Discovering Relaxed Functional Dependencies based on Multi-attribute Dominance

Loredana Caruccio, Vincenzo Deufemia, *Member, IEEE,* Felix Naumann, and Giuseppe Polese, *Member, IEEE*

Abstract—With the advent of big data and data lakes, data are often integrated from multiple sources. Such integrated data are often of poor quality, due to inconsistencies, errors, and so forth. One way to check the quality of data is to infer functional dependencies (FDs). However, in many modern applications it might be necessary to extract properties and relationships that are not captured through FDs, due to the necessity to admit exceptions, or to consider similarity rather than equality of data values. Relaxed FDs (RFDs) have been introduced to meet these needs, but their discovery from data adds further complexity to an already complex problem, also due to the necessity of specifying similarity and validity thresholds. We propose DOMINO, a new discovery algorithm for RFDs that exploits the concept of dominance in order to derive similarity thresholds of attribute values while inferring RFDs. An experimental evaluation on real datasets demonstrates the discovery performance and the effectiveness of the proposed algorithm.

Index Terms—Functional Dependencies, Data Profiling, Data Cleansing.

1 INTRODUCTION

MANY modern applications require integrating and processing data originating from distributed, heterogeneous, and autonomous data sources. Such data are often dirty and/or inconsistent, due to human mistakes, application errors, fraud, or different perspectives for which they were originally designed [1]. One of the solutions to detect such anomalies, and possibly resolve them, is to rely on functional dependencies (FDs), as they are properties that in principle hold only on correct data. However, there might be FDs admitting some exceptions, or that might be worth capturing on attributes that are more suitably compared through approximate matching paradigms, such as textual (e.g., address or product descriptions) and multimedia (e.g., photos or sounds) attributes. To this end, FDs have been extended by relaxing the constraint requiring them to be valid over the entire database [2], and/or by introducing approximation paradigms to compare data [3]. These so-called *relaxed functional dependencies* (RFDs) rely on thresholds to specify the similarity between tuples or to indicate the percentage of validity with respect to the entire database.

As an example, to detect fake accounts on Twitter it might be useful to identify correlations among account features [4], [5], such as their creation time and the number of their memberships to public lists. In fact, fake accounts are often created by BOTs in a short time interval, with predefined patterns for followers and public list memberships. The FD CreatedAt $\rightarrow$ ListCount needs approximate paradigms to compare both the CreatedAt (the timestamps for accounts created by a BOT might differ for few seconds), and the ListCount attribute values (the BOT might introduce small predefined differences between accounts). Moreover, the dependency should also admit exceptions (some BOTs could be smarter and introduce substantial differences in the number of public list memberships).

In order to effectively exploit RFDs in actual application domains, it is necessary to devise methods to automatically discover them from data, rather than specify them manually. However, while the availability of big data collections facilitates the discovery of meaningful RFDs, the huge amount of data and the necessity to determine the proper settings of thresholds make the discovery process computationally expensive and cumbersome. Indeed, tight thresholds can reduce the complexity of the discovery process, but might potentially prevent the detection of meaningful RFDs. On the other hand, loose thresholds not only increase the computational complexity, but might yield meaningless RFDs.

While several discovery algorithms for FDs have been proposed in the literature [6], few algorithms have been provided for RFD discovery [7]. Some of them can discover RFDs defined on approximate tuple comparison paradigms, and others discover RFDs admitting exceptions. In this paper we propose a new discovery algorithm, named DOMINO, for the class of RFDs defined on approximate tuple comparison paradigms, which in the rest of the paper we refer to as RFD_c. DOMINO relies on the concept of dominance from the multi-attribute decision theory [8] to determine the RFD_cs and the associated thresholds without requesting input parameters from the user. In particular, DOMINO represents the distance between attribute values as conflicting criteria that, if minimized, yield proper values of RFD_c thresholds. To this end, we have formalized the notion of minimal RFD_c, and have introduced a utility function to rank discovered RFD_cs in post-processing. Moreover, we provide proofs of correctness, minimality, and completeness of DOMINO.

- *L. Caruccio, V. Deufemia, and G. Polese are with the Department of Computer Science, University of Salerno, Italy. E-mail: {lcaruccio,deufemia,gpolese}@unisa.it*
- *F. Naumann is with the Hasso Plattner Institute, University of Potsdam, Germany. E-mail: felix.naumann@hpi.de*

Manuscript received October 15, 2018; revised July 16 , Accepted January 3, 2020. This is a post-peer-review, pre-copyedit version to appear on IEEE Transactions on Knowledge and Data Engineering. The early access version is available online at: https://doi.org/10.1109/TKDE.2020.2967722

We conducted an experimental evaluation of DOMINO on real datasets. In particular, we evaluated the effectiveness of the algorithm in its identification of RFD$_c$s, measuring the corresponding time and space performances by varying the number of rows and columns of the dataset. As a performance benchmark, we used the algorithms described in [9], [10], as they discover RFDs falling into the RFD$_c$ class. The results demonstrate that DOMINO achieves better performance trends in terms of time and space.

The main contributions of the paper are threefold. First, it provides a new approach for RFD$_c$ discovery, capable of automatically inferring similarity thresholds. Second, a new utility function enables the ranking of RFD$_c$s based on their cardinality and threshold values. Finally, an extensive experimental evaluation demonstrates the effectiveness and efficiency of our proposed RFD$_c$ discovery technique.

The article is organized as follows: Section 2 provides background definitions about RFDs, formulating the problem of discovering them from data, also presenting the proposed utility function. Section 3 explains how DOMINO models the RFD$_c$ discovery process, and how it exploits the concept of dominance to derive RFD$_c$s. The DOMINO algorithm itself is introduced in Section 4, whereas its experimental evaluation is reported in Section 5. Section 6 reviews the literature concerning RFD discovery algorithms. Finally, conclusions and further research are described in Section 7.

2 THE RFD DISCOVERY PROBLEM

In this section we introduce background concepts concerning the definition of RFDs, and the problem of discovering them from data.

2.1 Relaxed Functional Dependencies

Recall that an FD of a relational schema $\mathcal{R}$, defined over a set of attributes $attr(\mathcal{R})$, is a statement $X \rightarrow Y$ (X implies Y) with $X, Y \subseteq attr(\mathcal{R})$, such that, given an instance r over $\mathcal{R}$, $X \rightarrow Y$ is satisfied in r iff for every pair of tuples (t_1, t_2) in r, whenever $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$; where $t_i[X]$ denotes the projection of tuple t_i on the attribute set X.

With respect to the FD definition, which relies on exact comparisons between projections of tuples on attribute sets, the RFD definition generalizes the comparison paradigm to include similarity-based, or equivalently, distance-based comparisons, order-relations, and admitting the possibility that the dependency might hold only for a subset of a dataset.

Definition 1. Given a relational database schema $\mathcal{R}$, and $R = (A_1, \ldots, A_m)$ one of its relation schemata, an RFD ϱ on $\mathcal{R}$ is denoted by

$$X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2} \quad (1)$$

where

- $X = X_1, \ldots, X_h$ and $Y = Y_1, \ldots, Y_k$, with $X, Y \subseteq attr(R)$;
- $\Phi_1 = \bigwedge_{X_i \in X} \phi_i[X_i]$ ($\Phi_2 = \bigwedge_{Y_j \in Y} \phi_j[Y_j]$, resp.), where $\phi_i[X_i]$ ($\phi_j[Y_j]$, resp.) is a constraint on X_i (Y_j, resp.) with $i = 1, \ldots, h$ ($j = 1, \ldots, k$, resp.). Each $\phi_i[X_i]$ is either an order relation, or a predicate involving a similarity/distance function defined on the domain of X_i, plus a comparison operator with associated threshold. 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 $\phi_i[X_i]$ ($\phi_j[Y_j]$, 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 ϱ. Most commonly used coverage measures include 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.

The semantics of an RFD states that a relation instance r of R *satisfies* the RFD ϱ, denoted by $r \models \varrho$, iff: $\forall\, t_1, t_2 \in r$, if Φ_1 indicates true, then *almost always* Φ_2 indicates true. Here, *almost always* is expressed by the constraint $\Psi \leq \varepsilon$.

As an example, an RFD that might hold on the database of fake accounts described in Section 1 is:

$$\mathsf{CreatedAt}_{\phi_1} \xrightarrow{\psi(CreatedAt, \#Lists) \leq 0.2} \mathsf{\#Lists}_{\phi_2}$$

where ϕ_1 and ϕ_2 are constraints on a string similarity or distance function, and 0.2 is the maximum amount of tuples that can violate the RFD.

According to Definition 1, the canonical FD can be written as: $X_{\text{EQ}} \xrightarrow{\Psi \leq \varepsilon} Y_{\text{EQ}}$, where EQ is the equality constraint.

Definition 1 enables the description of all RFDs defined on single relation schemata. For RFDs defined on pairs of relation schemata [11] and/or subsets of them (for instance conditional RFDs) an extended version of Definition 1 can be found in [12].

Whenever $\Psi(X, Y) = 0$, that is, the RFD has to be satisfied by all tuples in r, Definition 1 can be simplified by using an arrow without symbols. RFDs satisfying this condition belong to the subclass named RFD$_c$. They are capable of capturing semantic relations between groups of values that appear to be "similar" (not just identical) and are the focus of this paper. As an example, the following RFD$_c$ captures the property that patients admitted to an hospital are assigned to similar wards if they have similar diagnoses and age:

$$\{\mathsf{Diagnosis}_{\phi_1}, \mathsf{Age}_{\phi_2}\} \rightarrow \mathsf{Ward}_{\phi_3}$$

For sake of simplicity, in order to describe the discovery process of RFD$_c$s from data, in the rest of the paper we will consider attribute constraints composed of the $\leq$ operator, and a distance function. In particular, we will refer to such kind of RFD$_c$s through the following simplified notation:

$$X_{1(\leq \alpha_1)} \ldots X_{n(\leq \alpha_n)} \rightarrow A_{(\leq \beta)}$$

where the distance function is omitted, and $\alpha_1, \ldots, \alpha_n, \beta$ are distance thresholds. All definitions, algorithms, and proofs that will be presented can be easily generalized to include other comparison operators, or order relations.

2.2 Discovering RFD$_c$s from data

The problem of discovering RFD$_c$s from data entails finding attribute combinations composing the LHS and RHS of a candidate RFD$_c$, together with proper thresholds to be associated to both sides of the dependency. Without loss of generality, in the rest of the paper we consider only

candidate RFD$_c$s with a single attribute on the RHS. Even so, the RFD$_c$ discovery problem has an extremely large search space. In fact, given a relation r with M attributes and n tuples, we need to consider all possible combinations of k attributes, k in the range $[2, M]$, by also considering for each of them k possible RHS attributes, and d^k possible threshold value combinations that can be associated to the selected attributes, where d represents the maximum number of possible values for a threshold. This complexity is synthesized by the formula:

$$\sum_{k=2}^{M} \binom{M}{k} k\, d^k$$

Notice that also the number of valid RFD$_c$s can be huge, mainly due to the inclusion property of distance functions. As an example, given an attribute A and a distance function δ on its domain, the tuple pairs t_1 and t_2 satisfying $A_{(\leq 5)}$, i.e., $\delta(t_1[A], t_2[A]) \leq 5$, will also satisfy $A_{(\leq 6)}$. This means that once an RFD$_c$ is found, many more RFD$_c$s can be derived from it by varying the threshold values (i.e., decreasing the ones on the LHS, or increasing the one on the RHS) or by adding attributes to the LHS. However, only one of these RFD$_c$s is worthwhile to be identified, that is, the *minimal* RFD$_c$, as defined in the following:

Definition 2. An RFD$_c$ $X_{1(\leq\alpha_1)} \dots X_{n(\leq\alpha_n)} \rightarrow A_{(\leq\beta)}$ on a relation r is said to be *minimal* iff

a) $X_{1(\leq\alpha_1+\varepsilon_1)} \dots X_{n(\leq\alpha_n+\varepsilon_n)} \rightarrow A_{(\leq\beta-\varepsilon_{n+1})}$ does not hold on r, where $\varepsilon_i \geq 0$ and $\exists\ j$ such that $\varepsilon_j > 0$, with $1 \leq i, j \leq n+1$; and

b) $X_{1(\leq\alpha_1)} \dots X_{i-1(\leq\alpha_{i-1})} X_{i+1(\leq\alpha_{i+1})} \dots X_{n(\leq\alpha_n)} \rightarrow A_{(\leq\beta)}$ does not hold on r, for any $1 \leq i \leq n$.

Thus, a minimal RFD$_c$ is a valid RFD$_c$ with maximum LHS thresholds and a minimum RHS threshold. In other words, if we increase an LHS threshold or decrease the RHS threshold of a minimal RFD$_c$ for some r it will no longer hold. Similar considerations apply when dropping attributes from the LHS of an RFD$_c$, since a minimal RFD$_c$ no longer holds when removing one of its LHS attributes.

As an example, let us consider the sample relation shown in Table 1, derived from a database of medical checks. Each row represents the medical check identifier (Check) and the Name of a performed check, the name of the Patient, and the prescription (PDate), execution (EDate), and release (RDate) dates. Based on Definition 2, only the first among the following RFD$_c$s[1] holding on the relation is minimal:

- $\mathsf{PDate}_{(\leq 1)} \rightarrow \mathsf{EDate}_{(\leq 1)}$,
- $\mathsf{PDate}_{(\leq 0)} \rightarrow \mathsf{EDate}_{(\leq 1)}$,
- $\mathsf{PDate}_{(\leq 0)} \rightarrow \mathsf{EDate}_{(\leq 2)}$, and
- $\mathsf{PDate}_{(\leq 1)}, \mathsf{RDate}_{(\leq 0)} \rightarrow \mathsf{EDate}_{(\leq 1)}$

This is because the LHS cardinality cannot be reduced further, and an increase of the LHS threshold or a decrease of the RHS one would lead to an invalid RFD$_c$.

