome framework, although it embeds many powerful algorithms for FD discovery, concerning RFDs it only includes an algorithm for discovering order dependencies and one for AFDs. However, as said above, the framework aims to discover a broader class of metadata other than functional dependencies.

3 Relaxed Functional Dependencies

In this section we will review the definition of FD, which will let us provide a general definition for RFDs.

FD *definition.* Consider a relational database schema $\mathcal{R}$ defined over a set of attributes $attr(\mathcal{R})$, derived as the union of the attributes from the relation schemas composing $\mathcal{R}$. Given an instance r of $\mathcal{R}$, an attribute $A \in attr(\mathcal{R})$,

[1] https://hpi.de/naumann/projects/data-profiling-and-analytics/metanome-data-profiling.html

and a tuple $t \in r$, we use $t[A]$ to denote the projection of t onto A; similarly, for a set X of attributes in $attr(\mathcal{R})$, $t[X]$ denotes the projection of t onto X. An FD over $\mathcal{R}$ is a statement $X \rightarrow Y$ (X implies Y), with $X, Y \subseteq attr(\mathcal{R})$, such that, given an instance r of $\mathcal{R}$, $X \rightarrow Y$ is satisfied in r if and only if for every pair of tuples (t_1, t_2) in r, whenever $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. X and Y represent the Left-Hand-Side (LHS) and Right-Hand-Side (RHS) of the FD, respectively.

The constraint defined by an FD can also be expressed by using the notion of tuple partition. In particular, given a set of attributes X, the *tuple partition* of an instance r on X, denoted by $\pi_X(r)$, is a collection of disjoint equivalence classes of tuples from r, each having a unique value combination for the attribute set X, and whose union corresponds to the relation r. The *equivalence class* of a tuple $t \in r$ with respect to X is denoted as $[t]_X$ and is defined as $[t]_X = \{u \in r \mid t[A] = u[A] \text{ for all } A \in X\}$. A partition π is said to *refine* another partition π' if every equivalence class in π is a subset of some equivalence class in π'. It is easy to show that an FD $X \rightarrow Y$ holds on r if and only if $\pi_X(r)$ refines $\pi_Y(r)$.

In the FD definition we notice that the projections of two tuples over a subset of attributes are compared by means of the equality function. This is one of the two dimensions that have been modified in order to define RFDs, by enabling the use of tuple comparisons based on *constraints*. A constraint is a predicate evaluating whether the distance or similarity between two values of an attribute falls within predefined intervals. More formally, let r be a relation instance of a relation schema R, $\mathbb{D}$ a set of distance or similarity functions defined on an attribute domain, and $\mathbb{B}$ a set of finite built-in comparison operators. A constraint ϕ is a logic expression of the form $\forall t_i, t_j \in r$ $(P_1 \bigwedge \ldots \bigwedge P_m)$, where P_k is a predicate $\delta(t_i[A], t_j[A])\theta_k c_k$, with $t_i[A]$ and $t_j[A]$ projections of tuples t_i and t_j on $A \in R$, $\delta \in \mathbb{D}$, $\theta_k \in \mathbb{B}$, and c_k a constant. Thus, a predicate of a constraint depends on a distance or similarity function defined on an attribute domain, plus one or more comparison operators with associated threshold values defining the feasible intervals of values. However, in order to simplify the explication of the algorithm, in the rest of the paper we will refer to constraints with one comparison operator and one threshold. As an example, the constraint $Jaro(addr1, addr2) \leq \varepsilon$ verifies that the Jaro distance (Elmagarmid et al., 2007) between the values $addr1$ and $addr2$ of the attribute Address is below the threshold value ε.

Another important characteristic of the FD definition is that it specifies a property of the database schema that must hold on every instance of it. This is the second dimension that has been modified to derive a general definition of RFD, which admits the possibility that the property might hold for a subset rather than all the tuples. The latter can be formally specified by means of a *coverage measure*.

Informally, an RFD is a functional dependency that relaxes on the tuple comparison, by using constraints on the distance or similarity between attribute values, and/or that relaxes on the extent, by using a coverage measure to indicate the minimum number or percentage of tuples on which the RFD

must hold, and/or by using conditions restricting the applicability domain of the RFD. Most of the RFDs defined in the literature relax on one of these two dimensions (Caruccio et al., 2016b). Those relaxing on both dimensions are called hybrid RFDs. In what follows, we recall a formal definition of RFD from our previous survey (Caruccio et al., 2016b), covering all these cases.

Definition 1 (Relaxed functional dependency) Consider a relational database schema $\mathcal{R}$, and a relation schema $R = (A_1, \ldots, A_m)$ of $\mathcal{R}$. An RFD φ on $\mathcal{R}$ is denoted by

$$X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2} \tag{1}$$

where

- $X = X_1, \ldots, X_h$ and $Y = Y_1, \ldots, Y_k$, with $X, Y \subseteq attr(R)$ and $X \cap Y = \varnothing$;
- $\Phi_1 = \bigwedge_{X_i \in X} \phi_i[X_i]$ ($\Phi_2 = \bigwedge_{Y_j \in Y} \phi_j[Y_j]$, resp.), where ϕ_i (ϕ_j, resp.) is a conjunction of predicates on X_i (Y_j, resp.) with $i = 1, \ldots, h$ ($j = 1, \ldots, k$, resp.). For any pair of tuples $(t_1, t_2) \in dom(R)$, the constraint Φ_1 (Φ_2, resp.) is true if $t_1[X_i]$ and $t_2[X_i]$ ($t_1[Y_j]$ and $t_2[Y_j]$, resp.) satisfy the constraint ϕ_i (ϕ_j, resp.) $\forall\ i \in [1, h]$ ($j \in [1, k]$, resp.).
- Ψ is a coverage measure defined on $dom(R)$, quantifying the amount of tuples violating or satisfying φ. It can be defined as a function $\Psi : dom(X) \times dom(Y) \to \mathbb{R}^+$, where $dom(X)$ is the cartesian product of the domains of attributes composing X. Among the most commonly used coverage measures there are the confidence, the *g3-error*, and the probability.
- ε is a threshold indicating the upper bound (or lower bound in case the comparison operator is $\geq$) for the result of the coverage measure.

Given $r \subseteq dom(R)$ a relation instance on R, r *satisfies* the RFD φ, denoted by $r \vDash \varphi$, if and only if: $\forall\ t_1, t_2 \in r$, if Φ_1 indicates true, then *almost always* Φ_2 indicates true. Here, *almost always* is expressed by the constraint $\Psi \leq \varepsilon$.

We can express the RFD constraint in a way similar to the FD definition. While the FD definition is based on the concept of tuple partition, i.e., disjoint equivalence classes, in RFD $\pi_X(r)$ ($\pi_Y(r)$, resp.) is composed of possibly intersecting subsets of tuples, each containing tuples that pairwise make Φ_1 (Φ_2, resp.) true, and whose union corresponds to the relation r. We say that $\pi_X(r)$ refines $\pi_Y(r)$ if every similarity subset of $\pi_X(r)$ is contained in some similarity subset of $\pi_Y(r)$. Thus, an RFD holds on the entire instance r if and only if $\pi_X(r)$ refines $\pi_Y(r)$. If the RFD holds almost always, we can express this property through a constraint on the coverage measure. Since the latter verifies how much $\pi_X(r)$ refines $\pi_Y(r)$, we can also see Ψ as a function of $\pi_X(r)$ and $\pi_Y(r)$.

For RFDs relaxing on the tuple comparison only, when no couple of tuples yields an RFD violation, the expression $\Psi(X, Y) = 0$ is omitted from the RFD expression. Thus, the canonical FD can also be written in terms of (1) as:

$$X_{\text{EQ}} \to Y_{\text{EQ}}$$

where EQ is the equality constraint.

As an example, in a database of scientific publications it is likely to have the same address and affiliation for authors with the same name. Thus, an FD {Author} → {Address, Affiliation} might hold. However, these attributes might have been stored using different abbreviations, hence the following RFD might hold:

$$\mathsf{Author}_{\phi_1} \rightarrow \{\mathsf{Address}_{\phi_2}, \mathsf{Affiliation}_{\phi_3}\}$$

where ϕ_1, ϕ_2, and ϕ_3 are constraints on a string similarity function. Moreover, since authors might change affiliation during their life, or there might be homonimies, the previous RFD should tolerate possible exceptions. This can be modeled by introducing a different coverage measure into the RFD:

$$\mathsf{Author}_{\phi_1} \xrightarrow{\psi(Author,Address,Affiliation) \leq 0.02} \{\mathsf{Address}_{\phi_2}, \mathsf{Affiliation}_{\phi_3}\}$$

A more general definition covering more types of RFDs has been provided in (Caruccio et al., 2016b).

4 Discovering RFDs

As seen in the previous section, the evaluation of an RFD $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2}$ entails (i) verifying constraints between the projections of pairs of tuples on attribute sets X and Y, and (ii) checking the amount of tuples for which the constraints hold on Y whenever they hold on X.

The discovery of RFDs is the problem of finding a set of *minimal* RFDs holding on a relation instance r. An RFD $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2}$ is *minimal* if there not exists a subset $Z \subset X$ such that $Z_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2}$ holds on r. With respect to the discovery of canonical FDs from data, this problem is significantly more complex, due to the extremely large search space induced by the evaluation of tuple similarity rather than equality.

The discovery of FDs entails finding a partition of tuples sharing the same values on RHS attributes whenever they share the same value on LHS attributes. One way to do this is by first generating FD candidates, and then verifying their validity and minimality. The validation process is usually accomplished by partitioning the tuples based on their values for the LHS and RHS attributes, and performing simple calculations on the cardinalities of partitions.

Whenever the constraints are specified for each attribute of the dataset, the discovery of RFDs reduces to finding both the subsets of tuples satisfying the constraints on the LHS attributes, and those satisfying the constraints on the RHS attributes (named LHS and RHS *similar pattern subsets*, respectively), hence verifying that each LHS similar pattern subset is contained in at least one RHS similar pattern subset, except for a number of cases below the threshold specified for the extent. Although this is similar to the discovery of FDs, the fact that in RFD discovery the similar pattern subsets are not disjoint considerably increases the computational complexity, as highlighted in Figure

Fig. 1: An example of tuple partition and of similar pattern subsets.

1. In fact, this prevents the possibility to adopt the validation methods used in FD discovery algorithms, yielding the necessity to conceive new validation methods for candidate RFDs. To this end, we propose a new validation method for RFD candidates based on similar pattern subsets of tuples, which checks the violations with respect to the user-specified threshold on the extent.

For sake of simplicity and w.l.o.g, in the rest of the article we assume that Y consists of a single attribute.

5 The DiMε Discovery Algorithm

The *Difference Matrix and ε-threshold* (DiMε) discovery algorithm presented in this section starts from user-specified thresholds, and it performs a level-by-level generation of candidate dependencies to be successively validated. Candidate RFD generation is accomplished by means of an attribute lattice built by considering all the possible attribute combinations. In particular, given a relation schema $R = \{A_1, \ldots A_n\}$, its attribute lattice will contain the empty set at Level 0, singleton sets, one for each attribute, at Level 1, pair sets, one for each possible combination of two attributes, at Level 2, and so forth. Finally, the last level, namely Level m, will contain a single set of all attributes from R.

