IndiBits: Incremental Discovery of Relaxed Functional Dependencies using Bitwise Similarity

Bernardo Breve, Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, Giuseppe Polese
Department of Computer Science
University of Salerno
via Giovanni Paolo II, n.132
{bbreve,lcaruccio,scirillo,deufemia,gpolese}@unisa.it

***Abstract*—One of the main challenges in data profiling is to efficiently extract metadata from dynamic information sources, by avoiding the processing of the whole dataset from scratch upon modifications. In this paper, we present INDIBITS, an algorithm for discovering relaxed functional dependencies (RFDs for short), which represent data relationships relying on approximate matching paradigms. INDIBITS is able to dynamically infer and update the RFDs holding on a dataset upon modification operations performed on it. It exploits a binary representation of data similarities, a new validation method, and specific search methods, to dynamically update the set of RFDs, based on previously holding RFDs and the type of modifications performed over data. Experimental results demonstrate the effectiveness of INDIBITS on real-world datasets, even in comparison with FD and RFD discovery algorithms in both static and dynamic scenarios.**

***Index Terms*—Data Profiling, Relaxed Functional Dependencies, Incremental Scenarios, Bitwise Similarities**

I. INTRODUCTION

Data profiling refers to the process aiming to analyze data in order to extract useful metadata from them [1]. Among these metadata, Functional Dependencies (FDs) received a considerable interest from the research community, mainly due to their application in advanced database operations, such as data cleansing, query optimization, and so forth [2], [3]. In the last decade, the definition of FD underwent several extensions, leading to the definition of Relaxed Functional Dependency (RFD) [4], [5], in order to tackle more complex problems, especially in the Big Data context. In particular, extensions have concerned the use of approximate comparisons between attribute values by means of similarity constraints, leading to the definition of RFDs relaxing on the attribute comparison (RFD_cs, for short), or of error measures to tolerate possible violations for a limited number of tuples, leading to the definition of RFDs relaxing on the extent (RFD_es, for short).

As an example, in the dataset snippet of Table I, the values of attributes heartrate and chiefcomplaint seem to be related to those of attribute acuity. However, due to different abbreviations used to represent symptoms for the attribute chiefcomplaint, and the necessity to admit a certain degree of tolerance when considering values of attribute heartrate, it is necessary to use approximate comparison methods to compare values of these attributes, such as the Levenshtein distance function for attribute symptoms and the absolute difference for the attribute heart rates. In particular, we notice that each time two tuples have a distance less than 5 on the chiefcomplaint attribute and less than 10 on heartrate, then they are equal on the attribute acuity. This is a kind of relationship that can be expressed by an RFD_c.

	subject_id	heartrate	acuity	chiefcomplaint
t_5	14640810	109	3	ABD CRAMPING/NAUSEA
t_6	16007921	88	3	ABD/BACK PAIN/N/V
t_7	16391209	93	3	ABD/BACK PAIN/FEVER

TABLE I: A snippet of the Triage dataset.

As for FDs, RFDs have been used in several application contexts, especially those in which data are collected from heterogeneous sources [6]. An example is the data imputation domain, which aims to populate missing values of database instances by identifying ideal candidates from the same instance, where it is not uncommon to see the use of RFDs as metadata to assist the imputation process [7], [8]. To provide these approaches with the required set of RFDs, discovery algorithms have been proposed to automatically infer RFDs from data, combining the task of searching for RFDs holding on a given dataset, with the identification of the "relaxed" constraints reflecting the meaning of the data [9]. However, due to the dynamic nature of many real-world datasets, it is necessary to adapt discovery processes in order to guarantee the update of the discovered metadata whenever the dataset changes. In particular, discovery algorithms need to process continuously evolving information, since "incremental" scenarios assume that data can be added, deleted, or updated over time [10]. Incremental discovery processes are demanded in many existing application domains for RFDs, like data imputation, when the underlying data continuously evolve [11].

Incremental scenarios make the RFD discovery problem more complex and challenging, especially for the representation and management of data and results. In fact, the number of FDs and RFDs holding on a given dataset can be exponential in the number of attributes, requiring the exploration of extremely large and complex search space [1]. The dynamic scenario adds further complexity having to cope with insertion, deletion, and updating operations. Re-executing the discovery process from scratch upon each change in the dataset is computationally burdensome [10]. Thus, solutions for dynamic scenarios should consider to efficiently (i) (re-)validate previously holding RFD_cs on the updated dataset, (ii) discover new possibly holding RFD_cs, and (iii) guarantee properties like correctness and minimality for discovered RFD_cs.

In this paper, we present INDIBITS, the first incremental discovery algorithm for RFDs relaxing on the attribute comparison (i.e., RFD_cs), which relies on a column-based search strategy to identify RFD_cs even when the underlying data change over time. In particular, INDIBITS optimizes the RFD_c discovery process through a binary representation capable of representing all the distances between the attribute values of a dataset in a compact way, facilitating their modifications upon data updates through efficient bitwise operations. Furthermore, INDIBITS adapts the refinement property used for validating RFD_cs to the dynamic context. Finally, a proper search strategy has been introduced for efficiently browsing the search space according to the specific data updates.

The paper is organized as follows. Section II describes literature approaches and tools for FDs and RFDs discovery. Section III presents the theoretical foundations of considered profiling metadata. Section IV provides an overview of INDIBITS, whereas its underlying data representation method is presented in Section V. Then, Section VI details the validation method underlying INDIBITS, and the procedures to handle insertion and deletion data modifications. Section VII reports experimental results to analyze the effectiveness of INDIBITS on real-world datasets, also when compared with other FD and RFD_c discovery algorithms. Finally, summary and future directions are included in Section VIII.

II. RELATED WORK

INDIBITS represents the first algorithm capable of updating the set of valid RFD_cs upon insertion, deletion, and update operations on data. In this section we categorize literature approaches in three different classes: those for (i) the discovery of FDs in static scenarios, (ii) the discovery of RFDs in static scenarios, and (iii) the incremental discovery of RFDs.