Based on the definitions above, discovering RFD$_c$s over a relation instance r is the problem of finding the complete set of valid and minimal RFD$_c$s holding on r.

1. Here the threshold values represent the allowed distance in terms of the number of days elapsing between the two dates.

2.3 Utility function for ranking RFD$_c$s

Even focusing the discovery process on minimal RFD$_c$s, their number can still be high. Thus, it is necessary to devise a way to consider the most meaningful minimal RFD$_c$s holding on a relation r. To this end, we introduce a utility function, aiming to rank the set of minimal RFD$_c$s resulting from the discovery process. We refer to such a function as *utility control*. In particular, given an RFD$_c$ $\varrho : X_{1(\leq\alpha_1)} \dots X_{n(\leq\alpha_n)} \rightarrow A_{(\leq\beta)}$, the utility control aims to analyze the representativeness of $\alpha_1, \dots, \alpha_n, \beta$ with respect to the distribution of distance values on the attributes of r. The goal is to assign a higher utility value to RFD$_c$s with minimum LHS cardinality, highest possible thresholds for the LHS attributes, and the lowest possible threshold for the RHS attribute, with respect to the distribution of distance values on each attribute.

Before formally introducing the utility control, let us first define the *attribute-threshold weight*, which is used to determine the significance of a threshold for an RFD$_c$.

Definition 3. Let r be an instance of a relation schema R, $\varrho : X_{1(\leq\alpha_1)} \dots X_{n(\leq\alpha_n)} \rightarrow A_{(\leq\beta)}$ an RFD$_c$ holding on r, B an attribute in $attr(R)\backslash A$. The *attribute-threshold weight* of B for ϱ is defined as

$$\sigma_B(r, \varrho) = \begin{cases} CDF_B(\alpha_i) & \text{if } B = X_i, \text{ with } i \in [1, n] \\ 1 & \text{otherwise} \end{cases}$$

where $CDF_B(\alpha_i)$ is the *cumulative distribution function* for value α_i of the distance distribution for attribute B.

The *attribute-threshold weight* permits to evaluate the utility of an LHS (i.e., the number of attributes and the threshold values), and is computed for all attributes B in $attr(R)\backslash A$. Thus, if $B \notin \{X_1, \dots, X_n\}$, then σ_B assigns the maximum value (e.g., 1), so as to yield a higher utility value for shorter LHSs. Otherwise, if $B \in \{X_1, \dots, X_n\}$, then σ_B assigns a value that is proportional to the number of tuples matching on B with a distance value less than or equal to α_i, aiming to yield a utility value as high as the number of matching tuples.

Definition 4. Let $\varrho : X_{1(\leq\alpha_1)} \dots X_{n(\leq\alpha_n)} \rightarrow A_{(\leq\beta)}$ be an RFD$_c$ holding on an instance r of a relation schema R. The *Utility Control* (UC) of ϱ is defined as

$$UC(r, \varrho) = \frac{\sum_{B \in attr(R)\backslash A} \sigma_B(r, \varrho)}{|attr(R)\backslash A|} + (1 - CDF_A(\beta))$$

Since for the LHS the UC value must be higher for lowest LHS cardinalities and higher LHS thresholds, we compute the average value of σ_B for each attribute $B \in attr(R)\backslash A$. On the other hand, for the RHS, the UC value must be higher for lower threshold values, hence we use $1 - CDF_A(\beta)$. The utility control also admits a weighted average formulation to balance the ranking between RFD$_c$s meeting LHS or RHS minimality properties. In our experiments, we have assigned a weight of 0.5 to both terms of the UC formula.

TABLE 1
A database of medical checks.

Check	Name	Patient	PDate	EDate	RDate
1	M-Mode Echo	Robert Smith	2017-07-02	2017-07-15	2017-07-18
2	ECG	Claire Green	2017-07-03	2017-07-16	2017-07-20
2	Echo	Jason Smith	2017-07-02	2017-07-16	2017-07-20
3	2Echocardiography	John Stewart	2017-07-05	2017-07-19	2017-07-19
2	Echocardiography	J Smith	2017-07-08	2017-07-19	2017-07-21
2	ECG	W J Smith	2017-07-12	2017-07-21	2017-07-24

3 DOMINANCE-BASED RFDc DISCOVERY

In this section, we present our RFD_c discovery methodology. It exploits the concept of dominance among vectors of distance values between tuple pairs of a relation instance to automatically derive the minimal RFD_cs and the thresholds associated to their attributes.

3.1 Solution overview

The goal of the proposed discovery process is to find the set of all minimal RFD_cs, including their associated distance thresholds. To pursue this goal, the process performs the following steps: first it computes the set of all distance vectors between tuple pairs of the input instance r of a relation schema R. A distance vector is associated to each tuple pair of r, and contains as many elements as the number of attributes of R, where the value of each element corresponds to the distance that the two tuples have on a given attribute. Second, for each attribute A of R, it seeks minimal RFD_cs with A as RHS by filtering distance vectors with value greater than β on A, for each possible threshold value β that attribute A can assume. Among these, it filters those vectors yielding minimal RFD_cs by exploiting the concept of *dominance* [8]:

Definition 5. Given two numerical vectors v_i and v_j, each containing n elements, v_i is said to *dominate* v_j iff $\forall\, h \in [1,n]\ v_i[X_h] \geq v_j[X_h]$ and $\exists\ k \in [1,n]$ s.t. $v_i[X_k] > v_j[X_k]$.

In particular, for each subset X of attributes of $attr(R)\backslash A$, among the filtered distance vectors, those yielding minimal RFD_cs are the ones that do not dominate each other on the values associated to X. They form a boundary, which has the property that a candidate RFD_c having $A_{(\leq\beta)}$ as RHS, and the attributes of X on the LHS, cannot be valid if the vector of its LHS thresholds either dominates or equals one of the distance vectors in the boundary. Thus, starting from each distance vector in the boundary, the proposed discovery process derives minimal RFD_cs by manipulating its elements in order to reach the greatest possible value combinations forming vectors that do not dominate each other, and none of the distance vectors in the boundary. This is done by tweaking the values of a distance vector in the boundary, lowering one of its elements, and trying to maximize the remaining ones to the extent that the derived vectors do not dominate or equal those in the boundary. As said above, each of the so derived vectors has associated an attribute set X of R, from which the LHS of a minimal RFD_c with $A_{(\leq\beta)}$ as RHS is derived by assigning the vector values as thresholds for the attributes in X.

As an example, given the relation instance shown in Table 1, in the first step the discovery process computes the distance vectors between tuple pairs, as shown in Figure 1(i). Thus, starting from them, by considering the attribute PDate as candidate RHS with possible threshold $\beta = 1$, and $X = \{\text{Patient, EDate, RDate}\}$ as candidate LHS attribute set, among the distance vectors having $\beta > 1$ for attribute PDate there exist some that dominate others on X. For instance, the distance vector $(6,6,6)$, derived projecting v_{14} on X, dominates $(5,5,4)$ derived from v_{15}. As a consequence, it cannot belong to the boundary for X and can be ignored. In particular, the boundary for attribute set X contains the following distance vectors: $(9,0,2)$ from v_7, $(1,2,3)$ from v_9, and $(4,3,1)$ from v_{11}. They do not dominate any other distance vectors having $\beta > 1$ for PDate.

It is easy to verify that any RFD_c candidate with $\text{PDate}_{(\leq 1)}$ as RHS, and X as LHS attribute set with threshold combination that either dominates or equals at least one distance vector in the boundary is not valid. For instance, the candidate RFD_c $\text{Patient}_{(\leq 5)}$,$\text{EDate}_{(\leq 3)}$,$\text{RDate}_{(\leq 1)}$ $\rightarrow$ $\text{PDate}_{(\leq 1)}$ is not valid, because the projection of v_{11} on Patient, EDate, RDate, i.e., $(4,3,1)$, represents one or more tuple pairs whose distance on the LHS attributes is below the LHS thresholds, i.e., $(5,3,1)$, but whose distance on PDate (i.e., the projection of v_{11} on PDate) is 6, hence is greater than the RHS threshold 1. Instead, a minimal RFD_c is $\text{Patient}_{(\leq 8)}$, $\text{EDate}_{(\leq 2)}$, $\text{RDate}_{(\leq 2)}$ $\rightarrow$ $\text{PDate}_{(\leq 1)}$. In fact, the RFD_c is trivially valid, since there are no distance vectors with $\beta > 1$ for PDate satisfying the candidate LHS thresholds. Moreover, by increasing even one of the three LHS threshold values yields a vector of thresholds that either dominates or equals at least one distance vector in the boundary.

More formally, given an instance r of a relation schema R, the proposed discovery process starts by computing a relation $\delta(r)$ containing all distinct distance values between all tuple pairs of r. This is called *distance relation* and is defined as follows:

Definition 6. Let r be an instance of a relation schema $R = (X_1, \ldots, X_m)$, and $\delta_{X_1}, \ldots, \delta_{X_m}$ a list of distance functions on $dom(X_1)$,..., $dom(X_m)$, respectively. The *distance relation* $\delta(r)$ of r is defined as:

$$\delta(r) = \{(\delta_{X_1}(t_i[X_1], t_j[X_1]), \ldots, \delta_{X_m}(t_i[X_m], t_j[X_m]))\mid t_i, t_j \in r \text{ and } i \neq j\}.$$

Each tuple of $\delta(r)$ or its projection on two or more attributes corresponds to a *distance vector*.

Next, the discovery process is iteratively accomplished for each candidate RHS attribute A_i of R, by

(i) sorting $\delta(r)$ based on the distance values on attribute A_i, yielding $\delta(r)_{A_i}$,

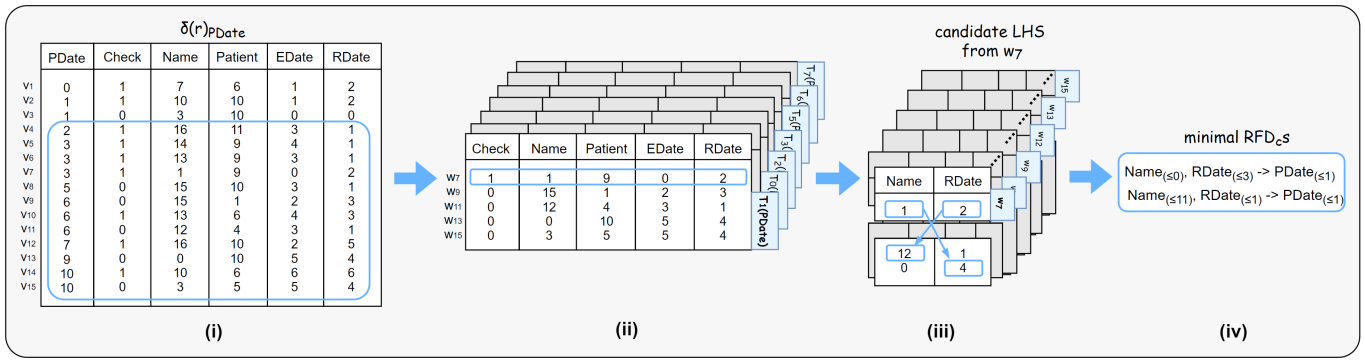


Fig. 1. The proposed discovery process with $A_i =$ PDate and $\beta = 1$.

(ii) constructing a relation $T_\beta(A_i)$ on $attr(R)\backslash A_i$, for each value of $\beta \in \pi_{A_i}(\delta(r)_{A_i}) \cup \{0\}$, containing only distance vectors contributing to the discovery of RFD$_c$s. In particular, each $T_\beta(A_i)$ contains all distance vectors $v \in \delta(r)_{A_i}$ such that $v[A_i] > \beta$, which do not dominate each other,

(iii) generating LHS candidates from all possible subsets of attributes in $attr(R)\backslash A_i$ and the distance vectors in $T_\beta(A_i)$, and

(iv) determining the combination of threshold values for each LHS candidate to generate the minimal RFD$_c$s.

Figure 1 shows this process on the relation of Table 1, with A_i=PDate and $\beta = 1$. The distance functions used to generate $\delta(r)$ are: the equality predicate for the Check attribute, the Levenshtein distance for attributes Name and Patient, and the number of days for the two date attributes. Figure 1(ii) shows T_1(PDate) and highlights the distance vector used for generating the LHS candidate of Figure 1(iii). The latter involves attributes Name and RDate, with distance values 1 and 2, respectively. These are used to determine the combination of threshold values for minimal RFD$_c$s, which are the ones shown in Figure 1(iv). The following sections detail steps (ii)-(iv) of the RFD$_c$ discovery process.

3.2 Constructing T_β relations

Given a candidate RHS attribute A_i of R and the instance $\delta(r)_{A_i}$, the goal of this step is to filter out all distance vectors of $\delta(r)_{A_i}$ that cannot contribute to the discovery of valid RFD$_c$s with A_i as RHS. This filtering entails computing the relations $T_\beta(A_i)$ for each candidate value of β, that is, for each value of A_i in $\delta(r)_{A_i}$ beyond the maximum value. Such relations are derived by exploiting the concept of dominance. In fact, if a distance vector v dominates a vector v', then v and its sub-vectors can never belong to a boundary whatever subset of attributes in $attr(R)\backslash A_i$ is considered as LHS candidate, hence it can be ignored during the RFD$_c$ discovery process.