Following the generation process of column-based methods highlighted in Figure 2, the proposed algorithm first generates attribute sets X at Level l, and then formulates all the possible RFDs $X\backslash\{A\} \rightarrow A$, with $A \in X$, to be successively validated. With respect to similar algorithms for FD discovery, the proposed one exploits their candidate RFD generation strategies, but it provides new pruning strategies to guarantee the generation of minimal candidate RFDs, and a new validation technique dealing with similarity subsets, and taking into consideration the possibility that an RFD might hold for a subset

of tuples. Dealing with tuple similarities rather than equality adds a considerable amount of complexity, since similarity subsets prevent the possibility to exploit properties of disjoint partitions for RFD validation purposes.

Since the *Candidate Generation* phase is the same of column-based FD discovery methods, in what follows we will focus on the remaining two phases of the proposed algorithm, namely *validation* and *pruning*. The Appendix contains the pseudo-code of the DIMε's algorithms for generating similar subsets and for validating candidate RFDs.

Fig. 2: The three main phases of column-based discovery methods (Liu et al., 2012).

5.1 Candidate RFD validation

Once candidate RFDs are generated by partitioning attribute sets from the lattice levels, their validation entails verifying that two tuples are similar on the RHS attribute, whenever they are similar on the LHS ones, and the property holds for a significant portion of the database. This is done by generating similarity subsets of tuples and using them to validate candidate RFDs.

5.1.1 Generating similarity subsets

As opposed to tuple partitions built in most FD discovery algorithms, similarity subsets of tuples might also intersect, as shown in Figure 1. In order to better explicate the generation process of similarity subsets, we will introduce the concept of *difference matrix*.

Definition 2 (Difference matrix of an attribute) Let r be an instance of a relation schema R, A an attribute of R, and δ a distance function defined on the domain of A. The *difference matrix* for A is a matrix M_A whose entry (i,j) contains the distance value $\delta(t_i[A], t_j[A])$ of the projections of tuples t_i and t_j on A.

Table 1 A sample dataset.

Tuple number	Height	Weight	Shoe size
1	175	70	40
2	175	75	39
3	175	69	40
4	176	71	40
5	178	81	41
6	169	73	37
7	170	62	39

	1	2	3	4	5	6	7
1	0	0	0	1	3	6	5
2	0	0	0	1	3	6	5
3	0	0	0	1	3	6	5
4	1	1	1	0	2	7	6
5	3	3	3	2	0	9	8
6	6	6	6	7	9	0	1
7	5	5	5	6	8	1	0

(a) Difference Matrix $M_{\mathsf{Height}_{\mathsf{abs}}}$

	1	2	3	4	5	6	7
1	0	5	1	1	11	3	8
2	5	0	6	4	6	2	13
3	1	6	0	2	12	4	7
4	1	4	2	0	10	2	9
5	11	6	12	10	0	8	19
6	3	2	4	2	8	0	11
7	8	13	7	9	19	11	0

(b) Difference Matrix of $M_{\mathsf{Weight}_{\mathsf{abs}}}$

	1	2	3	4	5	6	7
1	0	1	0	0	1	3	1
2	1	0	1	1	2	2	0
3	0	1	0	0	1	3	1
4	0	1	0	0	1	3	1
5	1	2	1	1	0	4	2
6	3	2	3	3	4	0	2
7	1	0	1	1	2	2	0

(c) Difference Matrix of $M_{\mathsf{Shoe\ Size}_{\mathsf{abs}}}$

	1	2	3	4	5	6	7
1	0,0	0,5	0,1	1,1	3,11	6,3	5,8
2	0,5	0,0	0,6	1,4	3,6	6,2	5,13
3	0,1	0,6	0,0	1,2	3,12	6,4	5,7
4	1,1	1,4	1,2	0,0	2,10	7,2	6,9
5	3,11	3,6	3,12	2,10	0,0	9,8	8,19
6	6,3	6,2	6,4	7,2	9,8	0,0	1,11
7	5,8	5,13	5,7	6,9	8,19	1,11	0,0

(d) Difference Matrix of $M_{\mathsf{Height}_{\mathsf{abs}},\mathsf{Weight}_{\mathsf{abs}}}$

Fig. 3: Difference Matrices for the dataset shown in Table 1.

Example 1 Let us consider the sample dataset in Table 1. Since it has all numerical attributes, we use the absolute difference, denoted with *abs*, as a distance function to construct the following three difference matrices: $M_{\mathsf{Height}_{\mathsf{abs}}}$, $M_{\mathsf{Weight}_{\mathsf{abs}}}$, and $M_{\mathsf{Shoe\ Size}_{\mathsf{abs}}}$, as shown in Figures 3(a), 3(b), and 3(c).

The definition of difference matrix can be easily generalized to attribute sets, in which an entry will contain an n-tuple of distance values, one for each attribute in the set.

Definition 3 (Difference matrix of an attribute set) Let r be an instance of a relation schema R, $X = \{A_1, \ldots, A_n\}$ an attribute set of R, and $\Delta = (\delta_1, \ldots, \delta_n)$ a sequence of distance functions defined on the domains of

$A_1, \ldots, A_n$, respectively. A *difference matrix* for X is a matrix M_{X_Δ} whose entry (i,j) contains the n-tuple $(d_1, \ldots, d_n)$, where $d_k = \delta_k(t_i[A_k], t_j[A_k])$, and t_i, t_j are tuples of r.

The difference matrix for an attribute set X can be derived from the difference matrices for the single attributes composing it, by concatenating elements with the same coordinates. As an example, for the dataset of Table 1, the difference matrix $M_{\mathsf{Height}_{\mathsf{abs}},\mathsf{Weight}_{\mathsf{abs}}}$ shown in Figure 3(d) is built from $M_{\mathsf{Height}_{\mathsf{abs}}}$ and $M_{\mathsf{Weight}_{\mathsf{abs}}}$ of Figures 3(a) and 3(b).

We exploit the notion of difference matrix to generate the *similar pattern* of a tuple t or a matrix row, which represents sets of tuples or matrix columns satisfying a given constraint wrt t. Successively, we group the tuples sharing the same similar patterns into *similar pattern subsets*.

Definition 4 (Similar pattern) Let r be an instance of a relation schema R, $X = \{A_1, \ldots, A_n\}$ an attribute set of R, M_{X_Δ} a difference matrix for X, and $\phi = (\phi_1, \ldots, \phi_n)$ a sequence of constraints on the values of M_{X_Δ}. A *similar pattern* of tuple t_i of M_{X_Δ}, denoted as $\tau_X^{t_i}$, is the sequence $(j_1, \ldots, j_h)$ with $h \leq |r|$ and $M[i, j_k] = (d_1, \ldots, d_n)$, where d_q satisfies the constraints ϕ_q $\forall q \in [1, n]$ and $\forall k \in [1, h]$.

Definition 5 (Similar pattern subsets) Let r be an instance of a relation schema R, $X = \{A_1, \ldots, A_n\}$ an attribute set of R, M_{X_Δ} a difference matrix for X, and $\phi = (\phi_1, \ldots, \phi_n)$ a sequence of constraints on the values of M_{X_Δ}. A *similar pattern subset* S_X for X is defined as $S_X = \{j_1, \ldots, j_h\}_{\{i_1,\ldots,i_k\}}$ with $1 \leq k \leq h \leq |r|$, $\tau_X^{i_p} = (j_1, \ldots, j_h)$ $\forall p \in [1, k]$ and $\tau_X^{i_v} \neq (j_1, \ldots, j_h)$ $\forall v \notin [1, k]$. The set of different *similar pattern subsets* for X is denoted as I_X.

Example 2 If in the dataset of Table 1 the user specifies constraints based on the *abs* function, the $\leq$ comparison operator, 1 as a threshold for both the attributes Height and Shoe Size, and 10 for the attribute Weight, then the following sets of similar pattern subsets are generated:

- $I_{\mathsf{Height}} = \{\{1,2,3,4\}_{\{1,2,3,4\}}, \{5\}_{\{5\}}, \{6,7\}_{\{6,7\}}\}$;
- $I_{\mathsf{Weight}} = \{\{1,2,3,4,6,7\}_{\{1,3\}}, \{1,2,3,4,5,6\}_{\{2,6\}}, \{1,2,3,4,5,6,7\}_{\{4\}}, \{2,4,5,6\}_{\{5\}}, \{1,3,4,7\}_{\{7\}}\}$;
- $I_{\mathsf{Shoe\ Size}} = \{\{1,2,3,4,5,7\}_{\{1,3,4\}}, \{1,2,3,4,7\}_{\{2,7\}}, \{1,3,4,5\}_{\{5\}}, \{6\}_{\{6\}}\}$.

These are highlighted in Figure 4 by using a different color for each subset.

In order to validate candidate RFDs, the proposed method reduces the number of similar pattern subsets for the attributes of the LHS, by eliminating singletons, since they correspond to the matrix diagonal entries that trivially compare each tuple with itself. This yields the definition of set of *stripped similar pattern subsets*.

Definition 6 (Set of stripped similar pattern subsets) Let r be an instance of a relation schema R, $X = \{A_1, \ldots, A_n\}$ an attribute set of R, and

	1	2	3	4	5	6	7
1	0	0	0	1	3	6	5
2	0	0	0	1	3	6	5
3	0	0	0	1	3	6	5
4	1	1	1	0	2	7	6
5	3	3	3	2	0	9	8
6	6	6	6	7	9	0	1
7	5	5	5	6	8	1	0

(a) Similar pattern subsets of Height

	1	2	3	4	5	6	7
1	0	5	1	1	11	3	8
2	5	0	6	4	6	2	13
3	1	6	0	2	12	4	7
4	1	4	2	0	10	2	9
5	11	6	12	10	0	8	19
6	3	2	4	2	8	0	11
7	8	13	7	9	19	11	0

(b) Similar pattern subsets of Weight

	1	2	3	4	5	6	7
1	0	1	0	0	1	3	1
2	1	0	1	1	2	2	0
3	0	1	0	0	1	3	1
4	0	1	0	0	1	3	1
5	1	2	1	1	0	4	2
6	3	2	3	3	4	0	2
7	1	0	1	1	2	2	0

(c) Similar pattern subsets of Shoe Size

Fig. 4: Colored difference matrices highlighting similar pattern subsets.

I_X a set of similar pattern subsets of X. The set of *stripped similar pattern subsets* $\widehat{I}_X$ of X is defined as

$$\{S_X \in I_X \mid S_X = \{j_1, \ldots, j_h\}_{\{i_1,\ldots,i_k\}} \text{ and } h > 1\}.$$

5.1.2 Validation of RFDs *relaxing only on the tuple comparison*

Stripped similar pattern subsets can be used to validate candidate RFDs based on the concept of *similar pattern subset refinement*.

Definition 7 A set $\widehat{I}_X$ is said to *refine* another set $\widehat{I}_{X\cup A}$ if every similar pattern subset in $\widehat{I}_X$ is contained in one of the subsets in $\widehat{I}_{X\cup A}$.

Formally, let $similar_X$ ($similar_Y$, resp.) be a similar pattern in $\widehat{I}_X$ ($\widehat{I}_Y$, resp.). If $\widehat{I}_X$ *refines* $\widehat{I}_Y$, then

$$similar_Y = \begin{cases} \{t_0, .., t, .., t_k\} \subseteq similar_X & \text{if } \exists\ \{t_0, .., t, .., t_k\}_{\{t_0,..,t,..,t_l\}} \in \widehat{I}_Y \\ \{t\} & \text{otherwise} \end{cases}$$

and is denoted $similar_Y \equiv similar_X$.

Based on the refinement notion, we derive the following lemma for RFDs.