FD *discovery.* The problem of discovering FDs from data dates back to the 1980s, when the first discovery algorithms were defined [2], [12], [13]. Over the years, the literature delineated two main categories of FD discovery algorithms: *column-based* and *row-based*, respectively. Column-based algorithms model the search space as an attribute lattice, which permits to consider candidate FDs at each lattice level in terms of edges, and rely on the Apriori search strategy [13], which also permits to exploit FDs validated at previous lattice levels to prune the search space and generate new candidate FDs. Examples of column-based algorithms include FD_Mine [3], FUN [14], and DFD [15]. Instead, row-based algorithms discover FDs by analyzing two data subsets, also known as *agree-set* and *difference-set*, which represent the product between all possible combinations of tuple pairs. These algorithms search for attribute sets agreeing on the values of certain tuple pairs, since they can functionally determine other attributes agreeing on the same tuple pairs. Examples of row-based algorithms include DepMiner [16], FastFD [17], and FDep [18]. Finally, in order to achieve good performances in high-dimensional datasets with many rows, the discovery algorithm HyFD [19] combines these two types of discovery strategies.

RFD *discovery.* As discussed above, RFDs represent FDs relaxing some constraints of the canonical FD definition. Among these, RFD_es are validated through either a condition or a coverage measure [20], such as the $g3$-error [21] or the error measure of super keys [22], which permit to identify the portion of the relation instance on which an FD is not valid. In general, RFD_es discovery algorithms rely on sampling [21], [23], [24], which consists in validating candidate RFD_es on samples of tuples, meaning that they hold on a relation instance with a given probability. Instead, RFD_cs generalize the definition of FDs by introducing similarity- or distance-based comparisons between attribute values [4]. One of the most recent algorithms for RFD_c discovery is DOMINO [9], which relies on the concept of dominance to automatically derive thresholds of attributes to compare their values through similarity functions. Lastly, in [25] the problem of discovering RFDs relaxing on both the attribute comparison and extent is proven to be NP-hard, hence an approximate algorithm for discovering both RFD_cs and RFD_es is presented, whereas in [26] another approximate solution for the same problem is provided, which relies on an evolutionary strategy.

Examples of RFDs validated through conditions are Conditional Functional Dependencies (CFDS) [27]. They identify the subset of data on which a dependency holds by either enforcing patterns of semantically related constants and/or by restricting the application domain of a dependency by means of a condition [28]. Examples of CFD discovery algorithms include CFDMiner, CTANE, FastCFD [27], and the greedy algorithm proposed in [29].

Incremental discovery. The discovery algorithms for FDs and RFDs discussed above need to be re-executed from scratch whenever the dataset is updated. One of the first theoretical proposals of an incremental FD discovery algorithm has been provided in [30]. It exploits the concepts of tuple partitions and FDs monotonicity to avoid the re-scanning of the entire dataset. A recent algorithm, named DYNFD, has been proposed in [31], which extends the HyFD algorithm [19] by enabling the discovery and update of FDs in dynamic datasets.

In the context of RFD discovery, incremental algorithms for RFD_es are AD-MINER [32] and BIRD [33]. The former exploits logical operations to automatically infer new RFD_es upon the insertion of new tuples, whereas the latter exploits data partitions to continuously update the set of holding RFD_es upon the insertion of new batches of data.

To the best of our knowledge, INDIBITS represents the first solution for the incremental discovery of RFD_cs. In fact, incremental algorithms existing in the literature are either suited for FDs [19], [30], [31] or RFD_es [32], [33]. The former cannot be used to also discover RFD_cs, since they do not consider similarity constraints between attribute values. On the other hand, approaches for the incremental discovery of RFD_es address a different problem with respect to the discovery of RFD_cs. Finally, employing static solutions for RFD_cs discovery [9], [25] in incremental scenarios turns out to be ineffective, since they would require the total re-execution of the discovery process each time the dataset changes.

III. PRELIMINARIES

In this section, we formally introduce preliminary concepts related to RFD$_c$ and its minimality property. A more general definition of RFD can be found in [4]. Table II summarizes the notations used throughout the paper.

As briefly introduced above, a similarity constraint is a predicate evaluating whether the distance or similarity between two values of an attribute falls within predefined intervals. More formally, let r be a relation instance on a relation schema R, a constraint ϕ over an attribute B, denoted as $\phi[B]$, is a predicate $\delta(t_i[B], t_j[B])\theta_k\varepsilon$, with $B \in attr(R)$, δ a distance or similarity function, θ_k a comparison operator, and ε a constant threshold. For sake of simplicity, in the rest of the paper, we will refer to constraints defined on distance functions, one comparison operator (i.e., $\leqslant$), and a threshold.

Definition 1 (RFD$_c$)**.** Let us consider a relational database schema $\mathcal{R}$, and a relation schema $R = (A_1, \ldots, A_m)$ of $\mathcal{R}$. An RFD$_c$ φ on $\mathcal{R}$ is denoted by $X_{\Phi_1} \rightarrow Y_{\Phi_2}$ where:

- $X = X_1, \ldots, X_h$ and $Y = Y_1, \ldots, Y_k$, with $X, Y \subseteq attr(R)$ and $X \cap Y = \varnothing$;
- $\Phi_1 = \bigwedge_{X_i \in X} \phi_i[X_i]$ ($\Phi_2 = \bigwedge_{Y_j \in Y} \phi_j[Y_j]$, resp.), where ϕ_i (ϕ_j, resp.) is a predicate on X_i (Y_j, resp.) with $i = 1, \ldots, h$ ($j = 1, \ldots, k$, resp.). For any pair of tuples (t_1,t_2) $\in dom(R)$, the constraint Φ_1 (Φ_2, resp.) is true, if $t_1[X_i]$ and $t_2[X_i]$ ($t_1[Y_j]$ and $t_2[Y_j]$, resp.) satisfy the constraint ϕ_i (ϕ_j, resp.) $\forall i \in [1, h]$ ($j \in [1, k]$, resp.).

Given a relation instance r of R, r *satisfies* the RFD$_c$ φ, denoted by $r \models \varphi$, if and only if: $\forall\ t_1, t_2 \in r$, if Φ_1 indicates true, then also Φ_2 indicates also true.

Without loss of generality, in what follows we consider only candidate RFD$_c$s with a single attribute on the RHS, i.e., $X_{\Phi_1} \rightarrow A_{\phi_2}$. Moreover, the following examples consider more a compact notation for the constraints.