The construction of the relation $T_\beta(A_i)$ starts from $\beta = max(\pi_{A_i}(\delta(r)_{A_i}) - max(\pi_{A_i}(\delta(r)_{A_i})))$, i.e., the second highest value of A_i in $\delta(r)_{A_i}$, and considers values from $\pi_{A_i}(\delta(r)_{A_i}) \cup \{0\}$ in decreasing order. In particular, let $\{d_1, \ldots, d_m\}$ be the values of β with $d_1 = 0$, $d_j < d_{j+1}$, and $1 \leq j < m$. The construction process initializes $T_{d_j}(A_i)$ with $T_{d_{j+1}}(A_i)$, and for each distance vector $v \in \delta(r)_{A_i}$, with $d_j < v[A_i] \leq d_{j+1}$, if the vector $w = \pi_{attr(R)\backslash A_i}(v)$ does not dominate other vectors already in $T_{d_j}(A_i)$, then it adds w to $T_{d_j}(A_i)$, and removes all distance vectors dominating w from it.

As an example, the T_β(PDate) relations for $\delta(r)_{\text{PDate}}$ of Figure 1(i) are shown in Table 2. To illustrate the construction process, we denote with v_j the distance vector of row j from $\delta(r)_{\text{PDate}}$, and with w_j its projection on $attr(R)\backslash$PDate. Starting with $\beta = 9$, the distance vectors v_{14} and v_{15} are the only ones having a value for PDate greater than 9, but as w_{14} dominates w_{15}, only the latter is added to T_9(PDate). For $\beta = 7$, T_7(PDate) is initialized with T_9(PDate), and only w_{13} is added to it since v_{13} is the only one for which $7 < v_{13}$[PDate] ≤ 9, and w_{13} does not dominate w_{15}. Conversely, for $\beta = 3$, T_3(PDate) is initialized with T_6(PDate), the distance vectors v_8, v_9, v_{10}, and v_{11} are the only ones having a value for PDate in the range $]2, 6]$, both w_8 and w_{10} dominate w_{11}, and w_{12} dominates w_9. Thus, w_9 and w_{11} are added to T_3(PDate) and w_{12} is removed.

TABLE 2
The T_β relations for PDate.

		Check	Name	Patient	EDate	RDate
T_0(PDate)	w_3	0	3	10	0	0
	w_7	1	1	9	0	2
	w_9	0	15	1	2	3
	w_{11}	0	12	4	3	1
	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4
T_1(PDate)	w_7	1	1	9	0	2
	w_9	0	15	1	2	3
	w_{11}	0	12	4	3	1
	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4
T_3(PDate)	w_9	0	15	1	2	3
	w_{11}	0	12	4	3	1
	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4
T_6(PDate)	w_{12}	1	16	10	2	5
	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4
T_7(PDate)	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4
T_9(PDate)	w_{15}	0	3	5	5	4

Each relation $T_\beta(A_i)$ is analyzed for deriving LHS candidates for RFD$_c$s with $A_{i(\leq\beta)}$ as RHS, trying to minimize the number of LHS attributes, and maximize their threshold values. This process is described next.

3.3 Generating LHS candidates

In this section we describe how to find LHS candidates for RFD$_c$s from a given relation $T_\beta(A_i)$. To this end, we need to consider all possible k-combinations of $n-1$ attributes from $T_\beta(A_i)$. We do this by examining the distance vectors in $T_\beta(A_i)$.

Formally, let $w \in T_\beta(A_i)$ be a distance vector with $n-1$ attributes. A sub-vector $s = \pi_{A_{k_1},\ldots,A_{k_l}}(w)$ with $1 \leq l \leq n-1$ is said to be a *viable candidate* iff there is no distance vector $w' \in T_\beta(A_i)$, $w' \neq w$, such that $s' = \pi_{A_{k_1},\ldots,A_{k_l}}(w') \neq s$, and s dominates s'. The verification of this property requires the pairwise comparison of all distance vectors in $T_\beta(A_i)$. To this end, for each $w \in T_\beta(A_i)$ we construct a *difference matrix* M_w^β containing the differences between w and any other distance vector in $T_\beta(A_i)$. For instance, if we consider the relation $T_1(\mathsf{PDate})$ and the distance vector w_7 shown in Table 2, each row of the matrix $M_{w_7}^1$ contains the differences between the values of w_7 and those of another distance vector in $T_1(\mathsf{PDate})$, as shown in Table 3.

TABLE 3
The difference matrix for w_7 of $T_1(\mathsf{PDate})$.

	Check	Name	Patient	EDate	RDate
$w_7 - w_9$	1	-14	8	-2	-1
$w_7 - w_{11}$	1	-11	5	-3	1
$w_7 - w_{13}$	1	1	-1	-5	-2
$w_7 - w_{15}$	1	-2	4	-5	-2

The formal definition of viable candidate in terms of the difference matrix is the following:

Definition 7. Let $w \in T_\beta(A_i)$ be a distance vector with $n-1$ attributes, $s = \pi_{A_{k_1},\ldots,A_{k_l}}(w)$, $1 \leq l \leq n-1$, and M_w^β the difference matrix of w; s is said to be a *viable candidate* iff for each row h of M_w^β

1) $\exists\ f$ such that $M_w^\beta(h, k_f) < 0$, or
2) $\forall\ f \in [1, l]\ M_w^\beta(h, k_f) = 0$.

The validation of the viability property allows us to identify LHS candidates of RFD$_c$s. Since to determine viable candidate vectors we have to analyze a huge number of distance vectors of different sizes, we reduce the search space by introducing the notion of distance vector *minimality*. In particular, a viable candidate vector is *minimal* when property (1) of Definition 7 holds for each row of the difference matrix. The minimality of a distance vector s guarantees that each distance vector u containing s as sub-vector is also a viable candidate. But since it will not be useful to determine RFD$_c$s with minimal LHSs, it can be pruned during the discovery process.

A further pruning rule that can be applied during the discovery of viable candidates is the identification of *non-negative* distance vectors:

Definition 8. Let $w \in T_\beta(A_i)$ be a distance vector with $n-1$ attributes, $s = \pi_{A_{k_1},\ldots,A_{k_l}}(w)$, $1 \leq l \leq n-1$, and M_w^β the difference matrix of w; s is said to be *non-negative* iff for each row h of M_w^β

1) $\forall\ f \in [1, l]\ M_w^\beta(h, k_f) \geq 0$, and
2) $\exists\ f$ such that $M_w^\beta(h, k_f) > 0$

As before, a distance vector u including a non-negative distance vector s as sub-vector is not useful to determine RFD$_c$s with minimal LHSs. Also, according to Definition 7, a non-negative distance vector s is not viable. As a consequence, when a non-negative distance vector is identified, the discovery process can ignore all the distance vectors including it.

As an example, let us consider the distance vector w_7 in $T_1(\mathsf{PDate})$ of Table 2, whose difference matrix $M_{w_7}^1$ is shown in Table 3. Figure 2 shows the result of searching viable candidates of w_7. The search process performs a bottom-up, level-by-level analysis of the lattice structure. In particular, the distance vector (1) on the attribute Check represents a non-negative distance vector of w_7. In fact, in Table 3 we notice that the non-negativity property is satisfied for column Check, hence according to the non-negativity pruning rule, all distance vectors containing the Check attribute are pruned and will not be analyzed.

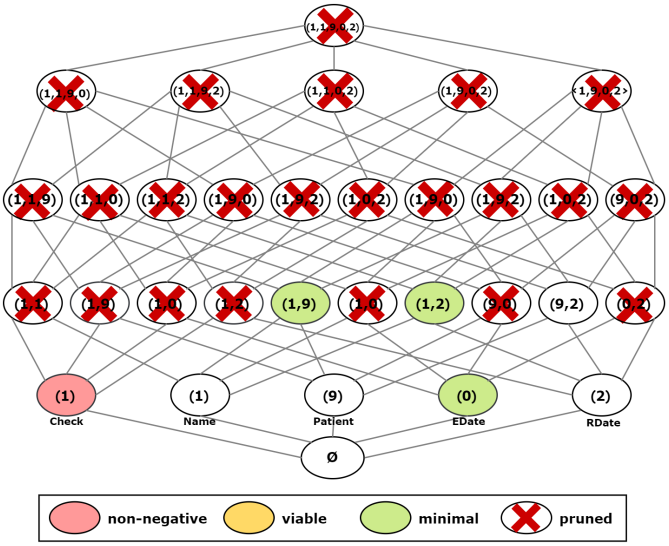


Fig. 2. Lattice of all distance vectors included in w_7 of $T_1(\mathsf{PDate})$. The figure highlights the distance vectors that the search process has detected as non-negative, viable, minimal, and those that have been pruned.

Moreover, the distance vector (0) on the attribute EDate is a minimal viable candidate. In fact, according to EDate values of Table 3, property (1) of Definition 7 is satisfied. Also, in this case, the search process will not analyze the distance vectors containing the EDate attribute. The process continues on the second level, where several minimal distance vectors are found. No further distance vectors need be analyzed from the third level in our example. At the end, all viable candidates are used to determine the RFD$_c$s holding on r, as detailed in the next section.

3.4 Determining threshold values of minimal RFD$_c$s

The goal of this step is to derive the thresholds of minimal RFD$_c$s starting from the viable candidates determined in the previous step. In particular, for each viable candidate s derived from $T_\beta(A_i)$, we determine the RFD$_c$s with $A_{i(\leq\beta)}$ as RHS, by searching the maximum non-dominating *thresholds* for the LHS attributes associated to s. This process is extended to all non-dominating distance vectors of $T_\beta(A_i)$ including s as sub-vector, level-by-level, until reaching a level with no RFD$_c$s.

The following rules allow to derive valid RFD_cs $X_{1(\leq\alpha_1)}\dots X_{h(\leq\alpha_h)} \to A_{(\leq\beta)}$ from a given viable candidate. For a better understanding, we introduce the rules subsequently for left-hand-sides with one attribute, with two attributes and then generalize to h attributes.

LHS with one attribute. Given a distance vector w in $T_\beta(A)$, if there exists a viable candidate $s = \pi_{X_k}(w) = (d_k)$, with $1 \leq k \leq h$ and $d_k > 0$, then a valid RFD_c is:

$$X_{k(\leq d_k - \varepsilon)} \to A_{(\leq\beta)}$$

where $\varepsilon > 0$ is a value determining the distance threshold $d_k - \varepsilon$ associated to the attribute X_k.

As an example, consider the first relation T_1(PDate) shown in Table 4. For the attribute RDate we have the viable candidate $s = (1)$, and for $\varepsilon = 1$ the RFD_c $\text{RDate}_{(\leq 0)} \to \text{PDate}_{(\leq 1)}$ can be determined.

Notice that according to the minimality property of RFD_cs, if s is also in a $T_{\beta'}(A)$, with $\beta' < \beta$, then it is considered only as the RFD_c with RHS threshold β'.

LHS with two attributes. Given a distance vector w in $T_\beta(A)$, if there exists a viable candidate $s = \pi_{X_{k_1},X_{k_2}}(w) = (d_1, d_2)$, with $1 \leq k_1 < k_2 \leq h$, then there exist at most the two following RFD_cs:

$$X_{k_1(\leq d_1-\varepsilon)}, X_{k_2(\leq\alpha_2)} \to A_{(\leq\beta)} \qquad \text{if } d_1 > 0 \text{ and } \exists\alpha_2$$

$$X_{k_1(\leq\alpha_1)}, X_{k_2(\leq d_2-\varepsilon)} \to A_{(\leq\beta)} \qquad \text{if } d_2 > 0 \text{ and } \exists\alpha_1$$

If the threshold value α_1 (α_2, resp.) exists, it will be determined according to s, and by analyzing the viable candidates s' in $T_\beta(A)$, with $s' \neq s$. In particular, $\alpha_j = p_j - \varepsilon$, where $p_j = \min_{s' \neq s} \pi_{X_j}(s')$ such that $p_j > d_j$ and $p_i < d_i$, for $i = 1, j = 2$, and vice versa.

As an example, let us consider T_1(PDate) in Table 4. For the attributes Name and RDate we have $s = \pi_{\text{Name,RDate}}(w_7) = (1, 2)$, and for $\varepsilon = 1$ the following RFD_cs can be determined:

1) $\text{Name}_{(\leq 0)}, \text{RDate}_{(\leq 3)} \to \text{PDate}_{(\leq 1)}$;
   from the first rule: $d_1 - \varepsilon = \pi_{\text{Name}}(w_7) - 1 = 1 - 1 = 0$, hence the threshold for the attribute Name is set to 0, whereas the threshold α_2 for the attribute RDate is set to 3. In fact, $\min_{s' \neq s} \pi_{\text{RDate}}(s')$ such that $\pi_{\text{RDate}}(s') > \pi_{\text{RDate}}(w_7)$ and $\pi_{\text{Name}}(s') < \pi_{\text{Name}}(w_7)$ is the value 4 of RDate from the distance vector w_{13}. Thus, $\alpha_2 = p_2 - \varepsilon = \pi_{\text{RDate}}(w_{13}) - 1 = 4 - 1 = 3$.
2) $\text{Name}_{(\leq 11)}, \text{RDate}_{(\leq 1)} \to \text{PDate}_{(\leq 1)}$;
   from the second rule: $d_2 - \varepsilon = \pi_{\text{RDate}}(w_7) - 1 = 2 - 1 = 1$, hence the threshold for the attribute RDate is set to 1, whereas the threshold α_1 for the attribute Name is set to 11. In fact, $\min_{s' \neq s} \pi_{\text{Name}}(s')$ such that $\pi_{\text{Name}}(s') > \pi_{\text{Name}}(w_7)$ and $\pi_{\text{RDate}}(s') < \pi_{\text{RDate}}(w_7)$ is the value 12 of Name from the distance vector w_{11}. Thus, $\alpha_1 = p_1 - \varepsilon = \pi_{\text{Name}}(w_{11}) - 1 = 12 - 1 = 11$.