	1	2	3	4	5	6	7
1	0,0	0,5	0,1	1,1	3,11	6,3	5,8
2	0,5	0,0	0,6	1,4	3,6	6,2	5,13
3	0,1	0,6	0,0	1,2	3,12	6,4	5,7
4	1,1	1,4	1,2	0,0	2,10	7,2	6,9
5	3,11	3,6	3,12	2,10	0,0	9,8	8,19
6	6,3	6,2	6,4	7,2	9,8	0,0	1,11
7	5,8	5,13	5,7	6,9	8,19	1,11	0,0

(a) Colored difference matrix for highlighting similar pattern subsets of Height and Weight

	1	2	3	4	5	6	7
1	0	1	0	0	1	3	1
2	1	0	1	1	2	2	0
3	0	1	0	0	1	3	1
4	0	1	0	0	1	3	1
5	1	2	1	1	0	4	2
6	3	2	3	3	4	0	2
7	1	0	1	1	2	2	0

(b) Colored difference matrix for highlighting similar pattern subsets of Shoe_Size, where the sub-matrix with thickest border highlights the inclusion property for RFD validation

Fig. 5: Colored difference matrices for validation.

Lemma 1 *A relaxed functional dependency* $X_{\Phi_1} \rightarrow A_{\Phi_2}$ *holds on an entire database instance if and only if* $\widehat{I}_X$ refines $\widehat{I}_A$.

Example 3 Let us consider the database instance of Table 1. If we set the constraints as defined in Example 2, then according to Lemma 1 the RFD

$$\{\mathsf{Height}_{\phi_1}, \mathsf{Weight}_{\phi_2}\} \rightarrow \mathsf{Shoe\ Size}_{\phi_3}$$

holds on the entire database. In fact, as shown in Figure 5, since $\widehat{I}_{\mathsf{Height,Weight}} = \{\{1,2,3,4\}_{\{1,2,3,4\}}\}$ and $\widehat{I}_{\mathsf{Shoe\_Size}} = \{\{1,2,3,4,5,7\}_{\{1,3,4\}}, \{1,2,3,4,7\}_{\{2,7\}}, \{1,3,4,5\}_{\{5\}}\}$, each similar pattern in $\widehat{I}_{\mathsf{Height,Weight}}$ is included in a similar pattern of $\widehat{I}_{\mathsf{Shoe\_Size}}$.

Alternatively, the refinement property between sets of stripped similar pattern subsets can be verified by calculating $||\widehat{I}_Z||$, which is defined as:

$$||\widehat{I}_Z|| = \frac{\sum_{t_i \in r}(|s_{t_i}| - 1)}{2} \forall s_{t_i} \in \widehat{I}_Z \tag{2}$$

This value represents the number of pairwise similar tuples in $\widehat{I}_Z$. Notice that, value 1 is subtracted to exclude the pairs consisting of same tuples, whereas the division by 2 has been necessary to consider each tuple pair only once. Since for an attribute set $W \subset Z$, $\widehat{I}_Z$ always refines $\widehat{I}_W$, it means that $\widehat{I}_{X \cup A}$ always refines $\widehat{I}_X$. However, we know that an RFD $X_{\Phi_1} \rightarrow A_{\Phi_2}$ holds only if $\widehat{I}_X$ is a refinement of $\widehat{I}_A$. Thus, since $\widehat{I}_{X \cup A}$ cannot have similar pattern subsets of size greater than those in $\widehat{I}_X$, the RFD $X \rightarrow A$ holds if $\widehat{I}_{X \cup A}$ and $\widehat{I}_X$ are equal, which yields the following lemma.

Lemma 2 *A relaxed functional dependency* $X_{\Phi_1} \rightarrow A_{\Phi_2}$ *holds on an entire database instance if and only if* $||\widehat{I}_{X \cup A}|| = ||\widehat{I}_X||$.

5.1.3 Validation of hybrid RFD*s*

In order to discover RFDs relaxing on the extent, DiMε should consider as valid the RFDs holding on a subset of tuples, according to a coverage measure and a user-defined threshold. In this case, it is not possible to use the previously defined lemma to validate candidate RFDs. In particular, to accomplish the validation process it is necessary to calculate the satisfiability degree of the instance according to a coverage measure.

In this paper, we show a version of DiMε relying on the *g*3-*error* coverage measure (Huhtala et al., 1999), which represents the minimum number of tuples to be removed from the database instance in order to make the RFD valid on the remaining ones. DiMε can also work with other coverage measures by overloading the function computing the measure from similar subsets.

The computation of the *g*3-*error* for RFDs with hybrid relaxation is an NP-complete problem (Song, 2010), since the MINIMUM VERTEX COVER problem can be reduced to it. In fact, if we consider a graph $G = (V, E)$, where each vertex in V corresponds to a tuple of a dataset, and each edge $(t_1, t_2) \in E$ corresponds to a tuple pair violating the satisfiability of an RFD φ, then the minimum vertex cover of G corresponds to the minimum number of tuples to be removed from the dataset to make φ valid, which allows us to compute the *g*3-*error*. However, a solution to this problem can be found in polynomial time in case of disjoint similar pattern subsets, in which case, we can use the same strategy of AFD discovery (Huhtala et al., 1999). The latter needs to be adapted to work with similar subsets instead of tuple partitions. In fact, in case of violations, each similar pattern subset in I_X containing violations will be partitioned into $n \geq 2$ subsets in $I_{X \cup A}$. Among these, the one of maximum size will contain the non-violating tuples. Thus, to compute the *g*3-*error* it will be sufficient to count the number of tuples falling in the remaining subsets, which can be performed in polynomial time. In particular, let $G_{X \to A} = \{similar_{X \cup A} \in I_{X \cup A} \mid similar_{X \cup A} \equiv similar_X$ with $similar_X \in I_X\}$, $|G_{X \to A}|_{S_X} = \{$size of the similar pattern subset $S_X \mid S_X \in I_X\}$, and $maxG_{X \to A} = max|G_{X \to A}|_{S_X}$. According to the refinement property, the *g*3-*error* can be derived by computing $maxG_{X \to A}$ for each $similar_X \in I_X$ and removing the sum of these values from the total number of tuples.

However, since in case of intersecting similar pattern subsets each of these tuples can fall in more than one similar pattern subset of I_X, it is not possible to perform a computation local to each similar pattern subset, but more a global analysis is required. Thus, we provide an approximate solution for calculating the *g*3-*error* in polynomial time in case of intersecting similar pattern subsets. It is based on a greedy strategy and can be easily generalized to calculate other coverage measures other than the *g*3-*error*. In particular, such solution is derived from an algorithm with factor-2 approximation for the MINIMUM VERTEX COVER problem (Johnson and Garey, 1979).

Example 4 Let us consider the database instance of Table 1. If we set the constraints as defined in Example 2, and the threshold 0.2 for the *g*3-*error*,

then the following RFD holds:

$$\text{Height}_{\phi_1} \xrightarrow{\Psi(Height,ShoeSize)\leq 0.2} \text{Shoe Size}_{\phi_2}$$

In fact, if we consider the similar pattern subsets

- $I_{\text{Height}} = \{\{1,2,3,4\}_{\{1,2,3,4\}}, \{5\}_{\{5\}}, \{6,7\}_{\{6,7\}}\}$
- $I_{\text{Height} \cup \text{Shoe Size}} = \{\{1,2,3,4\}_{\{1,2,3,4\}}, \{5\}_{\{5\}}, \{6\}_{\{6\}}, \{7\}_{\{7\}}\}$

for each set of I_{Height}, the maximum cardinality of the correspondent ones in $I_{\text{Height} \cup \text{Shoe Size}}$ are 4, 1, and 1, respectively. In particular, the first 1 is obtained from $\{5\}_{\{5\}}$ of I_{Height}, since each singleton set in I_{Height} corresponds to a singleton set in $I_{\text{Height} \cup \text{Shoe Size}}$, whereas the second 1 is obtained from $\{6,7\}_{\{6,7\}}$ of I_{Height}, since the equivalent maximum similar pattern subset in $I_{\text{Height} \cup \text{Shoe Size}}$ is a singleton. Consequently, we obtain the following error:

$$1 - \frac{(4+1+1)}{7} = 1 - \frac{6}{7} = \frac{1}{7} = 0.14$$

which satisfies the user-defined threshold of 0.2.

Limiting the complexity of the g3-error computation. In general, it is useful to limit the number of times the *g3-error* computation is performed. To this end, we can exploit the number of similar pairs $\|\widehat{I}_X\|$ and $\|\widehat{I}_{X \cup A}\|$, which can be derived from Equation (2). Such numbers can be used to compute the fraction of pairs violating an RFD:

$$v_{X \to A} = \frac{\|\widehat{I}_X\| - \|\widehat{I}_{X \cup A}\|}{\binom{|r|}{2}} \tag{3}$$

where r is a database instance, and the binomial coefficient represents the number of all possible tuple pairs that can be generated from it. If $v_{X \to A}$ exceeds a given threshold, then the candidate RFD $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$ cannot hold. However, to set such a threshold we cannot use the *g3-error* threshold, since this refers to the *fraction of tuples* that can violate $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$, whereas Equation (3) computes the *fraction of tuple pairs* violating the constraint associated to $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$. Thus, since we do not know how to map the fraction of violating tuples to the fraction of violating tuple pairs, we need to derive a threshold ε_c for this second fraction that best approximates the first one, so as to guarantee that for valid RFDs ε_c is not exceed. In particular, we can estimate ε_c from the *g3-error* threshold ε through the following equation:

$$\varepsilon_c = \frac{(\varepsilon \times |r|) \times max\left\{ |\tau_X^{t_i}| - 1 \;\middle|\; \tau_X^{t_i} \in \widehat{I}_X \right\}}{\binom{|r|}{2}} \tag{4}$$

where $|\tau_X^{t_i}|$ is the cardinality of the similar pattern of tuple t_i.