Example 1. *With respect to the example provided in Section I on the Triage dataset, the aforementioned relationship between heartrate, chiefcomplaint on a hand, and acuity on the other hand, can be captured by an* RFD$_c$ *expressed in the following form:*

$$heartrate_{(\leqslant 10)}, chiefcomplaint_{(\leqslant 5)} \rightarrow acuity_{(\leqslant 0)}$$

where ($\leqslant$ 10), ($\leqslant$ 5), *and* ($\leqslant$ 0) *define the constraints for attributes heartrate, chiefcomplaint, and acuity, respectively, by employing the same distance functions presented above.*

One of the most important characteristics of an RFD$_c$ is minimality, which guarantees that the RFD$_c$ no longer holds after either (i) increasing one or more thresholds on the LHS constraints, (ii) removing an LHS attribute, or (iii) decreasing the RHS threshold. Notice that, the minimality property can be restricted to case (ii) when fixed constraints for each attribute are considered. We formally define a *minimal* RFD$_c$ as follows:

Definition 2 (RFD$_c$ Minimality)**.** Given a relation schema R, $A \in attr(R)$, $X = \{X_1, \ldots, X_n\} \subset attr(R)$, and an RFD$_c$ $X_{\Phi_1} \rightarrow A_{\phi_2}$ holding on a instance r of R, with Φ_1 and ϕ_2 fixed, then the RFD$_c$ is minimal if and only if $\forall X_i \in X, \exists\ t_1, t_2 \in r$ for which $\phi_{X_1} \wedge \cdots \wedge \phi_{X_{i-1}}, \phi_{X_{i+1}} \wedge \cdots \wedge \phi_{X_n}$ indicates true, but ϕ_2 is not satisfied.

Symbol	Description
$\mathcal{R}$	Relational database schema
R	Relation schema
r	Relation instance
φ	RFD$_c$
X, Y, Z	Attribute sets
A, B, C	Attributes
a, b, c	Attribute values
t, t^-	Tuple of r, Tuple removed from r
$t[B]$	Projection of t on the attribute B
$t[X]$	Projection of t on the attribute set X
Φ	Set of distance constraints
ϕ	Distance constraint
M_B^τ	BAS Distance Matrix on the attribute B at time τ
$M_B^\tau[t_k]$	BAS Distance Vector on the attribute B for tuple t_k at time τ
S^τ	Similarity vectors at time τ
S_B^τ	Similarity vector on the attribute B at time τ
$S_B^\tau[t_k]$	Similarity value on the attribute B for tuple t_k at time τ
$S_X^\tau[t_k]$	Similarity value on the attribute set X for tuple t_k at time τ
m	Number of attributes in r
$\lvert r\rvert$	Number of tuples in r
$rfds_\tau$	Set of minimal RFD$_c$s at time τ

TABLE II: A summary of symbols used throughout the paper.

In other words, since we consider RFD$_c$s with a single attribute on the RHS, an RFD$_c$ is minimal when there is no valid RFD$_c$ with the same RHS, same distance constraints, and a subset of its LHS.

The discovery of RFD$_c$s is the problem of finding a set of all *minimal* RFD$_c$s holding on a relation instance r. As said above, one of the possible strategies for discovering RFD$_c$s (namely *column-based*) models the search space as a lattice, which permits to consider candidate RFD$_c$s at different levels in terms of edges. More specifically, let R be a relation schema with m attributes, the lattice representing all the candidate RFDs will consist of a collection of attribute sets, where Level 0 contains the empty set, Level 1 the singleton sets, i.e., a node for each attribute, Level 2 the pair sets, and so forth. Finally, the last level (Level m) contains a single set of all the attributes in R. By following the lattice-based search space representation, a column-based discovery strategy first generates attribute sets X at level l, and then formulates all the possible RFD$_c$s $X_{\Phi_1} \rightarrow A_{\phi_2}$, with $A \notin X$, to be successively validated. Then, considering the RFD$_c$s validated at level l, several pruning strategies can be applied in order to avoid the validation of not minimal candidate RFD$_c$s. Thus, whenever the constraints are specified for each attribute of the relation, the discovery of RFD$_c$s reduces to verify if whenever tuples satisfy the constraints on the LHS attributes, then they also satisfy the one on the RHS attribute. This requires tackling a problem that, in the worst case, is exponential in the number of columns and quadratic in the number of rows. In particular, when the similarity constraints are fixed, discovering RFD$_c$s over a relation instance r has the same theoretical complexity of the FD discovery problem, with a complexity's upper bound that is $O(|r|^2(\frac{m}{2})^2 2^m)$, where $\frac{m}{2} \cdot 2^m$ represents the number of candidates, and $\frac{m}{2} \cdot |r|^2$ is the complexity of a naive approach for the validation, where all pairs of tuples are compared on an average $\frac{m}{2}$ columns, i.e., the average number of attributes

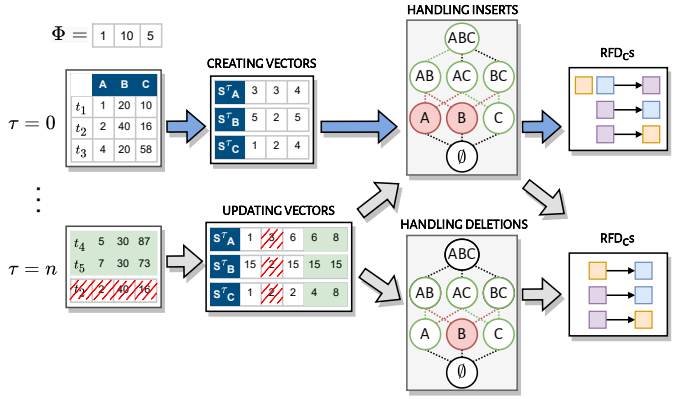


Fig. 1: Overview of the process underlying INDIBITS.

involved in an RFD$_c$ [34]. Nevertheless, differently from the equality, the similarity does not satisfy the transitivity property, preventing the possibility to adopt the validation methods of FDs. This leads to the necessity of conceiving new validation methods for evaluating candidate RFD$_c$s.

IV. INDIBITS

INDIBITS is an incremental discovery algorithm for RFD$_c$s relying on a column-based search strategy capable of discovering RFD$_c$s from a single relation. It considers an input threshold for each attribute to form distance constraints that will be then used for validating candidate RFD$_c$s.

INDIBITS monitors changes occurred in a relation instance in terms of insertion and deletion operations and it incrementally updates the set of valid RFD$_c$s. Notice that update operations can be expressed as a combination of deletion and insertion operations. In what follows, we will use the term *batch* to refer to groups of change operations.