LHS with h attributes. Given a distance vector w in $T_\beta(A)$, if there exists a viable candidate $s = \pi_{X_1,\dots,X_h}(w) = (d_1, \dots, d_h)$, then for each $d_i > 0$, with $i = 1, \dots, h$, the following RFD_c holds:

$$X_{1(\leq\alpha_1)}, \dots, X_{i-1(\leq\alpha_{i-1})}, X_{i(\leq d_i-\varepsilon)}, \\ X_{i+1(\leq\alpha_{i+1})}, \dots, X_{h(\leq\alpha_h)} \to A_{(\leq\beta)}$$

provided that the threshold values α_j exist for all $j = 1, \dots, i-1, i+1, \dots, h$. In this case, each α_j is determined according to s, and by analyzing the viable candidates s' in $T_\beta(A)$, with $s' \neq s$. In particular, we have to consider two different cases, based on the distance vector v obtained by removing d_i from s.

Case 1: If v is a non-dominating distance vector, then $\alpha_j = p_j - \varepsilon$, where $p_j = \min_{s' \neq s} \pi_{X_j}(s')$ such that:

1) $p_j > d_j$;
2) $p_i < d_i$, with $p_i = \pi_{X_i}(s')$;
3) $p_e \leq \alpha_e$, for each e in $[1, j-1]$ and $e \neq i$;
4) $\exists d_{e'} \geq p_{e'}$, with e' in $[j+1, h]$ and $e' \neq i$.

Case 2: If v is a dominating distance vector, then $\alpha_j = p_j - \varepsilon$, where $p_j = \min_{s' \neq s} \pi_{X_j}(s')$ such that:

1) $p_j > d_j$;
2) $p_i < d_i$, with $p_i = \pi_{X_i}(s')$;
3) $p_e \leq \alpha_e$, for each e in $[1, j-1]$ and $e \neq i$;
4) $d_{e'} \geq p_{e'}$, with e' in $[j+1, h]$ and $e' \neq i$.

As an example, for attributes Patient, EDate, and RDate in T_1(PDate) of Table 4, we have $s = \pi_{\text{Patient,EDate,RDate}}(w_9) = (1, 2, 3)$, and for $\varepsilon = 1$ the following RFD_c can be determined:

$$\text{Patient}_{(\leq 8)}, \text{EDate}_{(\leq 2)}, \text{RDate}_{(\leq 2)} \to \text{PDate}_{(\leq 1)}$$

since when $i = 3$ we have $d_3 - \varepsilon = \pi_{\text{RDate}}(w_9) - 1 = 3 - 1 = 2$, and according to the properties of case 1, α_1 from the attribute Patient is set to 8, and α_2 from the attribute EDate is set to 2. In particular, the vector w_7 satisfies $\pi_{\text{Patient}}(w_7) = \min_{s' \neq s} \pi_{\text{Patient}}(s')$ and

1) $\pi_{\text{Patient}}(w_7) > \pi_{\text{Patient}}(w_9)$,
2) $\pi_{\text{RDate}}(w_7) < \pi_{\text{RDate}}(w_9)$, and
4) $\pi_{\text{EDate}}(w_9) \geq \pi_{\text{EDate}}(w_7)$.

Thus, $\alpha_1 = p_1 - \varepsilon = \pi_{\text{Patient}}(w_7) - 1 = 9 - 1 = 8$. Similarly, the vector w_{11} satisfies $\pi_{\text{Patient}}(w_{11}) = \min_{s' \neq s} \pi_{\text{EDate}}(s')$ such that

1) $\pi_{\text{EDate}}(w_{11}) > \pi_{\text{EDate}}(w_9)$,
2) $\pi_{\text{RDate}}(w_{11}) < \pi_{\text{RDate}}(w_9)$, and
3) $\pi_{\text{Patient}}(w_{11}) \leq \alpha_1 = 8$.

Thus, $\alpha_2 = p_2 - \varepsilon = \pi_{\text{Patient}}(w_{11}) - 1 = 3 - 1 = 2$.

When $i = 1$ or $i = 2$ no RFD_c is generated. In particular, when $i = 1$, we have $d_1 - \varepsilon = \pi_{\text{Patient}}(w_9) - 1 = 1 - 1 = 0$, but in this case it does not exist any s' satisfying the property

TABLE 4
T_1(PDate) highlighting some viable candidates.

		Check	Name	Patient	EDate	RDate
T_1(PDate)	w_7	1	1	9	0	2
	w_9	0	15	1	2	3
	w_{11}	0	12	4	3	1
	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4
T_1(PDate)	w_7	1	1	9	0	2
	w_9	0	15	1	2	3
	w_{11}	0	12	4	3	1
	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4
T_1(PDate)	w_7	1	1	9	0	2
	w_9	0	15	1	2	3
	w_{11}	0	12	4	3	1
	w_{13}	0	0	10	5	4
	w_{15}	0	3	5	5	4

2), i.e., $\pi_{\text{Patient}}(s') < \pi_{\text{Patient}}(w_9)$. Similarly, when $i = 2$, we have $d_2 - \varepsilon = \pi_{\text{EDate}}(w_9) - 1 = 2 - 1 = 1$, and we can determine α_1 as $\pi_{\text{Patient}}(w_7) - 1 = 9 - 1 = 8$. However, we cannot determine α_3, since there exist no vector w such that $\pi_{\text{RDate}}(w) = \min_{s' \neq s} \pi_{\text{RDate}}(s')$, 1) $\pi_{\text{RDate}}(w) > \pi_{\text{RDate}}(w_9)$, and 2) $\pi_{\text{EDate}}(w) < \pi_{\text{EDate}}(w_9)$.

Notice that the user might want to constrain the range in which DOMINO should search threshold values. For instance, metric functional dependencies are RFD$_c$s in which the RHS threshold is always 0 [13]. DOMINO can accommodate such constraints by restricting its search strategy to threshold values within certain intervals. To this end, DOMINO will take in input a vector of maximum threshold values for each attribute, plus a specification whether such upper limits should hold always or only when the single attribute appears as RHS. In the first case, during the construction of the distance relation $\delta(r)$, DOMINO filters out the distance vectors exceeding specified threshold limits, whereas in the second case, it will construct T_β relations only for values of β less or equal to the threshold limit.

4 THE DOMINO ALGORITHM

In this section we present DOMINO and its procedures. Figure 3 summarizes the main steps of the DOMINO's discovery process, which are implemented by Algorithm 1. The process includes a data pre-processing phase, which relies on the input relation to construct ordered distance relations, one for each attribute (Line 4). Next, in Lines 5–8 DOMINO invokes the procedures to construct the T_β relations (Algorithm 2) and generate minimal RFD$_c$s (Algorithm 3). The detailed pseudo-code of DOMINO's main steps is available in the appendix of the online supplemental material.

At each step, Algorithm 2 picks the next distance vector w in descending order of attribute A_i (Line 2) and adds

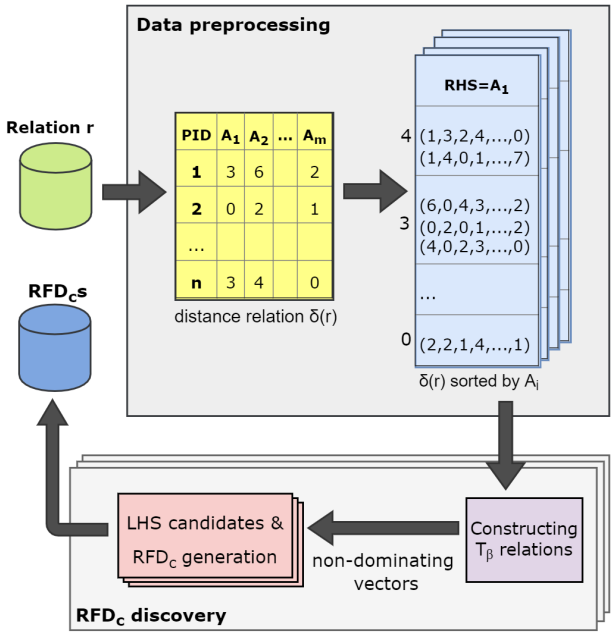


Fig. 3. The process of RFD$_c$ discovery of DOMINO.

Algorithm 1: DOMINO

Input : An instance relation r of size n
Output: The minimal set RFDcs of holding RFD$_c$s

```
1 RFDcs ← ∅;
2 δ(r) ← getDistanceRelation(r);
3 foreach A_i ∈ attr(R) do
4     δ(r)_{A_i} ← orderedRelation(δ(r));
5     T ← getTRel(δ(r)_{A_i});          // Alg. 2
6     foreach T_β ∈ T do
7         foreach w ∈ T_β do
8             RFDcs ← RFDcs ∪ getRFDs(T_β, w);
                // Alg. 3
9 return RFDcs;
```

Algorithm 2: Construction of T_β relations

Input : An ordered difference relation $\delta(r)_{A_i}$
Output: Set T of $T_\beta(A_i)$ relations

```
1  T ← ∅; T_β ← ∅;
2  foreach w ∈ δ(r)_{A_i} in descending order of A_i do
3      β ← next value of A_i smaller than w[A_i];
4      if the value β has not been issued before then
5          T ← T ∪ T_β;
6      if w does not dominate vectors in T_β then
7          add w to T_β;
8          remove from T_β vectors dominating w;
9  T ← T ∪ T_β;
10 return T;
```

it to a T_β relation whenever it does not dominate distance vectors already in T_β (Lines 7–8). Finally, when the β value changes (Lines 4–5), the algorithm saves the current T_β into the result set T, and uses it as the starting point for the new T_β.

Algorithm 3 considers the role of each distance vector in its T_β relation, by determining its minimal LHSs (*LHS minimality*), from which all possible valid RFD$_c$s can be generated (RFD$_c$ *generation*). In particular, the algorithm performs a combinatorial level-wise search to analyze all feasible LHSs of a given distance vector $w \in T_\beta$. After the creation of the difference matrix *DM* (Line 2), there is an *initialization phase* (Lines 3–4) where the lattice nodes of level 1 are created from *DM*'s columns. The *search phase* (Lines 5–11) validates and prunes column combinations according to the considered lattice level, yielding feasible LHSs. Furthermore, the lattice nodes of the next level are generated at Line 11. Finally, RFD$_c$ generation (Algorithm 4) is invoked at Line 12.

Algorithm 4 projects the input distance vector w on the columns of each input LHS (cc), and for each of these values (w_i) it evaluates the possibility to generate an RFD$_c$ by trying to associate a threshold to each column in cc, as defined in Section 3.4 (Line 3–5). In particular, when it is possible to define thresholds, a new RFD$_c$ is constructed (Line 5) and is added to the set containing previously discovered RFD$_c$s. However, if a LHS with the same thresholds is already contained in that set, the corresponding RFD$_c$ will be removed,

Algorithm 3: Generation of LHS candidates & Determination of thresholds for minimal RFD_cs

Input : A T_β relation from Algorithm 2
A distance vector w
Output: Set of discovered RFDcs for T_β

```
1  L ← 1; levelAttr ← ∅; LHS ← ∅;
2  create the difference matrix DM of w;
   // Initialization phase
3  foreach column j of DM do
4      add j to the set levelAttr of lattice nodes of
         level L;
   // Search LHS candidates
5  while L ≤ |w| or levelAttr ≠ ∅ do
6      foreach X in levelAttr do
7          if w[X] is viable then
8              LHS ← LHS ∪ X;
9              identify lattice nodes of level L+1
                 that can be pruned;
10     L ← L +1;
11     initialize levelAttr with the lattice nodes of
         level L without the pruned ones;
12 return generateRFDcs(T_β, w, LHS);
```

Algorithm 4: Determination of threshold values for minimal RFD_cs

Input : A T_β relation
A distance vector w
A set LHS of column combinations
Output: Set of discovered RFDcs for T_β

```
1  RFDcs ← ∅ ;
2  foreach cc ∈ LHS do
3      foreach value w_i of w[cc] do
4          if properties defined in 3.4 are satisfied by
             w_i, w[cc], and T_β then
5              create a new RFD_cs based on the
                 thresholds generated according to
                 Section 3.4, remove from RFDcs all
                 RFD_cs with same LHS and RHS
                 attributes, same LHS thresholds, and
                 higher RHS threshold, and add the
                 new RFD_cs to RFDcs;
6      foreach A ∉ cc do
7          if w[cc∪A] does not dominate vectors in T_β
             then
8              LHS ← LHS ∪ {cc ∪ A} ;
9  return RFDcs;
```

since it could be defined from a greater RHS threshold. Indeed, according to the *minimality property* (Section 2.2) we keep the RFD_cs with lower RHS thresholds. Moreover, new column combinations are added to the current LHS set, by combining $w[cc]$ with other attributes, provided that the resulting distance vector does not dominate vectors in T_β (Lines 6–8).

In what follows, we prove correctness, minimality, and completeness of the proposed DOMINO algorithm.

Correctness. One of the evaluation dimensions of a dependency discovery algorithm is correctness, which guarantees that all discovered dependencies are indeed valid.

Theorem 1. Each RFD_c discovered by DOMINO is valid.

Proof: We proceed by contradiction. Assume that DOMINO discovers an RFD_c $\varrho : X_{1(\leq\alpha_1)}, \ldots, X_{n(\leq\alpha_n)} \rightarrow A_{(\leq\beta)}$ that is *not* valid for the input instance r. Then, according to the definition of RFD_c, there exists at least one tuple pair (t_1, t_2) for which the constraints $X_{1(\leq\alpha_1)}, \ldots, X_{n(\leq\alpha_n)}$ are satisfied, but not $A_{(\leq\beta)}$. This means that for attribute constraints based on distance functions, $A_{(\leq\beta)}$ is not satisfied iff the distance between $t_1[A]$ and $t_2[A]$ is greater than β.

DOMINO discovers the RFD_cs for an RHS threshold β by analyzing all non-dominating distance vectors in T_β. Thus, if the RFD_c ϱ is not valid, then there exists at least one distance vector in the T_β relation that is dominated by the distance vectors $(\alpha_1, \ldots, \alpha_n)$.