Theorem 1 *Let r be an instance of a relational schema R, and $\varphi : X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$ a candidate* RFD *on r. If $v_{X \to A} > \varepsilon_c$ then φ does not hold on r.*

Proof Let us suppose that $v_{X \to A} > \varepsilon_c$ but the RFD $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$ holds on r. It follows that

$$\|\widehat{I}_X\| - \|\widehat{I}_{X \cup A}\| > (\varepsilon \times |r|) \times \max\left\{ |\tau_X^{t_i}| - 1 \;\middle|\; \tau_X^{t_i} \in \widehat{I}_X \right\}.$$

Now, from (2) we have that

$$\|\widehat{I}_X\| - \|\widehat{I}_{X \cup A}\| = \frac{\sum_{i=1}^{|r|} (|\tau_X^{t_i}| - 1)}{2} - \frac{\sum_{j=1}^{|r|} (|\tau_{X \cup A}^{t_i}| - 1)}{2}.$$

Let $\widehat{I}_Z$ be the set of stripped similar pattern subsets such that $\tau_Z^{t_i} \in \widehat{I}_X$, $\tau_Z^{t_i} \notin \widehat{I}_{X \cup A}$, and $|\widehat{I}_Z| = m$, then

$$\|\widehat{I}_X\| - \|\widehat{I}_{X \cup A}\| = \sum_{i=1}^{m} |\tau_Z^{t_i}| - 1.$$

Hence,

$$\sum_{i=1}^{m} |\tau_Z^{t_i}| - 1 > (\varepsilon \times |r|) \times \max\left\{ |\tau_X^{t_i}| - 1 \;\middle|\; \tau_X^{t_i} \in \widehat{I}_X \right\},$$

where $(\varepsilon \times |r|)$ represents the number of tuples that could be removed according to ε (we can denote it by n); hence $(\varepsilon \times |r|) \times \max\left\{ |\tau_X^{t_i}| - 1 \;\middle|\; \tau_X^{t_i} \in \widehat{I}_X \right\}$ can be written as $\sum_{k=1}^{n} |\tau_X^{t_k}| - 1$ with $|\tau_X^{t_k}| - 1 = \max\left\{ |\tau_X^{t_i}| - 1 \;\middle|\; \tau_X^{t_i} \in \widehat{I}_X \right\}$ indicating that for each tuple t_k that can be removed we can remove a stripped similar pattern subset with maximum cardinality. As a consequence,

$$|\tau_Z^{t_k}| - 1 \leq |\tau_X^{t_k}| - 1, \forall t_k \in \widehat{I}_X \backslash \widehat{I}_{X \cup A}$$

hence, in the worst case, we can remove n tuples $t_1, \ldots, t_n$ whose similar pattern subsets have a cardinality equal to the maximum cardinality for $\tau_X^{t_i} \in \widehat{I}_X$. Since $\sum_{i=1}^{m} (|\tau_Z^{t_i}| - 1) - \sum_{k=1}^{n} (|\tau_Z^{t_k}| - 1) > 0$, then $(m - n) > 0$, that is, by removing n tuples according to ε there still remains at least a tuple $t_i \in \widehat{I}_Z$. This means that there exists a tuple t_i such that $\tau_Z^{t_i} \in \widehat{I}_X$ and $\tau_Z^{t_i} \notin \widehat{I}_{X \cup A}$, hence, the number of tuples violating the RFD exceeds the threshold ε, which means that φ does not hold on r.

Example 5 Let us consider the candidate RFD

$$\mathsf{Height}_{\phi_1} \xrightarrow{\Psi(Height, ShoeSize) \leq 0.2} \mathsf{Shoe\ Size}_{\phi_2}$$

defined for the database instance of Table 1 and the user-defined threshold $\varepsilon = 0.2$ for the *g3-error*. According to (4)

$$\varepsilon_c = \frac{(0.2 \times 7) \times 3}{\binom{7}{2}} = \frac{0.2 \times 21}{21} = 0.2$$

Thus, since the fraction of tuple pairs violating the RFD ($v_{\text{Height}\rightarrow\text{Shoe Size}}$) is less than the following bound:

$$v_{\text{Height}\rightarrow\text{Shoe Size}} = \frac{7-6}{\binom{7}{2}} = \frac{1}{21} = 0.1 \leq \varepsilon_c$$

then DIMε proceeds to compute the g3-*error*.

On the other hand, if we consider the candidate RFD

$$\text{Weight}_{\phi_1} \xrightarrow{\Psi(Weight,Height)\leq 0.2} \text{Height}_{\phi_2}$$

the corresponding similar pattern subsets are:

- $I_{\text{Weight}} = \{\{1,2,3,4,6,7\}_{\{1,3\}}, \{1,2,3,4,5,6\}_{\{2,6\}}, \{1,2,3,4,5,6,7\}_{\{4\}}, \{2,4,5,6\}_{\{5\}}, \{1,3,4,7\}_{\{7\}}\}$;
- $I_{\text{Weight} \cup \text{Height}} = \{\{1,2,3,4\}_{\{1,2,3,4\}}, \{5\}_{\{5\}}, \{6\}_{\{6\}}, \{7\}_{\{7\}}\}$.

Then:

$$\varepsilon_c = \frac{(0.2\times 7)\times 6}{\binom{7}{2}} = \frac{8,4}{21} = 0.4 < v_{\text{Weight}\rightarrow\text{Height}} = \frac{16-6}{\binom{7}{2}} = \frac{10}{21} = 0.48$$

hence the RFD does not hold.

5.2 Pruning strategies

DIMε adapts the pruning strategies of the TANE algorithm (Huhtala et al., 1999), namely *rhs candidate pruning* and *key pruning*, to deal with similarity pattern subsets. Rhs candidate pruning relies on the concepts of *rhs candidate set* and *rhs*$^+$ *candidate set*, named $C(X)$ and $C^+(X)$, respectively, which contain the attributes A such that the RFD $X_{\Phi_1} \xrightarrow{\Psi\leq\varepsilon} A_{\Phi_2}$ needs to be checked. In particular, $C(X) = attr(R)\backslash\{A \in X \mid X\backslash A \rightarrow A \text{ holds}\}$ and $C^+(X) = \{A \in attr(R) \mid \forall B \in X : X\backslash\{A,B\} \rightarrow B \text{ does not hold}\}$. In general, $C(X)$ guarantees the minimality of the discovered RFDs, whereas $C^+(X)$ optimizes the pruning process.

These sets are initialized with all the attributes of the relation, and then are modified during the discovery process according to the already discovered RFDs. In particular, when an RFD $X\backslash\{A\}_{\Phi_1} \xrightarrow{\Psi\leq\varepsilon} A_{\Phi_2}$ is discovered, the pruning strategies permit to reduce the search space by removing: (1) the attribute A from $C^+(X)$, and (2) all attributes B in $R\backslash X$ from $C^+(X)$.

Moreover, the *key pruning* removes all the keys in the database instance and their supersets, since keys are always LHSs of some dependencies, hence their supersets cannot produce minimal dependencies. To this end, in order to recognize keys during the discovery of RFDs, DIMε exploits the number of similar pairs $\|\widehat{I}_X\|$, as defined in (2). Indeed, when $\|\widehat{I}_X\| = 0$ there will not be similar pairs for the attribute set X, and the set of stripped similar pattern subsets $\widehat{I}_X$ will be empty, since I_X will be composed of singletons only. In other words, X contains values that are all *different*.

Example 6 Let us consider the database instance of Table 1. If the user specifies the constraints ≤ 1 for attributes Height and Shoe Size, and ≤ 0 for attribute Weight, then $I_{\mathsf{Weight}} = \{\{1\}_{\{1\}}, \{2\}_{\{2\}}, \{3\}_{\{3\}}, \{4\}_{\{4\}}, \{5\}_{\{5\}}, \{6\}_{\{6\}}, \{7\}_{\{8\}}\}$, $\widehat{I}_{\mathsf{Weight}} = \varnothing$, and the number of similar pairs $\|\widehat{I}_{\mathsf{Weight}}\| = 0$; hence, Weight is a key. When DiMε recognizes such a key, it outputs the following two RFDs:

$$\mathsf{Weight}_{\phi_1} \rightarrow \mathsf{Height}_{\phi_2}$$

$$\mathsf{Weight}_{\phi_1} \rightarrow \mathsf{Shoe\ Size}_{\phi_3}$$

hence, no superset of {Weight} will be subsequently checked.

5.3 Complexity analysis

The discovery of FDs is an extremely complex problem, since the number of potential FDs can be exponential, and their detection requires analyzing a huge number of attribute combinations (Abedjan et al., 2015). With respect to this problem, RFD discovery adds further complexity, due to the necessity to also analyze relaxation criteria. For instance, in the case of RFDs relaxing on the tuple comparison method, it is not possible to exploit the disjointness property of equivalence classes, because the use of approximate comparison methods does not guarantee the transitivity property of the equivalence relation, and might yield intersecting similar pattern sets. This considerably increases the computational complexity of the RFD discovery problem, since the similarity can only be computed by examining all possible pairs of tuples.

More specifically, discovering FDs over a relation instance r entails finding all the possible attribute combinations, and for each of them find all the possible partitions forming the LHS and the RHS of candidate FDs. Given this, the FD discovery problem has an extremely large search space. In fact, for a relation r with M attributes and n tuples, we need to consider all the possible combinations of 2 to M attributes, counting each of them as many times as the number of attributes in the combination, in order to account for the number of different candidate RHSs with a single attribute. This complexity is synthesized by the sum $\sum_{k=2}^{M} \binom{M}{k} k$, which is $O(2^M)$. However, this only represents the number of candidate FDs, for which further processing is needed, such as for their validation. When similarity thresholds are provided in input, finding RFDs relaxing on the tuple comparison method does not add complexity to the search space, but the validation process becomes quadratic in the number of tuples, while in FD discovery is linear. This is due to the fact that the transitivity property is not satisfied for value similarities. On the other hand, the relaxation on the extent entails the computation of coverage values according to the considered coverage measure. However, since a coverage measure typically refers to tuples and not tuple pairs, evaluating it on intersecting similar pattern subsets is an NP-COMPLETE problem. For this reason, we defined an approximate algorithm to be used when it is not possible to exploit validation bounds and the LHS similar pattern subsets are intersecting. Moreover, the

fact that an RFD can also hold on a subset of tuples prevents the possibility of exploiting several pruning strategies of FD discovery algorithms.

Concerning the complexity of DiMε, given a relation instance r with M attributes and n tuples, when thresholds are pre-specified for both the tuple comparison and the coverage, DiMε first constructs the sets of stripped similar pattern subsets for each attribute in $O(n^2M)$ (execution of Algorithms 1-2 M times), then it performs a level-by-level generation of candidate RFDs, in which stripped similar pattern subsets are composed for multiple attributes in $O(n^2)$ time for each candidate RFD, yielding a whole complexity for the generation of candidate RFDs of $O(2^M n^2)$, where $O(n^2)$ is the complexity to generate stripped similar pattern subsets for multiple attributes (Algorithm 3). Moreover, each candidate RFD is validated by computing the *g*3-*error* (Algorithm 5 or Algorithm 7), whose complexity is $O(n^2)$ in the worst case. Thus, the overall complexity of DiMε is $O(2^M n^2)$.

The Appendix contains the pseudo-code of the DiMε's algorithms mentioned above.

5.4 Correctness

The following theorem proves that all the RFDs discovered by DiMε are valid.

Theorem 2 *Each* RFD *discovered by* DiMε *is valid.*

Proof We proceed by contradiction. Given an attribute set $X = \{X_1, \ldots, X_n\}$, let us assume that DiMε discovers an RFD

$$\varphi : X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$$

that is not valid for the input instance r, with $\Phi_1 = \phi_{X_1}, \ldots, \phi_{X_n}$, and $\Phi_2 = \phi_A$. Then, according to the definition of RFD, there exist at least m tuples such that $\frac{m}{|r|} > \varepsilon$, for which the constraints $\phi_{X_1}, \ldots, \phi_{X_n}$ are satisfied, but not ϕ_A. This means that there are elements of similar pattern subsets from I_X that are not contained into the same similar pattern subset of $I_{X \cup A}$. Thus, in order to validate the RFD φ, DiMε considers either (*i*) Algorithm 5 (see Appendix) when stripped similar pattern subsets are disjoint, or (*ii*) Algorithm 7 (see Appendix) otherwise. In particular, if φ is validated through Algorithm 5 each similar pattern subset S_X of I_X can be split into more than one disjoint similar pattern subsets $(S_{X_1}, \ldots, S_{X_k})$ of $I_{X \cup A}$. Thus, for each S_X of I_X, Algorithm 5 computes the minimum number of tuples m_{S_X} to be removed from r in order for φ to hold on the remaining tuples. Let $m' = \sum_{S_X \in I_X} m_{S_X}$ be the total number of tuples to be removed from r, since DiMε discovered φ, then Algorithm 5 obtained $\frac{m'}{|r|} \leq \varepsilon$, but this contradicts the original assumption, because $m \leq m'$.