An overview of the discovery process underlying INDIBITS is shown in Fig. 1. As we can see, INDIBITS reads the first batch of tuples at time $\tau = 0$ and creates the binary representation of the data according to the thresholds in Φ defined as input. From this, it performs the discovery process by considering only insertion operations and extracts the set of holding RFD$_c$s. Then, for each batch, INDIBITS updates the data representation and performs the discovery process according to the types of operations performed on the data.

More formally, let $rfds_\tau$ be the set of minimal RFD$_c$s holding on a relation instance r at a given time instant τ, then it is necessary to find a set of *minimal* RFD$_c$s $rfds_{\tau+1}$ holding on the updated relation at a time $\tau+1$. Consequently, it is possible to consider each RFD$_c$ φ holding at time τ as a new candidate RFD$_c$ at time $\tau + 1$, which will be surely valid if the last operation was a deletion, but it could no longer be minimal; whereas it might not hold if the last operation was an insertion. In case it was a deletion making φ: $X_{\Phi'_1} \rightarrow A_{\phi_2}$ not minimal, then it must generate and validate new candidate RFD$_c$s φ' that are *generalization* of φ, that is, $\varphi' : X'_{\Phi'_1} \rightarrow A_{\phi_2}$, with $X' \subset X$, and Φ'_1 is a conjunction of the similarity constraints defined on attributes in X'. Accordingly, in case the last operation is an insertion invalidating φ, then it must generate and validate new candidate RFD$_c$s φ'' that are *specialization* of φ, that is, $\varphi'' : X''_{\Phi''_1} \rightarrow A_{\phi_2}$, with $X \subset X''$ and Φ''_1 is a conjunction of the similarity constraints defined on the attributes X''. INDIBITS first processes all the deletions defined in a batch, and then the insertion operations, enabling the correct handling of the update operations on tuples. Consequently, INDIBITS browses the search space according to the types of operations, and starting from RFD$_c$s holding at time τ, it properly generates candidate RFD$_c$s by means of specialization/generalization strategies (see Sections VI-B and VI-C). Nevertheless, to verify the satisfiability of similarity constraints, and efficiently validate each candidate RFD$_c$, it relies on a compact representation of data similarities (see Section V), from which an efficient validation method has been defined (see Section VI-A).

V. BITWISE SIMILARITY

As mentioned above, we handled the complexity in representing data similarities, also according to possible data modifications, through an innovative strategy, founded over the definition of an ad-hoc data structure, i.e., a *similarity vector*.

In particular, INDIBITS is able to discover RFD$_c$s by considering specific input thresholds characterizing distance constraints to be associated with each attribute. The satisfiability degree limited by such thresholds is represented as a set of numerical values, which are grouped into similarity vectors. To better understand how similarity vectors are generated, we will introduce a theoretical concept called Binary Attribute Satisfiability (BAS) Distance Matrix. Fig. 2 depicts how we represent the satisfiability of attributes with respect to the considered threshold. In particular, for a given attribute B, a BAS Distance Matrix represents a triangular matrix whose row and column indices correspond to the tuples of a relation instance at a given time τ. Each matrix entry will contain either the value 0 or 1 depending on whether the pair of tuples corresponding to the entry has a distance on B greater or less than the threshold associated to B. In what follows, a more formal definition of the BAS Distance Matrix is provided.

Definition 3 (BAS Distance Matrix and Vector)**.** Let r be an instance at time τ of a relation schema R, $B \in attr(R)$, and ε an input threshold for a distance constraint $\phi[B]$. The *BAS Distance Matrix* of r for B, denoted as M^τ_B, is a triangular matrix defined as:

$$M^\tau_B[t_i][t_j] = \begin{cases} 1 & \text{if } \delta(t_i[B], t_j[B]) \leqslant \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

for each pair of tuples $(t_i,t_j) \in r$ with $i \leqslant j$. Thus, the k-th *BAS Distance Vector* for B is a bitwise vector of length $|r|$, denoted as $M^\tau_B[t_k]$, whose values are $\langle M^\tau_B[t_k][t_1], ..., M^\tau_B[t_k][t_k], M^\tau_B[t_{k+1}][t_k], ..., M^\tau_B[t_{|r|}][t_k]\rangle$.

A BAS Distance Vector $M^\tau_B[t_k]$ describes the tuples similar to t_k on attribute B. As an example, $M^\tau_{\text{chiefcomplaint}}[t_2]$ in Fig. 2 corresponds to the bitwise-based vector $0010000010_{(2)}$, and indicates that the value of t_2[chiefcomplaint] is similar to t_8[chiefcomplaint] and to itself. By considering a BAS distance

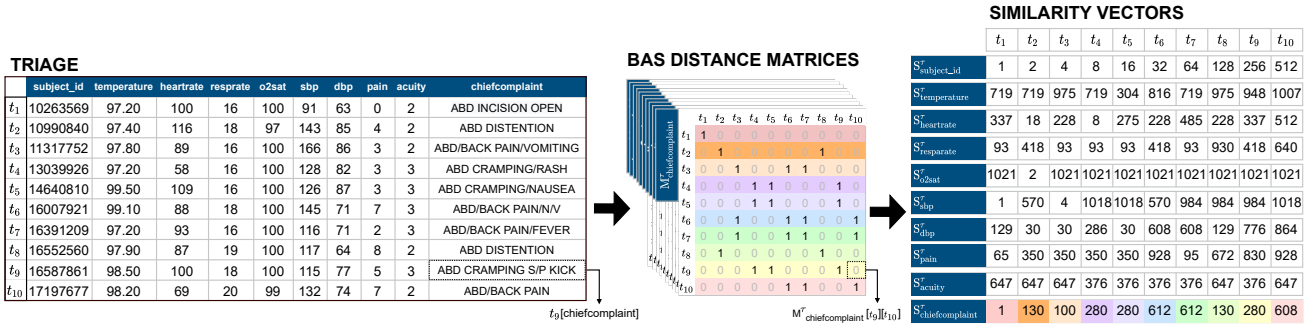


Fig. 2: Similarity Vectors created for the snippet of the *Triage* dataset from *Mimic*-III database.

vector as a decimal value, the similarity between all tuple pairs for a given attribute can be represented by a single vector, named *Similarity Vector*, of length $|r|$.

Definition 4 (Similarity Vector)**.** Let r be an instance at time τ of a relation schema R, $B \in attr(R)$, and M_B^τ a BAS Distance Matrix. The *similarity vector* on B, denoted as S_B^τ, is a vector of size $|r|$ such that $S_B^\tau[t_k]$ contains the decimal representation of $M_B^\tau[t_k]$.