While discovering an RFD_c, each attribute threshold is iteratively determined according to properties defined in Section 3.4, where thanks to Properties 2 and 3, in each iteration DOMINO determines all distance vectors in T_β dominated by the thresholds defined in previous iterations. Among them, DOMINO selects the minimum value d_j for X_j, by generating a threshold $\alpha_j = d_j - \varepsilon$. Once this process has determined the value of $\alpha_1, \ldots, \alpha_{n-1}$, DOMINO considers all distance vectors dominated by them (Properties 2 and 3), and selects the minimum distance d_n for X_n, generating $\alpha_n = d_n - \varepsilon$. This implies that no distance vector in T_β can be dominated by $\alpha_1, \ldots, \alpha_n$, contradicting the original assumption. □

Minimality. Another important evaluation dimension of a dependency discovery algorithm is its minimality, which guarantees that the discovered RFD_cs no longer hold after either (i) increasing one or more LHS thresholds, (ii) removing an LHS attribute, or (iii) decreasing the RHS threshold.

Theorem 2. DOMINO discovers a set of minimal RFD_cs according to the minimality property defined in Section 2.2.

Proof: The discovery process of DOMINO guarantees the minimization of the RHS threshold for two reasons: (i) for each candidate RHS attribute, the procedure for the *Construction of T_β relations* includes in each relation T_β all non-dominating distance vectors that can generate RFD_cs; (ii) Lines 6–7 of Algorithm 4 remove all RFD_cs with the same LHS thresholds, but a higher RHS threshold, since the T_β relations are examined in descending order.

DOMINO guarantees the maximization of LHS thresholds for two reasons: (i) during the process for the *Generation of LHS candidates*, each distance vectors in T_β is used to determine all minimal distance (sub-)vectors by considering all possible attribute combinations. The search process terminates when it finds either a minimal distance vector, or a distance vector that cannot be part of a non-dominating one. This means that all non-dominating distance vectors are considered for the *Determination of thresholds for minimal* RFD_cs; (ii) given a non-dominating distance vector

$s = (m_1, \ldots, m_k)$, the process for determining thresholds for RFD$_c$s tries to decrease each distance value m_i, and to determine thresholds α_j for $j \neq i$. Property 1 of Section 3.4 guarantees that each α_j is greater or equal to m_j, that is, LHS thresholds derived from s never decrease more than one distance value of s.

The minimization of attributes on the LHS is guaranteed by Property 4 of Section 3.4. In particular, referring to *Case 1* of that section, given a non-dominating distance vector $s = (m_1, \ldots, m_k)$, the process for the determination of thresholds for RFD$_c$s tries to decrease each distance value m_i, and to determine thresholds α_j, for $j \neq i$, when the distance vector $s' = (m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_k)$ is non-dominating, and cannot be dominated by any viable distance vector. If all distance vectors t' dominate s', with $t' \neq s'$, then the RFD$_c$s discovered for s' are also valid for s.

Referring to *Case 2* of Section 3.4, notice that $s' = (m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_k)$ dominates some distance vectors, and only m_i makes s non-dominating. Thus, for each $j \neq i$, only a distance vector $t'' = (p_1, \ldots, p_{j-1}, p_{j+1} \ldots, p_k)$, dominated by $s'' = (m_1, \ldots, m_{j-1}, m_{j+1}, \ldots, m_k)$, can be used to determine the thresholds a_j. Otherwise the RFD$_c$s generated by the distance vectors representing projections of s_k on a lower number of attributes, will also be valid for it. □

Completeness. The last important evaluation dimension of a dependency discovery algorithm is completeness, which guarantees that it discovers *all* minimal dependencies.

Theorem 3. DOMINO discovers all minimal RFD$_c$s.

Proof: In our proof by contradiction, assume that DOMINO misses a minimal RFD$_c$ $\varrho : X_{1(\leq\alpha_1)}, \ldots, X_{n(\leq\alpha_n)} \rightarrow A_{(\leq\beta)}$ holding on an input instance r. Then, the LHS thresholds of ϱ cannot dominate any distance vector included into T_β. Thus, for each distance vector t in T_β, if we consider $s = (t[X_1], \ldots, t[X_n])$, there exists a threshold α_i of ϱ such that $\alpha_i < m_i = t[X_i]$. Now given an attribute X_i, if we consider the correspondent value m_i for each distance vector in T_β, and DOMINO discovers an RFD$_c$ ϱ' from it, then ϱ' is correct and minimal, and $\alpha'_i \geq \alpha_i$. Moreover, if DOMINO does not determine any RFD$_c$ from m_i, then the properties for determining proper thresholds (Section 3.4) will never be satisfied. However, a correct RFD$_c$ cannot be generated if Properties 2 and 3 are not satisfied. And a minimal RFD$_c$ cannot be generated if Properties 1 and 4 are not satisfied. This contradicts the original assumption. □

5 Empirical Evaluation

We present experimental results concerning the performance of DOMINO in terms of discovered RFD$_c$s and execution time, and compare it with those achieved by the *differential dependency* (DD) and *matching dependency* (MD) discovery algorithms of [9] and [10], respectively.

5.1 Experimental setup

We implemented DOMINO in Java SE 10, using the Levenshtein distance for comparing textual attributes, the absolute difference for numerical attributes, and the alphabetic distance for individual characters. To manage very large distance relations, we used the SortedTableMap data structure of MapDB[2], an open-source, embedded Java database engine and collection framework.

During the data pre-processing phase, DOMINO constructs the distance relation, and linearly links distance vectors having the same thresholds β. This step allows to sort distance vectors according to a candidate RHS attribute in linear time.

DOMINO can be executed in several different modes, according to constraints that users can impose on thresholds. First, if DOMINO is executed unconstrained, it discovers all RFD$_c$s without limits on threshold values. This can lead to the discovery of RFD$_c$s with extremely broad threshold values, which are often meaningless. Thus, users can run DOMINO by specifying upper-bounds on the threshold values of each attribute of the dataset. Alternatively, users can run DOMINO in *log*-mode, which means that the maximum threshold values will be automatically inferred based on the value distribution of each attribute, whereas the variation step (discussed in Section 3.4) follows a logarithmic scale. In particular, for string data we use the weighted mean of string lengths as maximum threshold, and half of the mean square error for other data types. Moreover, the threshold variation step (referred to as ε-step) is set to 1 when β is in the range [0–7] and to $2^i - 1$ otherwise, where $i = 4, \ldots, \log_2(maxThr)$, and $maxThr$ is the maximum distance threshold. For float attributes with a small difference between the minimum and the maximum values the threshold variation step is set to 0.1 when β is in the range [0–7], and to $(2^i - 1)/10$ otherwise, with $i = 4, \ldots, \log_2(maxThr)$. Notice that the *log*-mode generates coarser threshold ranges for higher distance values: as the threshold increases, the fact that the distance between two attribute values falls within it tends to be less relevant. However, when β is in the range [0–7] we maintain a unary step, because the values in such range represent distances close to equality.

The experiments were performed on a machine with an Intel Xeon W 3.2GHz 8-core CPU with 64 GB of memory, running macOS High Sierra and a 64-Bit Java environment. We evaluated DOMINO on 13 real-world datasets, previously used for testing FD and RFD discovery algorithms [4], [9], [14]. Their statistics are reported in Table 5.

5.2 Discovery performance

Our experimental results are detailed in Table 5. We can observe that the number of discovered minimal RFD$_c$s is not necessarily greater than the number of FDs. In fact, given an FD $X \rightarrow Y$ there might be more tuples similar on X than those that are equal on X, but they might not be all similar on Y. Moreover, the possibly new RFD$_c$s induced by similarity might make previous FDs no longer minimal, hence they would be pruned. This effect depends on the characteristics of attribute domains, whereby with textual attributes there might be more dependencies based on similarity than equality.

Concerning time performances, we report both the preprocessing time to construct distance relations, and the time to discover RFD$_c$s. We can observe that the latter tends to

2. http://www.mapdb.org/

TABLE 5
Statistics and discovery results on the considered datasets.

Datasets	Statistics				#RFD$_c$s	Pre-proc. Time (sec.)	Discovery Time (sec.)	Memory Cons. (MB)
	#Columns	#Rows	#FD	Size (KB)				
Foodstamp	5	150	7	3	9	0.60	0.03	111
Iris	5	150	4	5	19	0.06	0.55	104
Balance-scale	5	625	1	7	1	0.31	0.13	94
Emissions	5	8,088	10	479	23	20.20	0.56	128
Vocab	5	21,638	4	530	6	116.28	0.03	128
Car-data	7	1,728	1	53	2	2.92	0.69	129
Chess	7	28,056	1	519	1	266.06	0.60	145
Cars	9	406	67	23	2,490	0.32	171.00	761
Abalone	9	4,177	187	137	3	9.09	0.81	332
Glass	11	214	170	12	1,462	0.17	77.00	510
Breast-Cancer	11	699	46	20	2,169	4.79	78.00	1,310
Bridges	13	108	142	6	6,487	0.16	307.00	1,279
Echocardiogram	13	132	538	6	7,333	0.55	639.00	1,282
Fake Accounts	15	3313	139	314	7,832	24.00	4,451.00	1,106

be related to the number of columns and the number of RFD$_c$s. However, it also depends on the characteristics of the analyzed dataset. For example, numerical attributes with close values produce many distance vectors, whereas textual attributes usually produce fewer.

As a further experiment, we ran DOMINO on the CiteSeer[3] dataset, evaluating the RFD$_c$s discovery performances by incrementally changing the size and the dimensionality of the dataset.

CiteSeer collects records of scientific papers including the following attributes: title, authors, year, description, id, subject, and publisher. We performed the experiments focusing on a sample R of $10,000$ rows of the entire dataset, by first focusing on different subsets of columns, and then on different subsets of rows. In particular, we considered several relations R_i, with $i = 1, \ldots, 5$, each derived by the projection of the first $i+1$ attributes of the whole relation R. We also created different instances I_j of R, with $j = 1, \ldots, 9$, where I_j contained the first $j \times 1,000$ rows, and finally, the overall relation R, which corresponds to R_6 and I_{10}. We have run several experimental sessions on each of these relation instances, by varying the ε-step, and evaluating the impact on the number of discovered RFD$_c$s and execution performance. In particular, we considered ε-steps of 1, 2, 3, and *log*, fixing $maxThr = 15$.

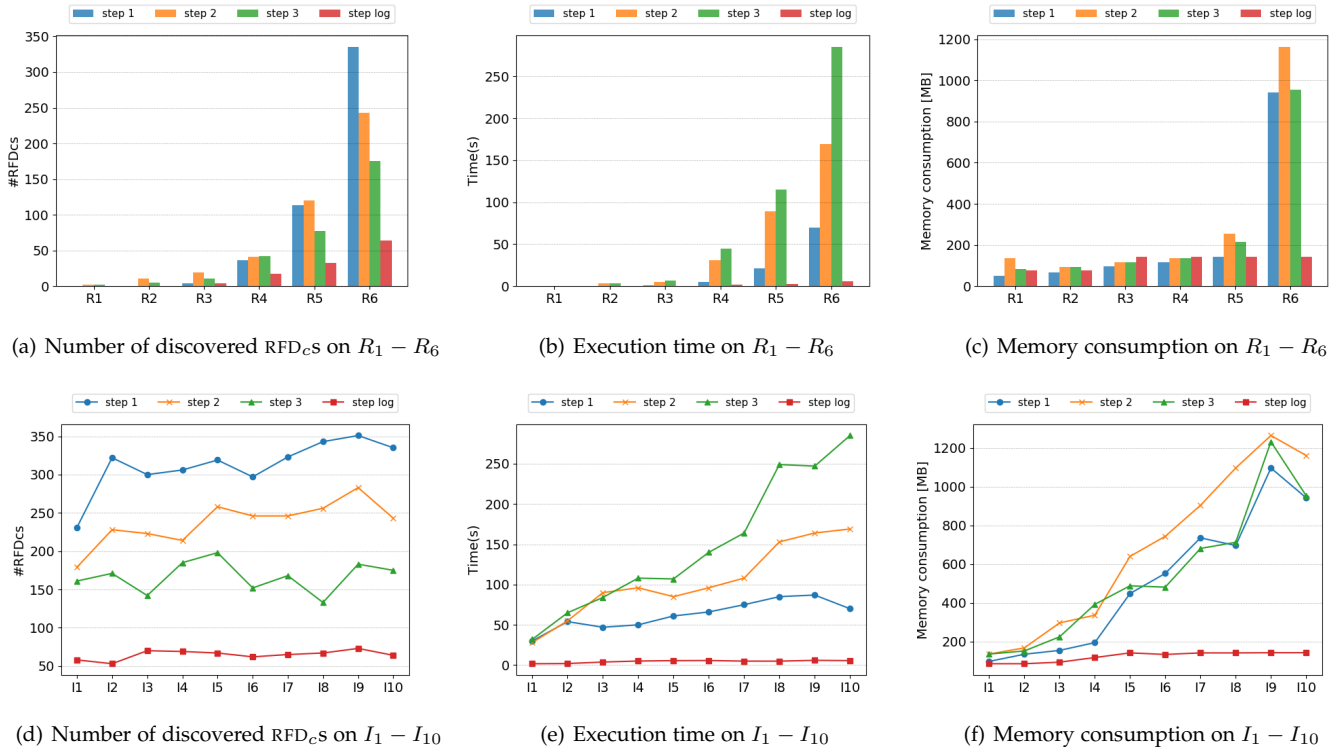
Figures 4(a), 4(b), and 4(c) show results of DOMINO on relations $R_1, \ldots, R_6$. As expected, the number of discovered RFD$_c$s generally increases when the ε-step is lower, since coarser ε-steps imply fewer candidate thresholds. Thus, for an RFD$_c$ discovered with a coarser ε-step there might be more discovered RFD$_c$s than with finer ε-steps. Concerning time performance, we can observe that the *log* step performs considerably better even though the number of steps is greater than Step 3. This improvement is due to the fact that DOMINO makes more attempts with narrow thresholds, which yield a lower number of RFD$_c$s. Memory consumption is almost similar from R_1 to R_4 for all analyzed ε-steps, but also in this case the *log* step achieves considerably better performance as the number of columns increases.