On the other hand, if φ is validated through Algorithm 7, the latter considers all violating tuple pairs that are similar on I_X but not on $I_{X \cup A}$. In

Table 2 Statistics on the datasets considered in the evaluation.

Datasets	# Columns	# Rows	# FD	Size [KB]
Iris	5	150	4	5
Balance-scale	5	625	1	7
Car-data	7	1,728	1	53
CiteSeer	7	10,000	10	8,610
Cars	9	406	67	23
Abalone	9	4,177	187	137
Breast-Cancer	11	699	46	20
Bridges	13	107	142	6
Lymphography	19	148	2,730	6

particular, Algorithm 7 iteratively selects the tuples to be removed from r until no more violating tuple pairs exist. For this reason, if Algorithm 7 validates φ, then we are sure that the number of selected tuples m' is a fraction $\frac{m'}{|r|} \leq \varepsilon$. As said before, this contradicts the original assumption, since $m \leq m'$.

This proof can be easily generalized to other coverage measures used in current RFD definitions, such as confidence or domain cardinality (Caruccio et al., 2016b).

6 Evaluation

In this section, we present the performances of DIMε in terms of discovered RFDs, also comparing its execution times with those of other approaches.

We implemented DIMε[2] in the Java language, by using the Levenshtein distance for comparing textual attributes (Levenshtein, 1966), the absolute difference for comparing numerical ones, and the $g3$-*error* as the coverage measure for extent evaluation (Kivinen and Mannila, 1995). The experiments have been performed on a machine with an Intel Xeon W 3.2GHz 8-core CPU, 64 GB RAM, running macOS High Sierra operating system, and with a 64-Bit Java environment.

The experiments were performed on several real-world datasets, previously used for evaluating FD and RFD discovery algorithms. In particular, we considered the Iris, Balance-scale, Car-data, Cars, Abalone, Breast-cancer, Bridges, and Lymphography datasets drawn from the UCI Machine Learning repository (Blake and Merz, 1998). For the comparative evaluation we chose a bigger dataset, namely the CiteSeer[3] dataset, containing data on scientific research papers. We focused on a subset of 10k of its tuples, projecting them on the subset of attributes (title,authors,year,description,id,subject, publisher). The characteristics of the considered datasets are reported in Table 2.

[2]The software and the datasets used in the evaluation are available online https://dastlab.github.io/dime/.

[3]http://csxstatic.ist.psu.edu/about/data

6.1 Discovery Results

In the first experiment we verified the effectiveness of DiMε in discovering RFDs relaxing on both the extent and the tuple comparison. In particular, we run several experiment sessions varying the similarity and the coverage measure thresholds simultaneously, in the range $[0, 4]$ with step 1, and in the range $[0, 0.4]$ with step 0.1, respectively. Only, for the Abalone dataset we considered different coverage measure thresholds, i.e. by varying them in the range $[0.0, 0.4]$.

Table 3 shows several examples of discovered RFDs for the Bridges dataset (which contains the characteristics of the Pittsburgh's bridges), according to different tuple comparison and extent thresholds, focusing on those with smaller LHS cardinality. Among the listed RFDs, we can notice that by admitting a $g3$-*error* of 20%, the Length of bridges implies their Type, whereas similar values of Erected and Type attributes always imply a similar value of Lanes. Another interesting example of discovered RFD is the hybrid RFD {Length,Type} → Location, indicating that whenever the Length and the Type of bridges are similar, then their Location is similar, within an error of 30% (extent threshold).

Figure 6 shows the number of RFDs extracted by DiMε from the UCI datasets according to several tuple comparison thresholds, whereby each line represents a different extent threshold. As expected, in the majority of cases, the relaxation on the extent increases the number of discovered RFDs. Moreover, we can observe that the highest number of RFDs is usually discovered with $g3$-*error* thresholds slightly above zero, i.e., 0.1 and 0.2. In fact, although intuitively a higher coverage threshold increases the number of potentially valid RFDs, the new ones might invalidate some previously discovered RFDs, since these could be no longer minimal.

Concerning the variation of tuple comparison thresholds, in most cases the number of discovered RFDs drastically drops when increasing them from 0 to 1. This is mainly due to the fact that with a zero threshold several tuples are possibly not similar to others on many LHS attribute candidates, yielding many key dependencies, which are likely to be invalidated when the tuple

Table 3 Several examples of discovered RFDs for the Bridges dataset.

Discovered RFDs	Tuple Comparison Thresholds	Extent Thresholds
Length→Type	0	0.2
Length→Lanes	1	0.1
Type→Lanes	1	0.1
Length→Purpose	1	0.3
Length,Type→Location	1	0.3
Erected,Type→Lanes	2	0.0
Length,Purpose→Material	2	0.2
Span→Location	3	0.4

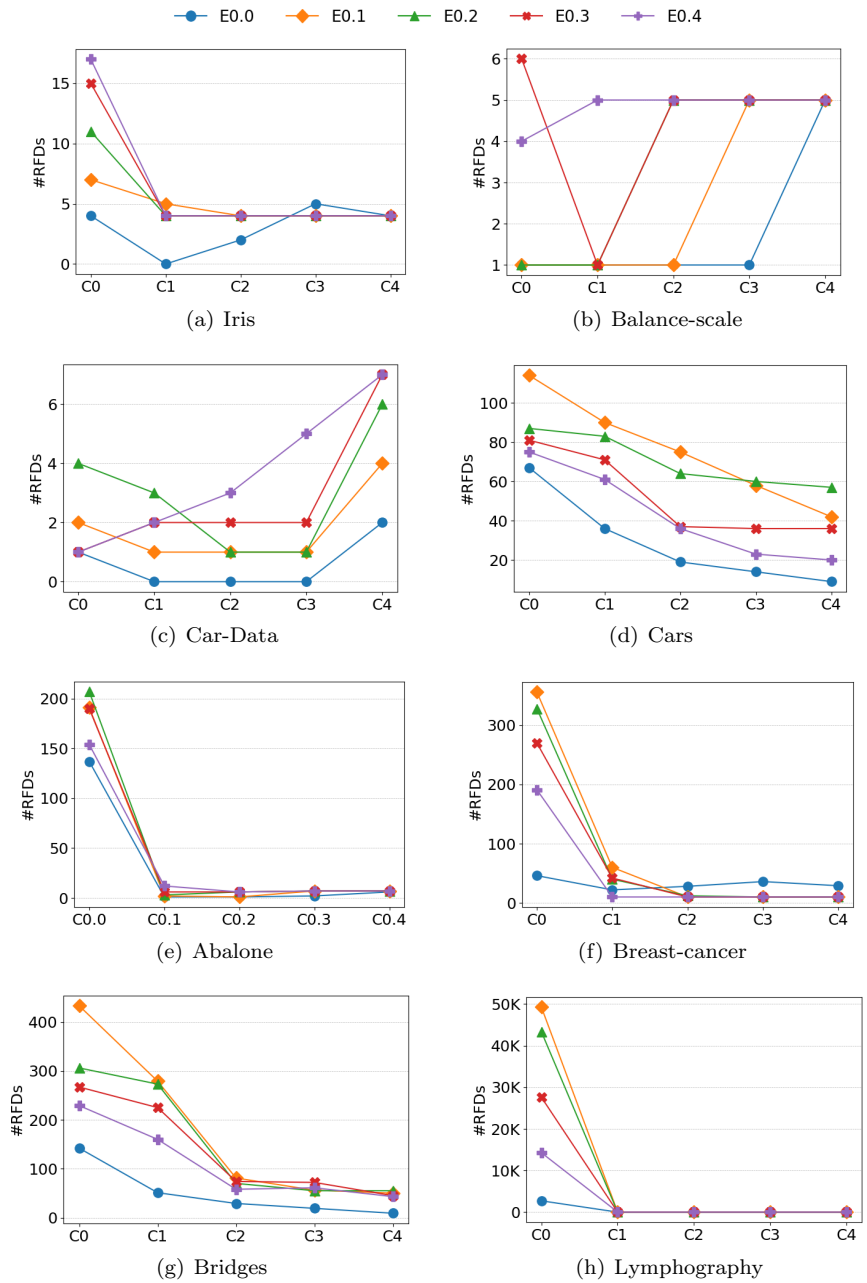


Fig. 6: Number of RFDs discovered by DiMε varying the tuple comparison and extent thresholds.

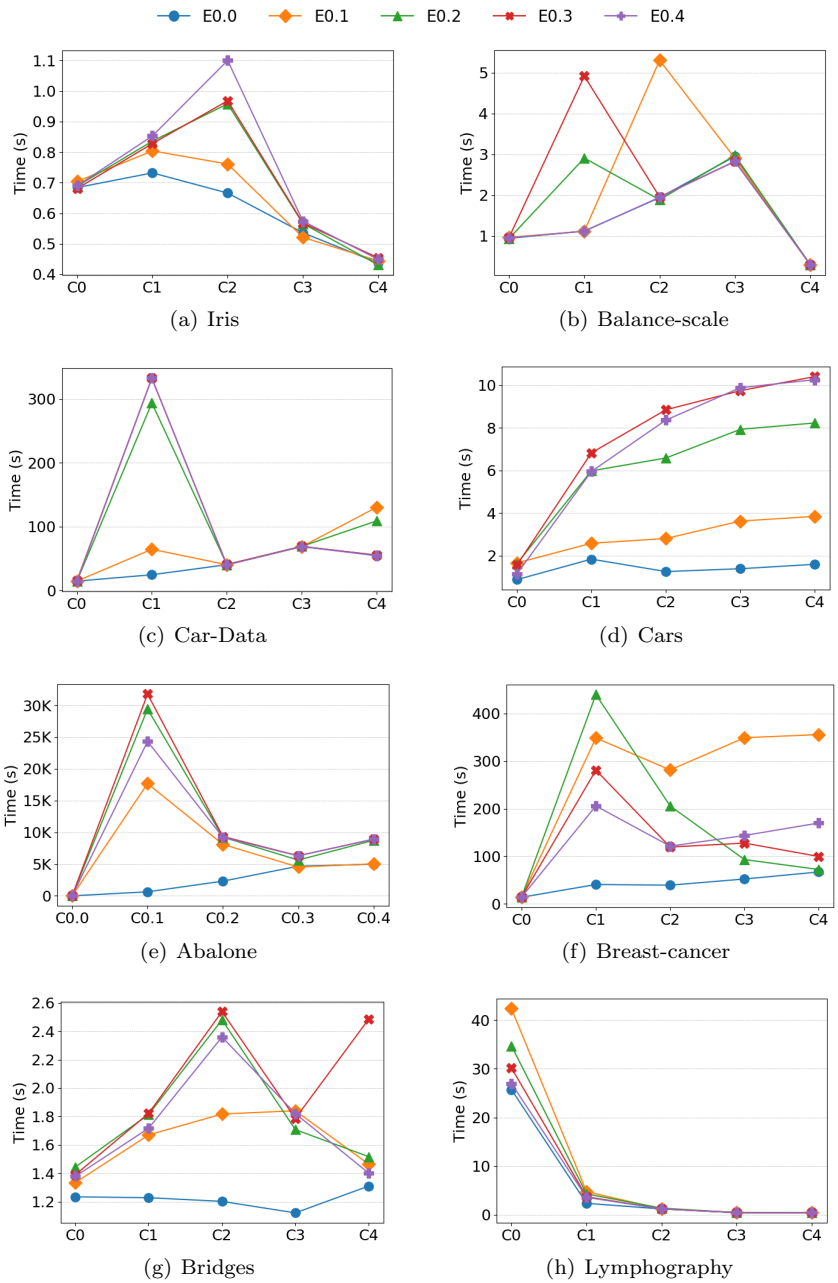
comparison threshold becomes greater than zero. Above the tuple comparison threshold 1 the behaviour is not easily predictable, since a higher threshold might both yield new discovered RFDs, because more RHS candidates might become similar, but also invalidate some existing RFDs, when it yields new tuple similarities on their LHS attributes, but not on their RHS attribute. Moreover, recall that a higher number of candidate RFDs might lead to a lower number of output RFDs, since it increases the possibility for some of them not to be minimal.