Example 2. *Fig. 2 depicts the BAS Distance Matrix for* chiefcomplaint *attribute and the similarity vectors obtained from the snippet of the Triage dataset. It has been constructed by considering the thresholds* $\{0, 1, 10, 1, 1, 20, 5, 2, 0, 8\}$*, yielding the BAS Distance Matrix for attribute* chiefcomplaint *to contain a value* 1 *when the corresponding pair of tuples have a distance value* $\leqslant 8$*. Each row of the BAS Distance Matrix corresponds to a binary value, and is represented in decimal format in a cell of the* chiefcomplaint *similarity vector. For instance, the value* $1001100000_{(2)}$ *on tuple* t_{10} *corresponds to* $608_{(10)}$*, and is stored in* $S^\tau_{chiefcomplaint}[t_{10}]$.

The list of Similarity Vectors, denoted as S^τ, summarizes the similarities between the tuples of the input relation, providing a lightweight representation that can be efficiently updated upon changes. In particular, the BAS distance Matrix can be updated by performing bitwise operations, as defined in the following sections.

A. Update of similarity vectors

The insertion of new tuples requires updating the data structures underlying IndiBits. Any newly inserted tuple yields the necessity to update the similarity vectors for the validation process. In particular, the similarity of each new tuple is evaluated with respect to the previous ones by calculating the distance of each attribute value of the new tuple from those of already processed tuples. This operation can be thought as the definition of a new row/column within the BAS Matrices of each attribute, as defined in the following.

Definition 5 (BAS Matrix Upon Tuple Insertion)**.** Let r be an instance of a relation schema R at time τ with $|r|$ rows, M_B^τ a BAS Distance Matrix for an attribute B of R, and $t_{|r|+1}$ a tuple added to r at time $\tau + 1$. $M_B^{\tau+1}$ contains $|r| + 1$ rows and is calculated as follows:

$$M_B^{\tau+1}[t_k] = \begin{cases} M_B^\tau[t_k] & \text{if } k \leqslant |r| \\ \langle \delta(t_1[B], t_k[B]), \ldots, \\ \qquad \delta(t_{k-1}[B], t_k[B]), 1 \rangle & \text{if } k = |r| + 1 \end{cases}$$

As discussed above, since the vectors in the BAS Matrices represent a value of the similarity vectors, this implies the creation of a new value within the similarity vectors. Nevertheless, the insertion of a new tuple can also affect the existing values in the similarity vectors. In fact, when a new attribute value is similar to another existing value according to the distance constraint, the new value in the BAS matrix is equal to 1. This means that the value $2^{(i-1)}$ is added to the existing value, where i represents the id of the new row/column added to the BAS matrix.

Example 3. *Let us consider the snippet of the Triage dataset in Fig. 3. After the insertion of the tuples* t_{11} *and* t_{12}*, two columns and rows have been added to the BAS Matrices of each attribute. Let us consider the BAS matrix of the attribute* chiefcomplaint*. As we can see, the value* ABD CRAMPING PREGNANT *of* t_{11} *is similar to the values* ABD CRAMPING/RASH*,* ABD CRAMPING/NAUSEA*, and* ABD CRAMPING S/P KICK *of the tuples* t_4*,* t_5*, and* t_9*, respectively. This has led to add a new value in the similarity vector of* chiefcomplaint*, i.e.,* $1304_{(10)}$*, which corresponds to the decimal representation of* $10100011000_{(2)}$*. Consequently, the values of* t_4*,* t_5*, and* t_9 *in the similarity vectors have also been updated. Likewise, the value for the* chiefcomplaint *of tuple* t_{12}*, i.e.,* ABD INCISION INFECTION*, is similar to* ABD INCISION OPEN *of* t_1*, yielding the insertion of the number* $2049_{(10)}$ *for* t_1 *and* t_{12} *in the similarity vectors.*

As for insertions, the deletion of tuples requires updating the data structures underlying IndiBits. Any tuple deleted at time $\tau+1$ yields the necessity to remove the values referring to it by updating similarity vectors. In particular, the deletion of a tuple can be thought as the drop of a row/column within the BAS Matrices of each attribute, as defined in the following.

Definition 6 (BAS Matrix Upon Tuple Deletion)**.** Let r be an instance at time τ of a relation schema R with $|r|$ rows, M_B^τ a BAS Distance Matrix for an attribute B of R, and t_p a tuple

deleted from r at time $\tau+1$. $M_B^{\tau+1}$ contains $|r|-1$ rows and is calculated as follows:

$$M_B^{\tau+1}[t_k] = \begin{cases} M_B^{\tau}[t_k] & \text{if } k < p \\ (M_B^{\tau}[t_k] \gg p) \vee (M_B^{\tau}[t_k] \Vdash p) & \text{if } k > p \end{cases}$$

where $M_B^{\tau}[t_k] \gg p = \big((M_B^{\tau}[t_k] \gg 1) \wedge [1^{|r|-p}\ 0^{p-1}]\big)$, and $M_B^{\tau}[t_k] \Vdash p = \big(M_B^{\tau}[t_k] \wedge [0^{|r|-p}\ 1^{p-1}]\big)$.

Consequently, whenever a row/column is removed from the matrices, their corresponding values are removed from each vector, yielding the reduction of their dimensionality. However, it is also necessary to update the remaining values in other vectors according to the values of the deleted tuple. In fact, the deletion of a tuple can reduce the value within a similarity vector of $2^{(i-1)}$, where i represents the *id* of the deleted tuple.

Example 4. *Let us consider the snippet of the Triage dataset in Fig. 4. After the deletions of tuples t_2 and t_{10}, two rows/columns have been dropped from the BAS Matrices of each attribute. Let us consider the BAS matrix of the attribute chiefcomplaint, after the deletion of tuples t_2 and t_{10}, their corresponding values have been removed from the similarity vectors, i.e., $130_{(10)}$ and $608_{(10)}$, which correspond to $0010000010_{(2)}$ and $1001100000_{(2)}$, respectively. Moreover, all the remaining values in the similarity vectors, i.e., $[1, 100, 280, 280, 612, 612, 130, 280]$ (Fig. 2), have been updated according to the deletions, yielding a new similarity vector for chiefcomplaint, i.e., $[1, 50, 140, 140, 50, 50, 64, 140]$.*

VI. Incremental RFD$_c$ Discovery

Section V presented the lightweight data structure underlying IndiBits enabling efficient validation and discovery processes, which are introduced in this section.