Figures 4(d), 4(e), and 4(f) show the performances of DOMINO on instances $I_1, \ldots, I_{10}$ of R_6. Notice that the introduction of new rows might yield local variations in the number of discovered RFD$_c$s. This is due to the fact that the added rows might either violate RFD$_c$s discovered on smaller relation instances or introduce new ones, by producing a non-monotonic trend on the number of discovered RFD$_c$s. Concerning execution time, as expected, it increases with the size of the instance. In particular, it increases faster for Step 3, because the insertion of new rows might potentially require the evaluation of more feasible patterns due to the intra-step variability. The trend to increase holds also for the memory consumption, even if it is not proportional to the step granularity, as shown in Fig. 4(c) and Fig. 4(f). This might be due to the fact that the task entailing more memory consumption is the one for constructing T_β relations (see Section 3.2). Thus, with a wider step, the number of candidate distance vectors to be added to T_β at each stage is higher, but this also increases the possibility of pruning distance vectors based on the dominance property. In Fig. 4(c) and Fig. 4(f), Step 2 often has the highest memory consumption, possibly because the vectors added to T_β at each stage are more numerous than those of Step 1, but less dominant than those of Step 3. In general, best time and memory performances are achieved by the *log*-step.

The experiments on the CiteSeer dataset also showed how the threshold values and LHS cardinalities of discovered RFD$_c$s change when varying the minimum requested UC value. The results obtained with *maxThr* $= 4$ and Step 1 are shown in Fig. 5, where each line represents the RFD$_c$s with a given attribute as RHS, provided that they yield at least one RFD$_c$ without the id attribute. The figure shows how the average LHS threshold values (Fig. 5(a)), the average RHS threshold values (Fig. 5(b)), and the average LHS cardinalities (Fig. 5(c)) change when varying the minimum requested UC value for discovered RFD$_c$s.

As expected, when the minimum requested UC value grows, the LHS thresholds tend to increase, whereas the RHS threshold and the LHS cardinality tend to decrease. Notice that the average LHS threshold value has been evaluated by considering a value greater than *maxThr* for attributes not appearing in the RFD$_c$. In this way, LHSs with fewer attributes gain higher average threshold values, which is consistent with the minimality criterion. The only exception for RHS thresholds occurs for Authors and Title attributes. In particular, the RFD$_c$s with UC values in the range [0.8–0.9] have LHS thresholds considerably higher

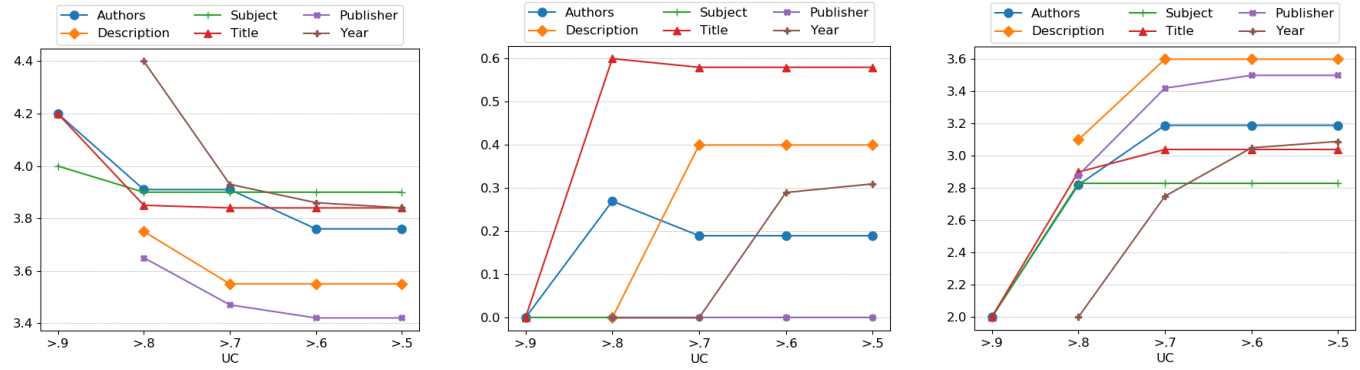
3. http://csxstatic.ist.psu.edu/downloads/data



(a) Number of discovered RFD$_c$s on $R_1 - R_6$ (b) Execution time on $R_1 - R_6$ (c) Memory consumption on $R_1 - R_6$

(d) Number of discovered RFD$_c$s on $I_1 - I_{10}$ (e) Execution time on $I_1 - I_{10}$ (f) Memory consumption on $I_1 - I_{10}$

Fig. 4. DOMINO results on the CiteSeer dataset.



(a) UC vs LHS thresholds (b) UC vs RHS thresholds (c) UC vs LHS cardinality

Fig. 5. Utility Control ranking results w.r.t. LHS thresholds, RHS thresholds, and LHS cardinality.

and LHS cardinality considerably lower than those in the range [0.7–8.0], so admitting some RFD$_c$s with a higher threshold on the RHS. For this reason the UC function has assigned a high value to these RFD$_c$s.

5.3 Qualitative analysis of DOMINO results

We performed a qualitative analysis on the results achieved by DOMINO on three datasets: Bridges, Cars, and Fake Accounts. In particular, we ran DOMINO by setting $maxThr = 4$, yielding only RFD$_c$s with low distance thresholds. Table 6 shows three examples of RFD$_c$s extracted from each considered dataset. We have focused only on RFD$_c$s with non-zero distance thresholds on at least one of the two sides.

By examining Table 6, the RFD$_c$s from the Bridges dataset reveal that bridges built in close years and with a similar length have same span, type, and number of lanes. Given that the dataset contains data of bridges whose construc-

TABLE 6
Some meaningful RFD$_c$s discovered by DOMINO.

Bridges
Erected$_{(\leq 2)}$,Length$_{(\leq 10)}$ → Span$_{(\leq 0)}$
Erected$_{(\leq 2)}$,Length$_{(\leq 0)}$ → Type$_{(\leq 0)}$
Erected$_{(\leq 1)}$,Length$_{(\leq 40)}$ → Lanes$_{(\leq 0)}$
Cars
MPG$_{(\leq 2)}$,Displacement$_{(\leq 0)}$ → Cylinders$_{(\leq 0)}$
Displacement$_{(\leq 4)}$,Weight$_{(\leq 20)}$, Acceleration$_{(\leq 0)}$,Model$_{(\leq 0)}$ → MPG$_{(\leq 2)}$
MPG$_{(\leq 0)}$,Horsepower$_{(\leq 1)}$, Weight$_{(\leq 10)}$,Acceleration$_{(\leq 2)}$ → Displacement$_{(\leq 0)}$
Fake Accounts
CreatedAt$_{(\leq 1)}$ → ListCount$_{(\leq 1)}$
ListCount$_{(\leq 2)}$,CreatedAt$_{(\leq 2)}$ → DefaultProfile$_{(\leq 0)}$
FriendCount$_{(\leq 0)}$,CreatedAt$_{(\leq 3)}$, ListCount$_{(\leq 0)}$ → FavouriteCount$_{(\leq 2)}$

tion period spans over many years, these RFD$_c$s probably reflect the fact that in close periods, and for bridges with a similar length, engineers either abideed by some standard construction norms, or they merely followed some common construction trend of that period.

The RFD$_c$s for the dataset Cars reveal that cars with similar characteristics, such as horsepower, weight, acceleration, and so on, also have similar consumption (MPG), same number of cylinders, and same displacement.

Finally, the RFD$_c$s extracted from the fake accounts dataset let us figure out typical behavior of BOTs automatically generating such accounts. In particular, they reveal that accounts created by BOTs having close creation time ($\leq$ 1) also have a similar number of subscribed lists (ListCount$_{(\leq 1)}$). Those having a slightly less similar creation time ($\leq$ 2) and ListCount$_{(\leq 2)}$, also have the same settings for the image profile (DefaultProfile), whereas those having the same number of followed users (FriendCount), the same ListCount, and similar creation time ($\leq$ 3), also have a similar number of liked tweets (FavouriteCount).

5.4 Discovering FDs with DOMINO in noise datasets

As a further experiment, we have tested the capability of DOMINO to recover FDs lost due to errors in the data. In particular, we have used the Bridges dataset, on which 142 FDs were discovered by TANE [6], an FD discovery algorithm, also capable of discovering RFDs relaxing on the extent based on the $g3$-error coverage measure. At each step, we have introduced some noise on a single attribute, involving between 10% and 50% of tuples. We have accomplished this on 5 out of 13 attributes of the dataset: two chars (River, Clear-G), two strings (Purpose, T-or-D), and one numerical Erected attribute. We altered their values by changing the character for chars, inserting a character for strings, and increasing or decreasing by 1 the numerical attributes.

Table 7 reports the results of such experiment, where first column reports the name of the altered attribute; second column the percentage of tuples for which the value of the given attribute was altered; third and fourth columns report the number of FDs discovered by TANE and of RFD$_c$s discovered by DOMINO, respectively, after the perturbation; fifth column reports how many of the original 142 FDs are not discovered by TANE after introducing the perturbation, whereas last column reports how many of these lost FDs are recovered by DOMINO configuring $maxThr = 1$ only for the perturbed attribute.

The results show that DOMINO has recovered most of the lost FDs but for the Erected attribute, which is the only numerical one. This is due to the fact that when altering a numerical attribute A for a tuple t_1, if $t_1[A]$ was equal to $t_2[A]$, it might happen that after the perturbation it becomes more similar to $t_3[A]$ than to $t_2[A]$. Thus, after the perturbation, previously holding FDs might no longer be discovered and new RFD$_c$s might arise due to the different clustering of tuples induced by the similarity. For string attributes it is more unlikely that small perturbations yield new similarities, even though this depends on the characteristics of the dataset at hand. To this end, second column tells us that an RFD discovery algorithm relaxing on the extent using the same percentage of error than the introduced noise would recover all the lost FDs. However, there are attributes for which the error is not limited to a percentage of tuples, like for instance an address attribute, for which there might be small differences spread across the whole dataset. In these cases, it is better using an RFD$_c$ discovery algorithm than one relaxing on the extent. Obviously, hybrid RFDs would be preferable, but their discovery is NP-complete [9], [10]. Moreover, we have noticed that the latter discover many more RFDs than the FDs previously holding on the dataset, which makes it more complex to understand the relevance of each discovered RFD.

TABLE 7
DOMINO's discovery performances upon introducing noise on the Bridges dataset.

	%noise	#FDs	#RFD$_c$s	#lost by TANE	#recovered by DOMINO
River	10%	153	166	1	0
	20%	164	175	2	1
	30%	166	172	0	0
	40%	175	175	1	0
	50%	190	204	2	1
Purpose	10%	131	166	8	8
	20%	130	171	5	4
	30%	132	168	8	7
	40%	145	170	8	6
	50%	155	198	11	11
T-or-D	10%	136	167	0	0
	20%	129	190	4	3
	30%	140	187	0	0
	40%	145	181	0	0
	50%	166	206	1	1
Erected	10%	142	164	0	0
	20%	160	190	0	0
	30%	159	194	0	0
	40%	151	189	3	0
	50%	152	190	6	0
Clear-G	10%	128	167	18	18
	20%	149	161	2	2
	30%	145	172	14	14
	40%	155	178	12	12
	50%	186	186	11	11

5.5 Comparison with DD and MD discovery algorithms

We performed a comparative evaluation of DOMINO with respect to the DD [9] and MD [10] discovery algorithms. They both represent a broad subclass of RFD$_c$s, and thus DOMINO can be easily configured to discover them. We re-implemented all the DD and MD algorithm variants for which pseudo-code was clearly described in [9], [10], respectively.

As in [9], we built ten different relation instances, starting with a subset of 2,000 rows, increasing them at each step by 2,000, until reaching the size of 20,000 rows, from which we produced five further relations by removing for each step the last column. Moreover, because the DD discovery algorithms require the specification of maximum thresholds, we ran DOMINO on these instances by setting $maxThr = 4$, i.e., for each attribute we considered five possible thresholds in the range [0–4], as in [9].

Among the different versions of the DD discovery algorithm presented in [9], we compared DOMINO 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 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 aims to validate DDs throughout the search space, which can be pruned according to the specific methodology; the second phase aims to test dependency minimality by removing all discovered DDs that can be inferred by other ones.

In contrast, the discovery algorithm presented in [10] takes as input the LHS and RHS attributes of an MD, together with the RHS threshold, and finds the LHS thresholds by computing statistical tuples of similarity thresholds from the given dataset. Moreover, the solution considers as input some thresholds to limit the search space, according to support and confidence quality measures. In order to compare DOMINO with such a solution, we implemented a complete MD discovery process by integrating the algorithm in [10] within a module considering all possible column combinations. As for threshold values, we mapped the DOMINO's distance range [0–4] to the MD's similarity range [6–10]. In this way, we were able to compare the achieved results with those of DOMINO by executing the MD algorithm with five RHS similarity thresholds in the range [0.6-1]. Moreover, we used threshold 0 as support and threshold 1 as confidence. Finally, we added a post-processing module to select only minimal MDs.

Among the different versions of the MD discovery algorithm presented in [10], we compared DOMINO with the following ones: (i) exact-algorithm (EA), which considers all possible statistical tuples according to a fixed search space; (ii) pruning by support (EPS), using a directed acyclic graph (DAG) to represent candidate similarity patterns, and a pruning rule based on the support quality measure; and (iii) pruning by support and confidence (EPSC), adding another pruning rule based on the confidence quality measure.