Figure 7 shows the execution times of DIMε for the UCI datasets. In general, although we expect higher execution times on datasets with more columns and/or rows, this is not necessarily true, because the attribute value distributions might heavily affect the computation steps to be performed, since they influence the number of tuple similarities, and hence the number of RFD candidates to be processed. As an example, for the Lymphography dataset (see Figure 7(h)) the execution times drastically drop when the tuple comparison threshold grows from 0 to 1. This is probably due to the fact that also the number of discovered RFDs decreases, and most importantly, their LHS cardinalities are low (see Figure 6(h)). We notice that worst time performances have been observed with the Abalone dataset (see Figure 7(e)), when a distance threshold 0.1 is chosen. This is due to the fact that such dataset contains an attribute that never appears as RHS, hence no pruning is performed on the sub-lattice rooted in it. Moreover, its value distribution yields many intersections among similar pattern subsets, preventing the application of the pruning strategies for the computation of the $g3$-*error* explained above.

6.2 Exact vs. approximate computation of the $g3$-error for hybrid RFD discovery

In what follows, we compare the results of DIMε with respect to a version of it using an exact algorithm for computing the $g3$-error. In particular, we considered the exact solution for the MINIMUM VERTEX COVER described in (Kleinberg and Tardos, 2006). The latter tries to find a minimum vertex cover with degree k by fixing a vertex at a time, and recursively searching a minimum vertex cover with degree $k-1$ from the graph obtained by removing such vertex and its incident edges, and so on. The complexity of this algorithm is $O(2^k kn)$, where n is the number of vertices. We integrated such solution into DIMε by computing a graph in which a vertex corresponds to a tuple involved in a violation, whereas an edge represents the violation between the tuples of the incident vertices.

We evaluated the cost of discovering RFDs on the Bridges dataset by considering two configurations in which we varied the number of tuples. The dataset instances considered the first n (with n = 30 and n = 60) tuples from the dataset, in the original order. Therefore, we ran the two versions of DIMε on the Bridges instances mentioned above, by setting a distance threshold in the



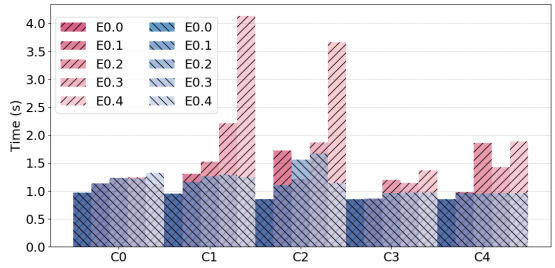
(a) Iris (b) Balance-scale

(c) Car-Data (d) Cars

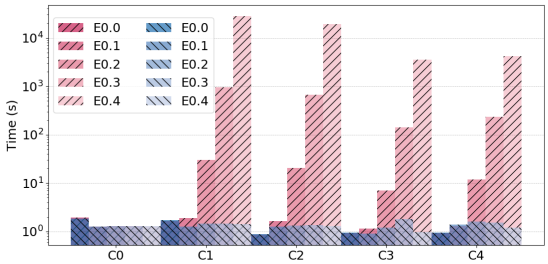
(e) Abalone (f) Breast-cancer

(g) Bridges (h) Lymphography

Fig. 7: Execution times for discovering RFDs by varying the similarity and the coverage measure thresholds.



(a) Bridges with 30 tuples



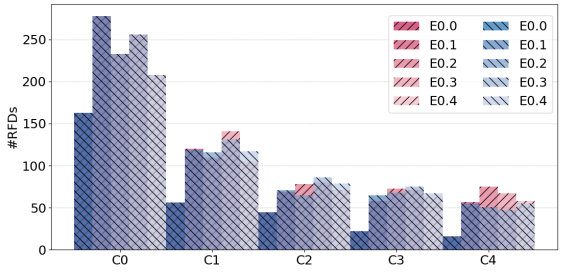
(b) Bridges with 60 tuples

Fig. 8: Comparison of execution times on the Bridges datasets.

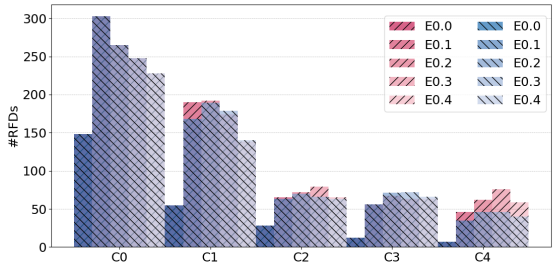
range $[0, 4]$ for all the attributes of the dataset, and a coverage threshold in the range $[0.0, 0.4]$.

Figure 8 shows the time performances of DIMε with the settings described above, in which the blue bars (with left-oriented diagonal lines) represent the execution times of the approximate solution, whereas the red bars (with right-oriented diagonal lines) represent the execution times of the exact one. Notice that for the smallest dataset the execution times are quite similar, but they grow exponentially by only doubling the number of tuples.

In this experiment we also compared the variability in the number of discovered RFDs of the two evaluated solutions. From Figure 9 we can notice that there are slight differences in the number of discovered RFDs. In particular, the red boxes (with right-oriented diagonal lines) on the top of the bars represent RFDs discovered only by the exact algorithm, whereas the light blue ones (with left-oriented diagonal lines) are those discovered only by the approximate algorithm. This is due to the fact the approximation misses the identification of few RFDs, but such missed RFDs could also reduce the effect of pruning with respect to the exact algorithm.



(a) Bridges with 30 tuples



(b) Bridges with 60 tuples

Fig. 9: Comparison of discovered RFDs on the Bridges datasets.

Table 4 Comparison between DiMε and TANE execution times (in seconds).

			DiMε		
Dataset	$\varepsilon = 0.0$	$\varepsilon = 0.1$	$\varepsilon = 0.2$	$\varepsilon = 0.3$	$\varepsilon = 0.4$
Bridges	1.23	1.33	1.44	1.39	1.38
Breast-Cancer	13.89	14.4	14.4	14.42	14.28
Lymphography	25.72	42.36	34.72	30.18	26.99
			TANE		
Dataset	$\varepsilon = 0.0$	$\varepsilon = 0.1$	$\varepsilon = 0.2$	$\varepsilon = 0.3$	$\varepsilon = 0.4$
Bridges	0.74	0.88	0.83	0.85	0.83
Breast-Cancer	0.87	1.11	1.17	1.15	1.86
Lymphography	9.54	15.68	16.1	13.25	11.78

6.3 Comparison with the TANE algorithm for discovering AFDs

In order to perform a comparative evaluation, we compared DiMε with the *approximate functional dependencies* (AFDs) discovery algorithm TANE (Huhtala et al., 1999), by using the datasets Bridges, Breast-Cancer, and Lymphography, and varying the coverage measure threshold in the range $[0, 0.4]$.

In Table 4, we have reproduced the time performances of DiMε with respect to TANE. As expected, TANE time performances are in general better than those of DiMε. In fact, as described in Section 4, the transitivity property of equivalence classes exploited by TANE reduces the general complexity of

the validation problem. Since DiMε aims to discover a broader class of RFDs, input datasets are formatted for generating similar subsets, waiving to the possibility of reducing the complexity based on the transitivity property. It is worth to notice that time performances shown in Table 4 reflect the fact that DiMε tackles more a complex problem.

6.4 Comparison with other RFD discovery algorithms

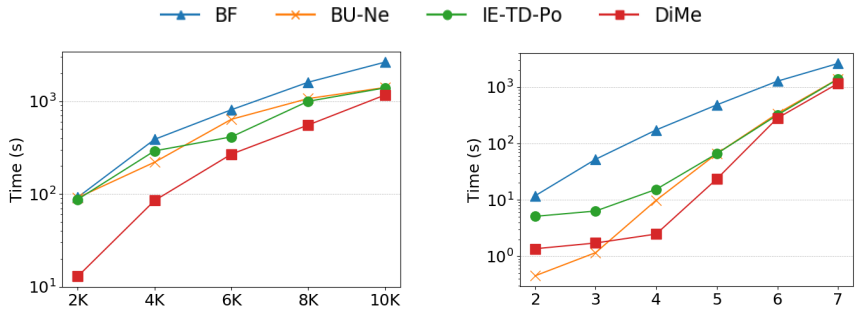
In what follows, we report the results of a comparison of DiMε with other RFD discovery algorithms. Since there are no algorithms discovering RFDs relaxing on both the tuple comparison and the extent, we implemented a brute force algorithm to compare with. Moreover, we compared DiMε with an algorithm discovering RFDs relaxing on the tuple comparison only, namely the DD discovery algorithm (Song and Chen, 2011).

We evaluated the cost of discovering RFDs on the CiteSeer dataset by varying the number of tuples in the range 2,000-10,000, and the number of attributes in the range 2-7. The dataset instances considered the first n ($n = 2000, \ldots, 10000$) tuples from the dataset, in the original order.

The comparison was accomplished by re-implementing several versions of the DD algorithm in Java SE 10, as described in (Song and Chen, 2011), and compared DiMε with the following three: (i) brute-force (BF), which validates all possible DDs according to a fixed search space; (ii) bottom-up with negative pruning (BU-Ne), which uses the subsumption order to reduce the search space; and (iii) instance-exclusion top-down with positive pruning (IE-TD-Po). While BU-Ne uses the subsumption order and the inference rules to reduce the search space, IE-TD-Po uses the subsumption order also to reduce the relation instance on which the algorithm performs the dependency validation. All the three versions split the discovery process into two phases: the first one aiming to validate DDs throughout the search space, which can be pruned according to the specific methodology; the second one aiming to test dependency minimality by removing all the discovered DDs that can be inferred by other ones.

We ran DiMε on the CiteSeer instances mentioned above, by setting a distance threshold equal to 2 for all the attributes of the dataset, and a coverage threshold equal to 0.0. In the same way, to execute the re-implementation of algorithms in (Song and Chen, 2011), we considered only one distance threshold, i.e., 2, for each attribute.

Figure 10 shows the time performances for discovering all RFDs with the settings described above. We can observe that DiMε scales well with the number of tuples. In particular, Figure 10(a) highlights that for all the selected datasets (by varying the number of tuples) DiMε outperforms the considered DD discovery algorithm versions. In particular, as we expected, the BF version achieves worst performances. Moreover, Figure 10(b) shows that by increasing the number of attributes, the time performance curve achieved by DiMε with CiteSeer is lower than the ones achieved by the DD algorithms. In particular,



(a) Varying the number of tuples (b) Varying the number of attributes

Fig. 10: Comparison of execution times on CiteSeer dataset.

only the BU-Ne version achieves better performances, but only on the two smallest datasets.