A. RFD$_c$ Validation

Starting from the representation of similarities provided by the similarity vectors S^τ, it is possible to efficiently verify if a candidate RFD$_c$ is satisfied on a given relation instance r at time τ by exploiting the refinement property between patterns of similarity values introduced in [25].

More formally, given the similarity vectors S^τ, a tuple t_k, and an attribute B, it is possible to consider the binary representation of the value $S_B^\tau[t_k]$, i.e., $M_B^\tau[t_k]$, representing all tuples that are similar to t_k on attribute B, according to the similarity constraint defined by $\phi[B]$. Consequently, for an attribute set X, it is possible to consider the binary representation of the value $S_X^\tau[t_k]$ representing all tuples that are similar to t_k on the values of each attribute in X, according to the similarity constraints defined by Φ. Notice that, the value $S_X^\tau[t_k]$ can be computed as $\bigwedge_{B\in X} S_B^\tau[t_k]$.

According to the refinement property, if we consider a set $X \cup A \supset X$, it is possible to say that $S_{X\cup A}^\tau$ always refines S_X^τ, since each tuple pair is similar on $X \cup A$ if and only if it is also similar on X.

$$||S_X^\tau|| = \frac{\sum_{t_k\in S^\tau} (|S_X^\tau[t_k]| - 1)}{2}$$

where $|S_X^\tau[t_k]|$ is the number of bits equal to 1 in the binary representation of $S_X^\tau[t_k]$, and represents the number of similar tuples in S_X^τ. Notice that, the value 1 is subtracted to exclude the pairs consisting of the same tuples, whereas the division by 2 allows to consider each tuple pair only once.

Since for an attribute set $X \cup A \supset X$, we can say that $S_{X\cup A}^\tau$ always refines S_X^τ, it is possible to state that $||S_{X\cup A}^\tau||$ is always lower than or equal to $||S_X^\tau||$. However, given an RFD$_c$ $\varphi : X_{\Phi_1} \rightarrow A_{\phi_2}$, if $||S_{X\cup A}^\tau||$ is lower than $||S_X^\tau||$, it means that there exists at least a tuple pair that is similar on X but not on $X \cup A$, leading to the detection of a violation. Thus, an RFD$_c$ $\varphi : X_{\Phi_1} \rightarrow A_{\phi_2}$ holds iff $||S_{X\cup A}^\tau|| = ||S_X^\tau||$.

Example 5. *According to the similarity vectors shown in Fig. 2, the following RFD$_c$ is valid:*

$$\textit{pain}_{(\leqslant 2)}, \textit{chiefcomplaint}_{(\leqslant 8)} \rightarrow \textit{o2sat}_{(\leqslant 1)}$$

In fact, if we consider the similarity vectors $S_{\textit{pain}}^\tau = [65, 350, 350, 350, 350, 928, 95, 672, 830, 928]$, $S_{\textit{chiefcomplaint}}^\tau = [1, 130, 100, 280, 280, 612, 612, 130, 280, 608]$, and $S_{\textit{o2sat}}^\tau = [1021, 2, 1021, 1021, 1021, 1021, 1021, 1021, 1021, 1021]$, it is possible to compute the vectors $S_X^\tau = S_{\{\textit{pain, chiefcomplaint}\}}^\tau = [1, 2, 68, 280, 280, 544, 68, 128, 280, 544]$ and $S_{X\cup A}^\tau = S_{\{\textit{pain, chiefcomplaint, o2sat}\}}^\tau = [1, 2, 68, 280, 280, 544, 68, 128, 280, 544]$, yielding the same number of similar pairs:

$$||S_X^\tau|| = \frac{0+0+1+2+2+1+1+0+2+1}{2} = ||S_{X\cup A}^\tau||$$

B. Handling Insertions

In this section, we discuss the strategy to maintain RFD$_c$s upon the insertion of new tuples. As discussed above, the insertion operations can confirm the validity or invalidate RFD$_c$s already validated at time τ, i.e., before the insertion of the new tuples. In particular, IndiBits considers as starting points the minimal RFD$_c$s validated on a relation r at time τ. This strategy enables IndiBits to avoid re-executing the discovery process from scratch, by keeping track of the previously holding RFD$_c$s. Notice that during the first execution of IndiBits, the most general RFD$_c$s in the search space are considered as starting points, i.e., all non-trivial RFD$_c$s having one attribute on the LHS. Starting from these, IndiBits performs the validation from the most general to the most specialized RFD$_c$s, i.e., from RFD$_c$s with the lowest number of the attributes on the LHS to those with the highest one.

More specifically, IndiBits validates each candidate RFD$_c$ according to the validation strategy discussed in Section VI-A. Then, for each candidate RFD$_c$ valid at time $\tau+1$, IndiBits prunes the search space according to the strategy proposed in [2]. In particular, let $X_{\Phi_1} \rightarrow A_{\Phi_2}$ be an RFD$_c$ valid at time $\tau+1$, with $X = X_1, \ldots, X_h$, $X, A \in attr(R)$, and $X \cap A = \varnothing$. For each attribute $B \in attr(R)$, with $B \notin \{X \cup A\}$, then $X \cup A \rightarrow B$ is not minimal at time $\tau+1$. The pruning strategy is applied whenever an RFD$_c$ is validated after the insertion of a batch of tuples, enabling IndiBits to greatly reduce the number of candidate RFD$_c$s to be validated. On the other hand, if there exists at least one candidate RFD$_c$ that is no longer valid at time $\tau+1$, IndiBits specializes it and generates new

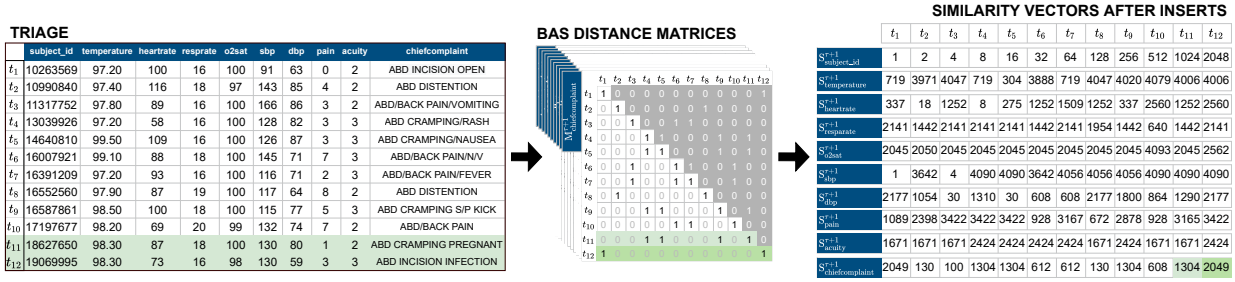


Fig. 3: Updating vectors after the insertion of new tuples over the snippet of the *Triage* dataset.