We implemented the above versions of both approaches described in [9], [10] in Java SE 10. Moreover, as BU-Ne and IE-TD-Po versions for DD discovery use recursion to split the search space, by removing some branches of the recursion tree based on the specific pruning strategy, we used the Java functional programming technique *tail recursion*, to guarantee the stack safety of recursion calls. All implementations and datasets used for our comparisons are available online[4].

Figure 6 shows time and memory performances achieved by DOMINO and the above mentioned versions of DD and MD discovery algorithms. By analyzing Figure 6(a), we notice a more explosive growth of the execution time for DD and MD discovery algorithms with respect to DOMINO when increasing the number of columns. In particular, with respect to the best DD discovery algorithm (IE-TD-Po), DOMINO reduces the execution time by two orders of magnitude on R_6. Concerning memory performance on $R_1 - R_6$ (Figure 6(b)), we notice that even though DOMINO starts with a higher memory consumption with respect to DD versions, it remains almost constant and outperforms DD algorithms as the number of columns increases. Moreover, even if the memory consumption remains almost constant also for the MD versions, DOMINO always requires over one order of magnitude less memory.

We used R_5 to compare time performances on several relation instances, since on each instance of R_6 the BU-Ne version would run for several days. In general, we notice that by varying the number of rows, time performance trends are not exponential for all the analyzed discovery algorithms. In particular, DOMINO achieves better time performances for all relation instances, as shown in Figure 6(c). Also in terms of memory performance, DOMINO achieves best results, as shown in Figure 6(d).

5.6 Time and memory requirements of DOMINO

As a final evaluation, we performed stress tests on DOMINO aiming to determine time and memory limits of DOMINO's. To this end, we focused on the Adult dataset, since it contains a suitable mix of textual and numerical attributes, and has a sufficiently high number of columns to stress the performances of DOMINO towards 24 hours of execution time and 30GB of memory limits.

Table 8 shows the results of our experiments by varying the number of columns, rows, and *maxThr*, where *log* indicates the execution without threshold limit and in *log*-mode. The values TL and ML represent the cases in which DOMINO exceeded the above mentioned time or memory limits. Notice that, since DOMINO relies on MapDB, which maps the main memory on secondary storage, it will continue running even when reaching the memory limit.

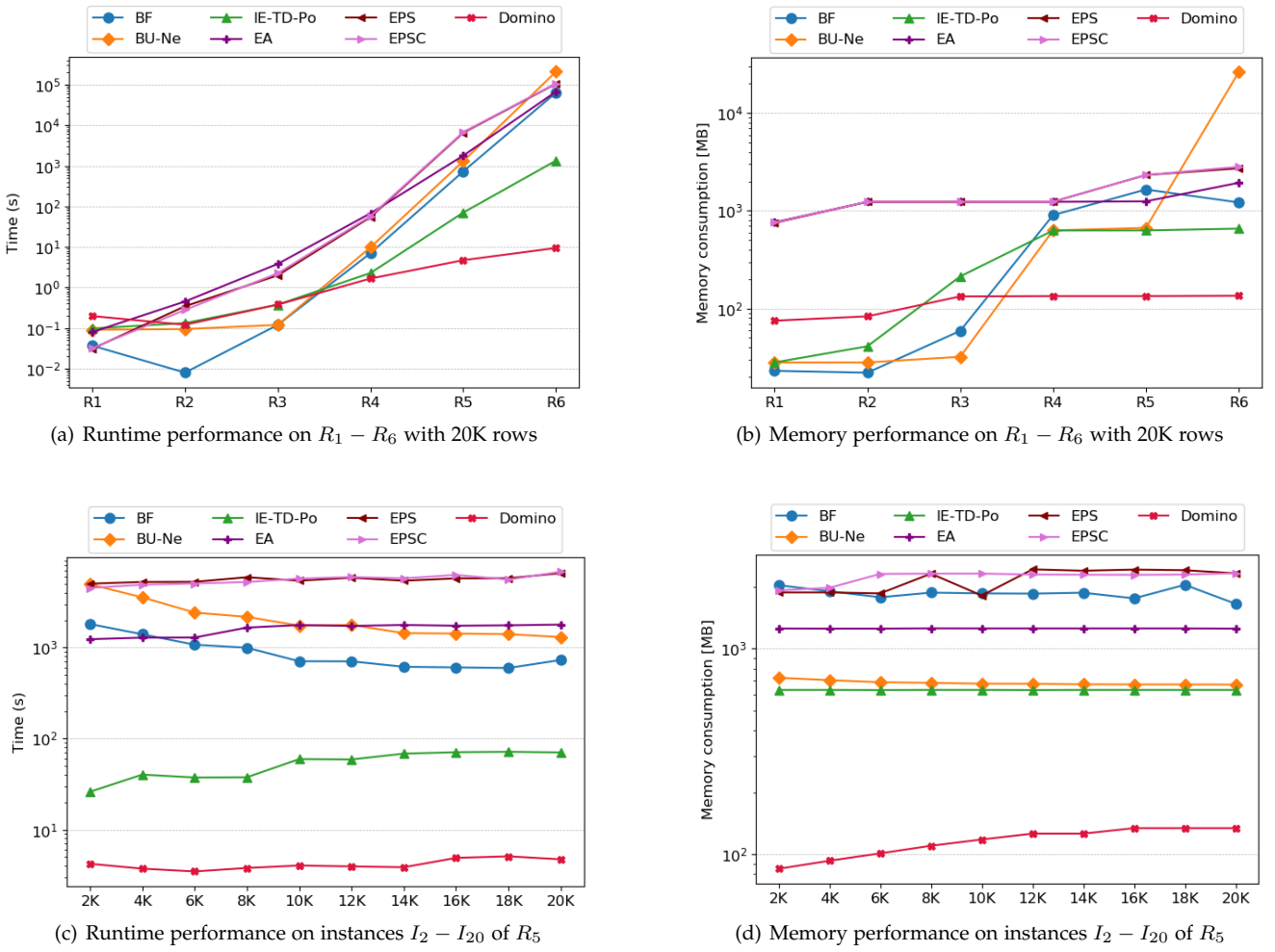
TABLE 8
Discovery results on the Adult datasets.

#Cols	#Rows	#maxThr	#RFD$_c$s	Time (s)	Mem. [MB]
10	32,561	log	2	13.00	1,491
11	32,561	log	2	485.00	6,759
12	32,561	log	2	1,666.00	10,570
13	32,561	log	13	21,745.00	20,327
14	32,561	log	16	57,902.00	15,820
15	32,561	log	-	TL	ML
15	2,561	log	5,267	4,167.00	4,414
15	7,561	log	1624	12,632.00	6,818
15	12,561	log	649	31,334.00	23,561
15	15,561	log	169	63,012.00	23,552
15	22,561	log	116	83,665.00	ML
15	27,561	log	76	TL	ML
15	32,561	log	-	TL	ML
15	32,561	1	28	96.00	2,581
15	32,561	2	97	698.00	3,355
15	32,561	3	288	2,868.00	5,038
15	32,561	4	634	15,171.00	7,986
15	32,561	5	1712	TL	18,226
15	32,561	log	-	TL	ML

TL: time limit of 24 hours exceeded
ML: memory limit of 30 GB exceeded

DOMINO exceeds the time limit of 24h when using 15 columns and more than 32,561 rows. Similar observations are true for the memory limit, even though this is exceeded already with more than 15,561 rows. Concerning *maxThr*, we notice that although for $maxThr = 5$ there is low memory consumption, DOMINO exceeds the time limit. If we lower *maxThr* to 4, both time and memory performances considerably improve.

4. https://dast-unisa.github.io/Domino-SW/



(a) Runtime performance on $R_1 - R_6$ with 20K rows

(b) Memory performance on $R_1 - R_6$ with 20K rows

(c) Runtime performance on instances $I_2 - I_{20}$ of R_5

(d) Memory performance on instances $I_2 - I_{20}$ of R_5

Fig. 6. Runtime and memory performance of BF, BU-Ne, IE-TD-Po, EA, EPS, EPSC, and DOMINO on the CiteSeer dataset with *maxThr* $= 4$.

6 Related Work

Methods for the automatic discovery of FDs belong to two main categories: top-down or bottom-up candidate selection. The formers start generating candidate FDs based on an attribute lattice, validate them, and then use valid FDs to prune the search space for candidate FDs yet to be verified. The bottom-up methods compare attribute values for each pair of tuples, in order to generate two different sets of attribute subsets from which candidate FDs are derived.

In top-down approaches we find TANE [6], FD_Mine [15], FUN [16], and DFD [17], whereas in bottom-up ones we have DepMiner [18], FastFD [19], and FDep [20]. Experiments show that top-down algorithms tend to perform better on datasets with many rows and few columns, whereas bottom-up algorithms perform better with few rows and many columns [14]. The fastest algorithms on a reasonable mix of rows (up to 300K) and columns are DFD (up to 25 columns), TANE (from 25 to 65 columns), and FDep (over 65 columns). Finally, hybrid algorithms combine row-efficient and column-efficient discovery techniques. In particular, the algorithm described in [21] first calculates FDs on a small subset of randomly selected records, and validating them on the entire dataset. This algorithm scales better than others when the number of rows and the size of the result set increase. The DHyFD algorithm proposed in [22] introduces a new data structure, namely extended FD-trees, and a dynamic data manager (DDM), following the column-based approach over extended FD-trees, and switching to a row-based technique when many FDs are supposed to be valid.

In general, FD discovery algorithms achieve good time and space performance, thanks to the disjointness property of equivalence classes. This property is still preserved for RFDs relaxing on the extent, such as approximate functional dependencies (AFDs), for which several efficient discovery algorithms have been provided [6], [23]. Instead, in the context of RFD$_c$s, the use of distance functions yields intersecting classes, due to the non-transitiveness of distance functions, which considerably increases the computational complexity. In the last few decades, there has been a proliferation of new RFDs definitions [12]. Some of them relax on the tuple comparison or the extent only, whereas few of them relax on both. However, many of the RFD definitions in the literature are not equipped with algorithms for their discovery from data [7], [24], especially the hybrid ones, since their discovery has been proven to be NP-complete [9], [10]. To this end, some recent work proposes approximate discovery algorithms for hybrid RFDs in order to tackle NP-completeness [25], [26]. In what follows we review the RFD$_c$s

for which a discovery algorithm has indeed been provided.

MDs were proposed for object identification and defined in terms of similarity predicates to accommodate errors and different representations in unreliable data sources [11]. The discovery algorithms presented in [10] evaluate the utility of MDs in a given database instance, determining the threshold pattern for MDs. Utility is measured through confidence and support of MDs, while thresholds are determined based on the statistical data distribution. Pruning strategies are introduced to filter out candidate patterns with low support. With respect to these algorithms, DOMINO is more general, as it discovers all minimal RFD_cs without requesting the specification of the RHS. Moreover, DOMINO embeds techniques to derive minimal RFD_cs, which is considered an open issue in [10].

DDs specify constraints on attribute value differences instead of exact matches of canonical FDs [9]. Their discovery from data inherits the exponential complexity from the FD discovery problem. The discovery algorithm proposed in [9] is based on reduction algorithms: once fixed the RHS differential function for each attribute in r, the set of left-reduced LHS differential functions are found to form DDs. The pruning strategies proposed to improve discovery performances are based on the subsumption order of differential functions, the implication of DDs, and the instance exclusion. The approach for DD discovery proposed in [27] reduces the search space of the problem by assuming a user-specified distance threshold as upper limit for the distance intervals of LHSs. The discovery algorithm is based on a distance-based subspace clustering model, and includes further pruning strategies to enable efficient discovery of DDs for high threshold values. The algorithm proposed in [28] mines a minimal cover of DDs based on association rules. In particular, the approach applies the algorithm presented in [29] for mining a class of non-redundant association rules, which are then transformed into DDs. The latter are pruned in order to generate a minimal cover of interesting DDs. With respect to DD discovery algorithms, DOMINO is able to discover RFD_cs without the specification of user specified thresholds, hence facing more a complex problem.

7 Conclusion and Future Work

This paper proposes DOMINO, a novel discovery algorithm for RFD_cs, which relies on the concept of dominance from the multi-attribute decision theory, accomplishing the discovery process without requesting the specification of input thresholds. We proved that DOMINO discovers the complete set of minimal and correct RFD_cs. Although the complexity of RFD_c discovery is exponential in the number of attributes and the corresponding distance functions, experimental results show that DOMINO achieves good discovery and time performance on real-world datasets.

In the future, we plan to investigate further pruning strategies for the minimality and RFD_c generation phases, as well as heuristics to make the discovery process more efficient, even admitting approximate results. Moreover, we would like to investigate the possibility of extending DOMINO to discover hybrid RFDs, hence capturing RFD_cs admitting some exceptions. To this end, we cannot simply adopt common coverage measure validation strategies from *approximate functional dependencies* [30], because the strategy used by DOMINO to automatically infer thresholds relies on candidate threshold value generation instead of candidate attribute set generation. We plan to produce a version of DOMINO for distributed architectures in order to further improve time performances. Finally, it could be interesting to investigate the possibility of developing an incremental version of DOMINO, in order to update the set of discovered RFD_cs as the dataset instance evolves, without having to recompute it from scratch [31].

References

[1] W. Fan, F. Geerts, S. Ma, N. Tang, and W. Yu, "Data quality problems beyond consistency and deduplication," in *In Search of Elegance in the Theory and Practice of Computation: Essays Dedicated to Peter Buneman*, V. Tannen *et al.*, Eds. Springer, 2013, pp. 237–249.

[2] U. Nambiar and S. Kambhampati, "Mining approximate functional dependencies and concept similarities to answer imprecise queries," in *Proc. Workshop on Web and Databases (WebDB)*, 2004, pp. 73–78.

[3] W. Fan, F. Geerts, and X. Jia, "A revival of integrity constraints for data cleaning," *PVLDB*, vol. 1, no. 2, pp. 1522–1523, 2008.