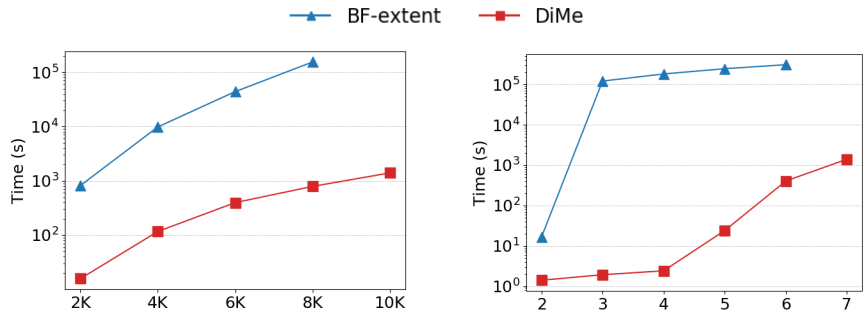
As mentioned above, the last experimental session concerned the comparison of DiMε execution times with respect to those of a brute force algorithm, named *BF-extent*, which is able to discover RFDs with hybrid relaxation. To this end, we extended the brute-force (BF) DD discovery algorithm in order to enable it to calculate the *g*3-*error* during the validation phase, comparing it with a fixed coverage measure threshold. More specifically, since the validation of hybrid RFDs is an NP-COMPLETE problem, we implemented the same greedy solution used for DiMε, which represents a factor-2 approximation algorithm for the MINIMUM VERTEX COVER problem (Johnson and Garey, 1979).

The comparison was accomplished by running both DiMε and BF-extent on the CiteSeer instances mentioned above, by setting a distance threshold equal to 2 for all the attributes of the dataset, and a coverage threshold equal to 0.2. The time performances achieved by the compared algorithms are shown in Figure 11. In particular, in Figure 11(a) we can observe that for all the considered dataset instances (obtained by varying the number of tuples) DiMε outperforms BF-extent. In fact, non only DiMε achieves better performances by several orders of magnitude, but BF-extent execution times increase faster as the number of tuples grows. This gap is even more evident by increasing the number of attributes, as shown in Figure 11(b).

7 Conclusion and future work

We have proposed DiMε, an algorithm for discovering RFDs from data. It exploits lattice-based strategies of FD discovery algorithms for generating RFD candidates, providing new pruning strategies and a new validation method to guarantee the generation of minimal RFDs.

The problem of discovering RFDs adds a considerable complexity to the dependency discovery process, since the relaxation criteria reduce the pos-



(a) Varying the number of tuples (b) Varying the number of attributes

Fig. 11: Comparison of execution times on CiteSeer dataset with hybrid relaxation.

sibilities of pruning search paths, and prevent the possibility to exploit the properties of disjoint partitions during the validation phase. The performed evaluation highlights the effectiveness of DiMε in the discovery of RFDs relaxing on both the tuple comparison and the extent dimensions.

In the future, we would like to further investigate the considered problem in order to derive an algorithm for RFD discovery capable of automatically inferring the threshold ranges of their validation, instead of requesting them to the user. Indeed, we can currently generalize our approach to solve this problem, but in exponential time. Thus, we would like to investigate further pruning techniques to make it work more efficiently. Finally, since the number of discovered RFDs is often huge, it is difficult immediately grasp insights from them. Thus, starting from what is proposed in (Caruccio et al., 2019), we would like to further investigate visual metaphors capable of representing numerous RFDs in a synthetic and easy way.

8 Appendix - The DiMε Algorithm

The DiMε discovery algorithm follows the process of column-based discovery algorithms, in which the generation of candidate RFDs for a given instance r of a relation schema R is accomplished through a level-by-level visit to a lattice structure, by considering all the possible k-combinations of attributes, and by increasing k at each level in the range $[2, n]$, where n is the number of attributes in R. Moreover, at each level only un-pruned candidates (see Section 5.2) are considered, and for each of them a validation process is performed. For this reason, DiMε can be considered as a column-based algorithm. However, since it aims to discover RFDs, it has to employ a new validation strategy, and new data configurations to highlight similarities between tuples.

Among the phases of the DiMε's discovery process described in Section 5, in what follows we provide the pseudo-code for those introducing some novelty with respect to other column-based discovery algorithms.

Generating similar pattern subsets. The procedure for generating the set of similar pattern subsets for an attribute A is shown in Algorithm 1. It takes as input a relation instance r, an attribute A, and a constraint ϕ_A, in order to construct and return I_A. In particular, for each tuple $t_i \in r$ (Line 4), it obtains the similar pattern $\tau_X^{t_i}$ by means of Algorithm 2 (Line 5), adding it to I_A, and associating t_i to it (Lines 6-7). Notice that it is not necessary to explicitly store difference matrices. Indeed, Algorithm 2 directly constructs stripped similar pattern subsets from differences on tuple pairs. It takes as input a relation instance r, an attribute A, a tuple t_i, and a constraint ϕ_A, in order to construct and return $\tau_X^{t_i}$. In particular, it analyzes each tuple $t_j \in r$ in order to evaluate the difference between values of tuple pairs (t_i, t_j) on attribute A (Line 4); it calculates such difference according to the constraint ϕ_A (Line 5), and it inserts t_j into $\tau_X^{t_i}$ if and only if the computed difference satisfies ϕ_A (Lines 6-8).

The procedure for generating the set of similar pattern subsets for an attribute set $X \cup A$ is shown in Algorithm 3. It generates $I_{X \cup A}$ from I_X and I_A. In particular, for each similar pattern of I_X, it obtains the tuples associated to it by means of the procedure GetTuples (Line 5). Furthermore, for each associated tuple t_i, it obtains the similar pattern $\tau_A^{t_i}$ from I_A, in order to construct the similar pattern subsets for $I_{X \cup A}$, by intersecting the two considered ones from I_X and I_A (Line 8), adding it to $I_{X \cup A}$ and associating t_i to it (Lines 9-10).

Validating a candidate RFD. The general DiMε validation phase is described in Algorithm 4. First of all, it computes $v_{X \to A}$ and ϵ_c, and if $v_{X \to A} > \epsilon_c$ it discards the candidate RFD (Lines 5-8). Otherwise, it verifies the disjointness property, and if it holds, it invokes Algorithm 5 for an exact computation of the $g3$-*error* in polynomial time, otherwise Algorithm 7 is invoked for an approximate computation of the $g3$-*error*.

In the following, we provide the details of the two algorithms for calculating the $g3$-error.

Algorithm 1 Generation of similar pattern subsets for an attribute A according to ϕ_A

```
1:  procedure GetSimSubsets(relation r, attribute A, constraint φ_A)
2:      Set I_A                                  ▷ The set of similar pattern subsets
3:      Set similar
4:      for each tuple t_i ∈ r do
5:          similar = SimilarPattern(r, A, t_i, φ_A)        ▷ (Algorithm 2)
6:          I_A = I_A ∪ {similar}
7:          Associate(similar, t_i)
8:      end for
9:      return I_A
10: end procedure
```

Algorithm 2 Generation of similar pattern w.r.t. a specified constraint

```
1: procedure SimilarPattern(relation r, attribute A, int t_i, constraint φ_A)
2:     int diff
3:     Set similar
4:     for each tuple t_j ∈ r do
5:         diff← δ_A(t_i[A],t_j[A])
6:         if Statisfy(diff, φ_A) then
7:             insert t_j into similar
8:         end if
9:     end for
10:    return similar
11: end procedure
```

Algorithm 3 Generation of similar pattern subsets for the attribute set $X \cup A$

```
1: procedure MultipleSimPatternSubsets(Set I_X, Set I_A)
2:     Set I_{X∪A}
3:     Set similar, similar_A, associatedTuples
4:     for each set similar_X of I_X do
5:         associatedTuples = GetTuples(similar_X, I_X)
6:         for each tuple t_i of associatedTuples do
7:             similar_A = GetSimSubsets(t_i, I_A)
8:             similar = similar_X ∩ similar_A
9:             I_{X∪A} = I_{X∪A}∪ {similar}
10:            Associate(similar, t_i)
11:        end for
12:    end for
13: end procedure
```

Computing the g3-error for disjoint similar pattern subsets. The procedure that DiMε follows to calculate the *g3-error* is shown in Algorithm 5. It takes as input the sets I_X and $I_{X \cup A}$, plus a relation instance r to calculate the fraction of tuples to be removed to make the candidate RFD $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$ valid. In particular, it analyzes each $similar_X \in I_X$ (Lines 5-9), and it obtains the maximum cardinality $max|G_{X \to A}|_{S_X}$ (Algorithm 6), according to the associated tuples of $similar_X$ (Lines 6-7). Then, the obtained sum is normalized with respect to the total number of tuples, and removed from 1, as defined by the *g3-error* definition (Line 10).

Algorithm 6 takes as input a set similar_X, its associated tuples, and a set $I_{X \cup A}$, in order to calculate the maximum cardinality $max|G_{X \to A}|_{S_X}$. In particular, for each $\text{similar}_{X \cup A}$, it identifies all $\text{similar}_{X \cup A} \equiv \text{similar}_X$ (Line 7), aiming to obtain the maximum cardinality among them (Lines 8-10).

Algorithm 4 Validation of a candidate RFD $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$

```
1: procedure CandidateValidation(relation instance r, Set I_X, Set I_{X∪A},
   double ε)
2:     double error ← 1
3:     Set Î_X = GetStripped(I_X)
4:     Set Î_{X∪A} = GetStripped(I_{X∪A})   ▷ stripped similar pattern subsets
5:     double v_{X→A} = violatingPairs(r, Î_X, Î_{X∪A})
6:     double ε_c = GetBound(r, ε, Î_X, Î_{X∪A})
7:     if v_{X→A} > ε_c then
8:         return false
9:     else
10:        if CheckDisjointness(Î_X) then
11:            error = GetError(I_X, I_{X∪A}, r)
12:        else
13:            error = GreedyGetError(I_X, I_{X∪A}, r)
14:        end if
15:        if error ≤ ε then
16:            return true
17:        end if
18:    end if
19:    return false
20: end procedure
```

Algorithm 5 Computation of the *g*3-*error* for a candidate RFD $X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} A_{\Phi_2}$

```
1: procedure GetError(Set I_X, Set I_{X∪A}, dataset r)
2:     Set assocTuples ← ∅
3:     double error ← 0
4:     int sum, maxCardinality
5:     for each set similar_X of I_X do
6:         assocTuples = GetAssociatedTuples(similar_X, I_X)
7:         maxCardinality = GetMaxCard(assocTuples, similar_X, I_{X∪A})
8:         sum ← sum + maxCardinality
9:     end for
10:    error ← 1 - (sum/|r|)
11:    return error
12: end procedure
```

Computing the g3-error for intersecting similar pattern subsets. In what follows, we show the greedy solution to calculate the *g*3-*error* for intersecting similar pattern subsets. It is derived from an algorithm with factor-2 approximation for the MINIMUM VERTEX COVER problem (Johnson and Garey, 1979).