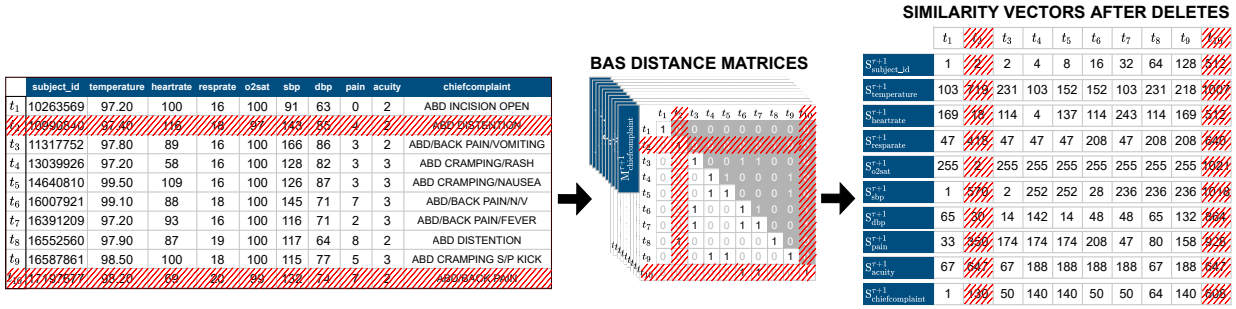


Fig. 4: Updating vectors after the deletion of some tuples over the snippet of the *Triage* dataset.

candidate RFD$_c$s. To ensure the minimality of the resulting RFD$_c$s at time $\tau+1$, before analyzing each newly specialized RFD$_c$, INDIBITS checks if there exists at least one valid RFD$_c$ at time $\tau+1$ that generalizes it. If so, the specialization is a valid and not minimal RFD$_c$, since an RFD$_c$ that generalizes it has already been validated at time $\tau+1$.

Example 6. *Let us consider the snippet of the Triage dataset in Fig. 3, and the set $rfds_\tau$ containing the following* RFD$_c$*s*
φ_1 : *subject_id*$_{(\leqslant 0)} \rightarrow$ *o2sat*$_{(\leqslant 1)}$,
φ_2 : *temperature*$_{(\leqslant 1)}$, *heartrate*$_{(\leqslant 10)} \rightarrow$ *o2sat*$_{(\leqslant 1)}$,
φ_3 : *dbp*$_{(\leqslant 5)}$, *chiefcomplaint*$_{(\leqslant 8)} \rightarrow$ *o2sat*$_{(\leqslant 1)}$.
After the insertion of the tuples t_{11} and t_{12}, the RFD φ_3 *is no longer valid, while the others continue to be. Thus,* INDIBITS *generates the specializations of φ_3, such as:*
φ_{31} : *subject_id*$_{(\leqslant 0)}$, *dbp*$_{(\leqslant 5)}$, *chiefcomplaint*$_{(\leqslant 8)} \rightarrow$ *o2sat*$_{(\leqslant 1)}$,
φ_{32} : *temperature*$_{(\leqslant 1)}$, *dbp*$_{(\leqslant 5)}$, *chiefcomplaint*$_{(\leqslant 8)} \rightarrow$ *o2sat*$_{(\leqslant 1)}$,
and so forth. But the RFD$_c$ φ_{31} *is a specialization of φ_1 included in $rfds_\tau$, and it does not require to be validated, whereas the* RFD$_c$ φ_{32} *must be validated at time $\tau+1$.*

Algorithm 1 shows the procedure for updating vectors upon each new tuple insertion at time $\tau+1$. It starts by considering the set of already processed tuples in the relation r with the corresponding similarity vectors at time $\tau+1$, and a new tuple t inserted at time $\tau+1$. The procedure first reads the size of the stored tuples, which corresponds to the id of the newly inserted tuple t (Line 2). Then, for each attribute B in $attr(R)$, it updates the similarity vectors according to the strategy defined in Section V-A (Lines 2-7). More specifically, the procedure compares the value of the new tuple over each attribute B with those of the tuples t_k already processed (Lines 4-7). If these values are similar according to the distance constraint $\phi[B]$, then the procedure updates the existing values in the similarity vectors, by adding 2^{id} to the existing $S^\tau_B[t_k]$ (Line 6). Then, it also updates the value $S^\tau_B[t_{id}]$ by adding the value 2^k corresponding to t_k (Line 7). The procedure ends by returning the resulting similarity vectors (Lines 8-9).

Algorithm 1 Updating vectors after inserting a new tuple

INPUT: A relation instance r at time τ; A new tuple t at time $\tau+1$; A set Φ of distance constraints; Similarity vectors S^τ
OUTPUT: Updated similarity vectors $S^{\tau+1}$

1: $id \leftarrow |r|$
2: **for each** $B \in attr(R)$ **do**
3: $\quad S^\tau_B[t_{id}] \leftarrow 2^{id}$
4: $\quad$ **for** $k \in 0 \ldots |r|$ **do**
5: $\quad\quad$ **if** $\delta(t[B], t_k[B])$ **satisfies** $\phi[B]$ **then**
6: $\quad\quad\quad S^\tau_B[t_k] \leftarrow S^\tau_B[t_k] + 2^{id}$
7: $\quad\quad\quad S^\tau_B[t_{id}] \leftarrow S^\tau_B[t_{id}] + 2^k$
8: $S^{\tau+1} \leftarrow S^\tau$
9: **return** $S^{\tau+1}$