[4] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "Fame for sale: Efficient detection of fake Twitter followers," *Decision Support Systems*, vol. 80, pp. 56–71, 2015.

[5] L. Caruccio, D. Desiato, and G. Polese, "Fake account identification in social networks," in *Proc. of the International Conference on Big Data (BigData)*, 2018, pp. 5078–5085.

[6] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," *The Computer J.*, vol. 42, no. 2, pp. 100–111, 1999.

[7] J. Liu, J. Li, C. Liu, and Y. Chen, "Discover dependencies from data - A review," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 2, pp. 251–264, 2012.

[8] J. Zhang and P. Pu, "Survey of solving multi-attribute decision problems," Swiss Federal Institute of Technology, Lausanne, Switzerland, Tech. Rep., 2004.

[9] S. Song and L. Chen, "Differential dependencies: Reasoning and discovery," *ACM Transactions on Database Systems (TODS)*, vol. 36, pp. 16:1–16:41, 2011.

[10] ——, "Efficient discovery of similarity constraints for matching dependencies," *Data and Knowledge Engineering (DKE)*, vol. 87, pp. 146–166, 2013.

[11] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma, "Dynamic constraints for record matching," *VLDB Journal*, vol. 20, pp. 495–520, 2011.

[12] L. Caruccio, V. Deufemia, and G. Polese, "Relaxed functional dependencies – A survey of approaches," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 147–165, 2016.

[13] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian, "Metric functional dependencies," in *Proc. of the International Conference on Data Engineering (ICDE)*, 2009, pp. 1275–1278.

[14] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann, "Functional dependency discovery: An experimental evaluation of seven algorithms," *PVLDB*, vol. 8, no. 10, pp. 1082–1093, 2015.

[15] H. Yao, H. J. Hamilton, and C. J. Butz, "FD_Mine: Discovering functional dependencies in a database using equivalences," in *Proc. Int. Conf. on Data Mining*, ser. ICDM, 2002, pp. 729–732.

[16] N. Novelli and R. Cicchetti, "FUN: An efficient algorithm for mining functional and embedded dependencies," in *Proc. Int. Conf. on Database Theory (ICDT)*, 2001, pp. 189–203.

[17] Z. Abedjan, P. Schulze, and F. Naumann, "DFD: Efficient functional dependency discovery," in *Proc. Int. Conf. on Information and Knowl. Management (CIKM)*, 2014, pp. 949–958.

[18] S. Lopes, J.-M. Petit, and L. Lakhal, "Efficient discovery of functional dependencies and Armstrong relations," in *Proc. Int. Conf. on Extending Database Technology*, 2000, pp. 350–364.

[19] C. Wyss, C. Giannella, and E. Robertson, "FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances," in *Proc. Int. Conf. on Data Warehousing and Knowl. Discovery (DaWaK)*, 2001, pp. 101–110.

[20] P. A. Flach and I. Savnik, "Database dependency discovery: A machine learning approach," *AI Communication*, vol. 12, no. 3, pp. 139–160, 1999.

[21] T. Papenbrock and F. Naumann, "A hybrid approach to functional dependency discovery," in *Proc. Int. Conf. on Management of Data (SIGMOD)*, 2016, pp. 821–833.
[22] Z. Wei and S. Link, "Discovery and ranking of functional dependencies," in *Proc. of the International Conference on Data Engineering (ICDE)*, 2019, pp. 1526–1537.
[23] P. Mandros, M. Boley, and J. Vreeken, "Discovering reliable approximate functional dependencies," 2017, pp. 355–363.
[24] S. Kruse and F. Naumann, "Efficient discovery of approximate dependencies," *PVLDB*, vol. 11, no. 7, pp. 759–772, 2018.
[25] Y. Wang, S. Song, L. Chen, J. X. Yu, and H. Cheng, "Discovering conditional matching rules," *ACM Trans. Knowl. Discov. Data*, vol. 11, no. 4, pp. 46:1–46:38, 2017.
[26] L. Caruccio, V. Deufemia, and G. Polese, "Evolutionary mining of relaxed dependencies from big data collections," in *Proc. Int. Conf. on Web Intelligence, Mining and Semantics*, ser. WIMS, 2017, pp. 5:1–5:10.
[27] S. Kwashie, J. Liu, J. Li, and F. Ye, "Mining differential dependencies: A subspace clustering approach," in *Proc. of Australasian Database Conference*, ser. ADC, 2014, pp. 50–61.
[28] ——, "Efficient discovery of differential dependencies through association rules mining," in *Proc. of Australasian Database Conference*, ser. ADC, 2015, pp. 3–15.
[29] J. Li, "On optimal rule discovery," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 4, pp. 460–471, 2006.
[30] J. Kivinen and H. Mannila, "Approximate inference of functional dependencies from relations," *Theor. Comput. Sci.*, vol. 149, no. 1, pp. 129–149, 1995.
[31] P. Schirmer, T. Papenbrock, S. Kruse, F. Naumann, D. Hempfing, T. Mayer, and D. Neuschäfer-Rube, "DynFD: Functional dependency discovery in dynamic datasets," in *Proc. Int. Conf. on Extending Database Technology*, 2019, pp. 253–264.

Felix Naumann studied mathematics, economy, and computer sciences at the University of Technology in Berlin. After receiving the diploma (MA) in 1997, he joined the graduate school "Distributed Information Systems" at Humboldt University of Berlin and successfully defended his PhD thesis in 2000. In 2001 and 2002, he worked at the IBM Almaden Research Center on topics around data integration. From 2003-2006, he was assistant professor for information integration at the Humboldt-University of Berlin. Since 2006, he is the chair for information systems at the Hasso Plattner Institute at the University of Potsdam in Germany.

Loredana Caruccio received from the University of Salerno the MSc (cum laude) in Computer Science in 2012 and the PhD in Management & Information Technology in 2018. In 2017, she has been visiting student at the Hasso Plattner Institute of the University of Potsdam. She is currently post-doctoral researcher at the Department of Computer Science of the University of Salerno. Her research interests include data science and web engineering.

Giuseppe Polese is an associate professor in the University of Salerno. His research interests include visual languages, databases, and web engineering. He has a Laurea Degree (cum laude) in Computer Science at University of Salerno, a MSc in Computer Science from the University of Pittsburgh (USA), and a PhD in applied mathematics and computer science from the University of Naples. From 1989 to 1991, he was a software engineer and project manager at the Italian Airspace Company, Alenia. He has been a consultant for several software firms, including Siemens, Olivetti, and Telecom Italia.

Vincenzo Deufemia received the Laurea Degree (cum laude) and the PhD in Computer Science from the University of Salerno in 1999 and 2003, respectively. He has been a visiting researcher at the Laboratory of Advanced enterprise Information Management Systems (AeIMS) of University of Western Sydney, Australia in 2013. He is currently an associate professor at the Department of Computer Science – University of Salerno. He has served as a program committee member for several international conferences and workshops. His research interests include data science, pattern recognition, and intelligent user interfaces.

APPENDIX
DETAILED PSEUDO-CODE OF DOMINO

In what follows, we present the detailed pseudo-code of DOMINO's three main steps, which are executed for each candidate RHS attribute.

Algorithm 5 shows the construction of the $T_\beta(A)$ relations for a given attribute A. Starting from the maximum value of β for A, all other values of β will be considered in descending order down to 0 (Line 3). Lines 11–19 are the core of the algorithm. A given distance vector w is added to $T_\beta(A)$ only if it does not dominate all the other distance vectors w' already in $T_\beta(A)$. It is worth noting that Lines 13–14 verify whether w' dominates w, i.e., whether the new checked distance vector w is dominated by another pattern w' previously added to the considered $T_\beta(A)$ relation. If this is the case, since a $T_\beta(A)$ relation must contain only non-dominating distance vectors, w' is removed (Line 14) from $T_\beta(A)$. Finally, Lines 4–6 initialize the current T_β when the β value changes.

Algorithm 5: Construction of T_β relations

Input : An ordered difference relation $\delta(r)$
Output: Set of T_β relations

```
1  β ← getMaxBeta();
2  T_β ← ∅; dominates ← false;
3  foreach w ∈ δ(r) in descending order wrt. the order of δ(r) do
4      if getBeta(w) ≠ β then
5          β ← getBeta(w');
           // T_β is put into T
6          if T_β ≠ ∅ then addSet(T, T_β);
7      if T_β = ∅ then
           // First distance vector
8          add(T_β, w);
9      else
10         dominates ← false;
11         foreach w' ∈ T_β do
12             if !dominance(w, w') then
13                 if dominance(w', w) then
                       // w' dominates w
14                     remove(T_β, w');
15             else
16                 dominates ← true;
17                 break;
18         if !dominates then
19             add(T_β, w);
20 return T;
```

The second step of the DOMINO's discovery process considers the role of each distance vector in each $T_\beta(A)$ it belongs to, by determining its minimal LHS candidates, from which all possible valid RFD_cs can be generated (i.e., by determining the proper threshold values). The pseudo-code of this phase is shown in Algorithm 6, where a combinatorial level-wise search is performed to consider all feasible LHSs of a given distance vector $w \in T_\beta(A)$. In particular, after the creation of the difference matrix DM (Line 2),

Algorithm 6: Generation of LHS candidates & Determination of thresholds for minimal RFD_cs

Input : A T_β relation from Algorithm 5
A w distance vector
Current set RFDcs of discovered RFD_cs
Output: Set of discovered RFDcs for T_β

```
1  Level ← 1; levelAttr ← ∅; LHS ← ∅;
   // Creation of the Matrix DM
2  DM = createDifferenceMatrix(w);
   // Initialization phase
3  foreach row r ∈ DM do
4      foreach column j of r do
5          add(levelAttr, j);
6      foreach value e in column j of r do
7          e ← getElement(DM, j);
8          if e = 0 then
9              add(columnEqualSet [j], e);
10         else if e < 0 then
11             add(columnSet [j], e);
   // Search phase
12 while Level ≤ numAttr do
13     LHS ← validate&Prune(levelAttr,
14     columnSet, columnEqualSet);
15     Level ← Level +1;
16     levelAttr ←superset(levelAttr);
17     if size(levelAttr) = 0 then
18         break;
19 return generateRFDcs(LHS, T_β, w,
   RFDcs);
```

there is an *initialization phase* (Lines 3–11) where distance vector values are added to sets associated to each column (values less than zero are added to columnSet, see Line 11, whereas values equal to zero are added to columnEqualSet, see Line 9). Such column sets are used during the validation and pruning steps of the search phase, which are accomplished according to the method presented in Section 3.3. In particular, if n is the number of columns, the *search phase* (Lines 12–18) validates and prunes column combinations according to the considered lattice level, the columnSet, and the columnEqualSet (starting from level 1 of the lattice, corresponding to singleton LHSs, and ending to level n), yielding the generation of feasible LHSs. Furthermore, new column combinations for the next level are generated at Line 16, by considering only column combinations that are not pruned, making the union of the columnSet and the intersection of the columnEqualSet for proper columns involved in new column combinations. When the next level of column combinations is void, the search phase is interrupted (Lines 17–18). Finally, RFD_c generation is invoked (Line 19), whose pseudo-code is shown in Algorithm 7.

The algorithm for generating RFD_cs considers a feasible distance vector w (Line 3), and for each of its attributes m_i verifies the possibility to generate an RFD_c by trying to

Algorithm 7: Determination of thresholds for minimal RFD_cs

Input : A T_β relation
A w distance vector
A set of attributes LHS for w
The Set RFDcs of discovered RFD_cs
Output: Set of discovered RFDcs for T_β

```
1  admissible ← false; superset ← false;
2  foreach column combination cc ∈ LHS do
3      s_k ← getVector(w, cc);
4      superset ← false;
5      foreach column i of s_k do
6          m_i ← getElement(s_k, i);
7          if m_i > 0 then
8              admissible ← true;
9              thrs ← voidThrs(s_k);
10             foreach column j ∈ s_k do
11                 if j = i then
12                     add(thrs,j,m_i −1);
13                 else
                       // Return minimum
                          value as defined
                          in 3.4
14                     minThrs ←
                        findMin(T_β,m_i,s_k,j,thrs);
15                     if minThrs <> −1 then
16                         add(thrs, j,minThrs −1);
17                     else
18                         admissible ← false;
19                         break;
20             if admissible = true then
21                 newRFDc ←
                    createRFDc(thrs, β);
22                 if contains(RFDcs,thrs) then
23                     removeRFDc(RFDcs,thrs);
24                 add(RFDcs,newRFDc);
25                 superset ← true;
26     if superset = true then
           // Return only non-dominating
              vectors
27         super ← getSuperVectors(w,cc);
28         if super <> null then
29             foreach column comb cc′ ∈ super do
30                 add(LHS,cc′);
```

associate a threshold to each column appearing in the given LHS combination. In particular, as defined in Section 3.4, when $m_i > 0$, its threshold is set to $m_i - 1$, for an ε-step of 1 (Line 12), whereas the thresholds for columns $j \neq i$ are set as specified in Section 3.4 (Lines 13–19). A flag named *admissible* is used to detect when the threshold definition process is successful. Thus, when *admissible* is true, a new RFD_c is constructed (Line 20) and added to the set containing previously discovered RFD_cs. However, if an LHS with the same thresholds is already contained in that set, the corresponding RFD_c will be removed, since it could be defined from a previous T_β. Indeed, according to the minimality property of RFD_cs (Section 2.2), we keep the RFD_cs with lower RHS threshold. Moreover, a flag named *superset* is used to verify when at least a new RFD_c is added to the complete set of RFD_cs. Therefore, only when *superset* is true (Line 25), it is possible to discover RFD_cs for supersets of the LHS attributes of the current distance vector w. Thus, all feasible supersets are added to the current LHS set (Lines 27–30).