To calculate the *g*3-*error* for a candidate RFD $X \rightarrow A$, Algorithm 7 takes as input the sets I_X and $I_{X \cup A}$, and a relation instance r. In particular, it selects

Algorithm 6 Computation of the maximum cardinality of similar pattern subsets equivalent to a given set

```
1: procedure GetMaxCard(Set assTuples_X, Set similar_X, Set I_{X∪A})
2:     Set assTuples_{X∪A}
3:     int maxCardinality
4:     maxCardinality ← 0
5:     for each set similar_{X∪A} of I_{X∪A} do
6:         assTuples_{X∪A} = GetAssociatedTuples(similar_{X∪A}, I_{X∪A})
7:         if similar_X ⊇ similar_{X∪A} and assTuples_X ⊇ assTuples_{X∪A} then
8:             if |assTuples_{X∪A}| > maxCardinality then
9:                 maxCardinality ← |assTuples_{X∪A}|
10:            end if
11:        end if
12:    end for
13:    return maxCardinality
14: end procedure
```

Algorithm 7 Greedy computation of the *g3-error* for a candidate RFD $X \rightarrow A$

```
1: procedure GreedyGetError(Set I_X, Set I_{X∪A}, relation r)
2:     Set errorSet ← ∅
3:     Set violatingPairs = getViolatingTuplePairs(I_X, I_{X∪A})
4:     while violatingPairs <> ∅ do
5:         extract a tuple pair p = (t_1, t_2) ∈ violatingPairs
6:         errorSet = errorSet ∪ p
7:         violatingPairs = removeOtherPairs(violatingPairs, t_1, t_2)
8:     end while
9:     return |errorSet|/|r|
10: end procedure
```

all tuple pairs that are similar on I_X but not on $I_{X \cup A}$ (Line 3), whereas in Lines 4-8 it iteratively selects a violating tuple pair $p = (t_1, t_2)$ in order to remove all violating pairs involving t_1 or t_2. Finally, the algorithm normalizes the number of violating tuple pairs selected on Line 5 (errorSet) on the size of the relation instance r (Line 9).

References

Abedjan Z, Schulze P, Naumann F (2014) DFD: Efficient functional dependency discovery. In: Proceedings of the 23rd ACM International Conference on Information and Knowledge Management, CIKM '14, pp 949–958, URL https://doi.org/10.1145/2661829.2661884

Abedjan Z, Golab L, Naumann F (2015) Profiling relational data: A survey. The VLDB Journal 24(4):557–581, URL https://doi.org/10.1007/s00778-015-0389-y

Arenas M, Libkin L (2004) A normal form for XML documents. ACM Transactions on Database Systems 29(1):195–232, URL https://doi.org/10.1145/974750.974757

Berti-Équille L, Harmouch H, Naumann F, Novelli N, Thirumuruganathan S (2018) Discovery of genuine functional dependencies from relational data with missing values. Proceedings of the VLDB Endowment 11(8):880–892, URL https://doi.org/10.14778/3204028.3204032

Blake CL, Merz CJ (1998) UCI repository of machine learning databases. URL https://archive.ics.uci.edu/ml/index.php

Bohannon P, Fan W, Geerts F, Jia X, Kementsietsidis A (2007) Conditional functional dependencies for data cleaning. In: Proceedings of the 25th International Conference on Data Engineering, ICDE '07, pp 746–755, URL https://doi.org/10.1109/ICDE.2007.367920

Caruccio L, Deufemia V, Polese G (2016a) On the discovery of relaxed functional dependencies. In: Proceedings of the 20th International Database Engineering & Applications Symposium, IDEAS '16, pp 53–61, URL https://doi.org/10.1145/2938503.2938519

Caruccio L, Deufemia V, Polese G (2016b) Relaxed functional dependencies – A survey of approaches. IEEE Transactions on Knowledge and Data Engineering 28(1):147–165, URL https://doi.org/10.1109/TKDE.2015.2472010

Caruccio L, Deufemia V, Polese G (2019) Visualization of (multimedia) dependencies from big data. Multimedia Tools and Applications 78(23):33,151–33,167, URL https://doi.org/10.1007/s11042-019-07951-0

Chang SK, Deufemia V, Polese G, Vacca M (2007) A normalization framework for multimedia databases. IEEE Transactions on Knowledge and Data Engineering 19(12):1666–1679, URL https://doi.org/10.1109/TKDE.2007.190651

Chardin B, Coquery E, Pailloux M, Petit JM (2017) RQL: A query language for rule discovery in databases. Theoretical Computer Science 658:357–374, URL https://doi.org/10.1016/j.tcs.2016.11.004

Chiang F, Miller RJ (2008) Discovering data quality rules. Proceedings of the VLDB Endowment 1(1):1166–1177, URL https://doi.org/10.14778/1453856.1453980

Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19(1):1–16, URL https://doi.org/10.1109/TKDE.2007.250581

Fan W, Geerts F, Lakshmanan LVS, Xiong M (2009) Discovering conditional functional dependencies. In: Proceedings of the 25th International Conference on Data Engineering, ICDE '09, pp 1231–1234, URL https://doi.org/10.1109/ICDE.2009.208

Fan W, Gao H, Jia X, Li J, Ma S (2011) Dynamic constraints for record matching. The VLDB Journal 20(4):495–520, URL https://doi.org/

10.1007/s00778-010-0206-6
Flach PA, Savnik I (1999) Database dependency discovery: A machine learning approach. AI communications 12(3):139–160
Giannella C, Robertson E (2004) On approximation measures for functional dependencies. Information Systems 29(6):483–507, URL https://doi.org/10.1016/j.is.2003.10.006
Golab L, Karloff H, Korn F, Srivastava D, Yu B (2008) On generating near-optimal tableaux for conditional functional dependencies. Proceedings of the VLDB Endowment 1(1):376–390, URL https://doi.org/10.14778/1453856.1453900
Huhtala Y, Kärkkäinen J, Porkka P, Toivonen H (1999) TANE: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal 42(2):100–111, URL https://doi.org/10.1093/comjnl/42.2.100
Ilyas IF, Markl V, Haas P, Brown P, Aboulnaga A (2004) CORDS: Automatic discovery of correlations and soft functional dependencies. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, SIGMOD '04, pp 647–658, URL https://doi.org/10.1145/1007568.1007641
Johnson DS, Garey MR (1979) Computers and intractability: A guide to the theory of NP-completeness. WH Freeman
Kim M, Candan KS (2016) Decomposition-by-normalization (DBN): Leveraging approximate functional dependencies for efficient CP and tucker decompositions. Data Mining and Knowledge Discovery 30(1):1–46, URL https://doi.org/10.1007/s10618-015-0401-6
King RS, Legendre JJ (2003) Discovery of functional and approximate functional dependencies in relational databases. Advances in Decision Sciences 7(1):49–59, URL https://doi.org/10.1155/S117391260300004X
Kivinen J, Mannila H (1995) Approximate inference of functional dependencies from relations. Theoretical Computer Science 149(1):129–149, URL https://doi.org/10.1016/0304-3975(95)00028-U
Kleinberg J, Tardos E (2006) Algorithm design. Pearson Education India
Kwashie S, Liu J, Li J, Ye F (2014) Mining differential dependencies: A subspace clustering approach. In: Proceedings of the 25th Australasian Database Conference, ADC '14, pp 50–61, URL https://doi.org/10.1007/978-3-319-08608-8_5
Kwashie S, Liu J, Li J, Ye F (2015) Efficient discovery of differential dependencies through association rules mining. In: Proceedings of the 26th Australasian Database Conference, ADC '15, pp 3–15, URL https://doi.org/10.1007/978-3-319-19548-3_1
Lee ML, Ling TW, Low WL (2002) Designing functional dependencies for XML. In: Proceedings of the 8th International Conference on Extending Database Technology, EDBT '02, pp 124–141, URL https://doi.org/10.1007/3-540-45876-X_10
Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10(8):707–710

Li J (2006) On optimal rule discovery. IEEE Transactions on Knowledge and Data Engineering 18(4):460–471, URL https://doi.org/10.1109/TKDE.2006.1599385

Liu J, Li J, Liu C, Chen Y (2012) Discover dependencies from data - A review. IEEE Transactions on Knowledge and Data Engineering 24(2):251–264, URL https://doi.org/10.1109/TKDE.2010.197

Lopes S, Petit JM, Lakhal L (2000) Efficient discovery of functional dependencies and Armstrong relations. In: Proceedings of the 7th International Conference on Extending Database Technology, EDBT '00, pp 350–364, URL https://doi.org/10.1007/3-540-46439-5_24

Nambiar U, Kambhampati S (2004) Mining approximate functional dependencies and concept similarities to answer imprecise queries. In: Proceedings of the 7th International Workshop on the Web and Databases, WebDB '04, pp 73–78, URL https://doi.org/10.1145/1017074.1017093

Novelli N, Cicchetti R (2001) FUN: An efficient algorithm for mining functional and embedded dependencies. In: Proceedings of the 8th International Conference Database Theory, ICDT '01, pp 189–203, URL https://doi.org/10.1007/3-540-44503-X_13

Papenbrock T, Naumann F (2016) A hybrid approach to functional dependency discovery. In: Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data, SIGMOD '16, pp 821–833, URL https://doi.org/10.1145/2882903.2915203

Papenbrock T, Bergmann T, Finke M, Zwiener J, Naumann F (2015a) Data profiling with Metanome. Proceedings of the VLDB Endowment 8(12):1860–1863, URL https://doi.org/10.14778/2824032.2824086

Papenbrock T, Ehrlich J, Marten J, Neubert T, Rudolph JP, Schönberg M, Zwiener J, Naumann F (2015b) Functional dependency discovery: An experimental evaluation of seven algorithms. Proceedings of the VLDB Endowment 8(10):1082–1093, URL https://doi.org/10.14778/2794367.2794377

Raju KVSVN, Majumdar AK (1988) Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. ACM Transactions on Database Systems 13(2):129–166, URL https://doi.org/10.1145/42338.42344

Sánchez D, Serrano JM, Blanco I, Martín-Bautista MJ, Vila MA (2008) Using association rules to mine for strong approximate dependencies. Data Mining and Knowledge Discovery 16(3):313–348, URL https://doi.org/10.1007/s10618-008-0092-3

Song S (2010) Data dependencies in the presence of difference. PhD thesis, The Hong Kong University

Song S, Chen L (2011) Differential dependencies: Reasoning and discovery. ACM Transactions on Database Systems 36:16, URL https://doi.org/10.1145/2000824.2000826

Song S, Chen L (2013) Efficient discovery of similarity constraints for matching dependencies. Data & Knowledge Engineering 87:146–166, URL https://doi.org/10.1016/j.datak.2013.06.003

Song S, Chen L, Cheng H (2014) Efficient determination of distance thresholds for differential dependencies. IEEE Transactions on Knowledge and Data Engineering 26(9):2179–2192, URL https://doi.org/10.1109/TKDE.2013.84

Song S, Sun Y, Zhang A, Chen L, Wang J (2018) Enriching data imputation under similarity rule constraints. To appear in IEEE Transactions on Knowledge and Data Engineering

Szlichta J, Golab L, Srivastava D (2015) On axiomatization and inference complexity over a hierarchy of functional dependencies. In: Proceedings of the 9th Alberto Mendelzon International Workshop on Foundations of Data Management, AMW '15

Vianu V (1987) Dynamic functional dependencies and database aging. Journal of the ACM 34(1):28–59, URL https://doi.org/10.1145/7531.7918

Wyss C, Giannella C, Robertson E (2001) FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In: Proceedings of the 3rd International Conference on Data Warehousing and Knowledge Discovery, DaWaK '01, pp 101–110, URL https://doi.org/10.1007/3-540-44801-2_11

Yao H, Hamilton HJ (2007) Mining functional dependencies from data. Data Mining and Knowledge Discovery 16(2):197–219, URL https://doi.org/10.1007/s10618-007-0083-9

Yao H, Hamilton HJ, Butz CJ (2002) FD_Mine: Discovering functional dependencies in a database using equivalences. In: Proceedings of the 2nd International Conference on Data Mining, ICDM '02, pp 729–732, URL https://doi.org/10.1109/ICDM.2002.1184040