Algorithm 2 shows the main procedure of INDIBITS executed for every batch of new tuples inserted at time $\tau+1$. It considers the set of all RFD$_c$s holding at time τ ($rfds_\tau$), the set of all RFD$_c$s that were not valid at time τ ($invalidRfds_\tau$), and the similarity vectors updated upon the insertion of the new batch of tuples at time $\tau+1$ ($S^{\tau+1}$). The procedure starts by considering candidate RFD$_c$s from the most general to the most specialized ones, i.e., from RFD$_c$s with the minimum number of attributes on the LHS to the one with the maximum one (Line 1). Then, it validates each candidate RFD$_c$ according to the approach described in Section VI-A (Lines 3-4). Thus, if the analyzed RFD$_c$ is valid at time $\tau+1$, the procedure prunes the search space by removing holding RFD$_c$s according to the pruning strategy discussed above (Lines 7-8). On the contrary, if a candidate RFD$_c$ is not valid at time $\tau+1$, it is added to the set of invalid RFD$_c$s (Lines 5-6). Then, the procedure removes the invalid RFD$_c$s from $rfds_\tau$, and it

Algorithm 2 Main procedure of INDIBITS after inserting tuples

INPUT: A set $rfds_\tau$ of minimal RFD$_c$s; A set $invalidRfds_\tau$ of RFD$_c$s not valid at time τ; Similarity vectors $S^{\tau+1}$
OUTPUT: Updated $rfds_{\tau+1}$, updated $invalidRfds_{\tau+1}$

1: **for** $level \in 0 \dots m$ **do**
2: $\quad$ $rfdsInvalidated \leftarrow \varnothing$
3: $\quad$ **for each** $\varphi \in rfds_\tau.getLevel(level)$ **do**
4: $\quad\quad$ **if not** $validate(\varphi, S^{\tau+1})$ **then**
5: $\quad\quad\quad$ $rfdsInvalidated \leftarrow rfdsInvalidated \cup \{\varphi\}$
6: $\quad\quad\quad$ $invalidRfds_\tau \leftarrow invalidRfds_\tau \cup \{\varphi\}$
7: $\quad\quad$ **else**
8: $\quad\quad\quad$ $rfds_\tau.pruneRfds(\varphi)$
9: $\quad$ **for each** $\varphi \in rfdsInvalidated$ **do**
10: $\quad\quad$ $rfds_\tau \leftarrow rfds_\tau \backslash \{\varphi\}$
11: $\quad\quad$ $spec \leftarrow specialize(\varphi)$
12: $\quad\quad$ **for each** $\varphi' \in spec$ **do**
13: $\quad\quad\quad$ **if not** $rfds_\tau.containsGeneralizations(\varphi')$ **then**
14: $\quad\quad\quad\quad$ **if not** $isPruned(\varphi')$ **then**
15: $\quad\quad\quad\quad\quad$ $rfds_\tau \leftarrow rfds_\tau \cup \{\varphi'\}$
16: $rfds_{\tau+1} \leftarrow rfds_\tau$
17: $invalidRfds_{\tau+1} \leftarrow invalidRfds_\tau$
18: **return** $rfds_{\tau+1}$, $invalidRfds_{\tau+1}$

generates their specializations (Lines 10-11). However, before adding the resulting specializations to the set of the candidate RFD$_c$s, the procedure checks their minimality with respect to the RFD$_c$s already validated and those pruned at time $\tau + 1$ (Lines 12-15). The procedure continues its execution until all dependencies have been analyzed. Finally, it returns the sets of holding and invalid RFD$_c$s at time $\tau + 1$ (Lines 16-18).

C. Handling Deletions

Let us now introduce the approach for updating the set of valid RFD$_c$s upon the deletion of some existing tuples. As discussed above, the deletion of one or more tuples always confirms, at time $\tau + 1$, the validity of the previously holding RFD$_c$s, but it could lead to the validation of some RFD$_c$s that were not valid at time τ. In the last case, it is necessary to check the minimality of previously valid RFD$_c$s with respect to those newly validated at time $\tau+1$. In particular, INDIBITS starts by considering the minimal RFD$_c$s holding on a relation instance r at time τ and performs the validation from the most specialized to the most generalized RFD$_c$s.

INDIBITS generalizes each RFD$_c$ holding at time τ and considers the direct generalizations as new candidate RFD$_c$s at time $\tau + 1$. If none of these are valid, INDIBITS confirms the validity of the RFD$_c$ from which the generalizations have been produced. On the contrary, for each candidate RFD$_c$ validated at time $\tau + 1$, INDIBITS removes all the RFD$_c$s that are specializations of it and it continues to generalize them until the direct generalization are all invalid, or until it is no longer possible to generalize, i.e., when it is not possible to remove any attribute from the LHS of an RFD$_c$.

More formally, let $X_{\Phi_1} \rightarrow A_{\Phi_2}$ be an RFD$_c$ valid at time τ with $X, A \in attr(R)$ and $X \cap A = \varnothing$. After the deletion of a batch of tuples, INDIBITS generalizes $X_{\Phi_1} \rightarrow A_{\Phi_2}$, by introducing all the direct generalizations $\{X_{\Phi_1} \backslash B\} \rightarrow A_{\Phi_2}$ for each $B \in X$ as new candidates. If at least one of them is valid at time $\tau + 1$, then RFD$_c$ $X_{\Phi_1} \rightarrow A_{\Phi_2}$ is no longer minimal and other more general RFD$_c$s can be validated.

Algorithm 3 Updating vectors after deleting a tuple

INPUT: A relation r at time τ; A tuple t^- deleted at time $\tau + 1$; Similarity vectors S^τ
OUTPUT: Updated similarity vectors $S^{\tau+1}$

1: $id \leftarrow getTupleId(t^-)$
2: **for each** $B \in attr(R)$ **do**
3: $\quad$ **for** $k \in 0 \dots |r|$ **do**
4: $\quad\quad$ **if** $k \neq id$ **then**
5: $\quad\quad\quad$ $v_1 \leftarrow getBits(S^\tau_B[t_k], 0, id - 1)$
6: $\quad\quad\quad$ $v_2 \leftarrow getBits(S^\tau_B[t_k], id + 1, |r| - 1) \gg 1$
7: $\quad\quad\quad$ $v \leftarrow v_1.concat(v_2)$
8: $\quad\quad\quad$ $S^\tau_B[t_k] \leftarrow convertBinaryNumber(v)$
9: $S^{\tau+1} \leftarrow S^\tau$
10: **return** $S^{\tau+1}$

Example 7. *Let us consider the snippet dataset in Fig. 4, and let us suppose that the following* RFD$_c$
φ_1 : $heartrate_{(\leqslant 10)}, o2sat_{(\leqslant 1)}, sbp_{(\leqslant 20)}, pain_{(\leqslant 2)} \rightarrow temperature_{(\leqslant 1)}$
is minimal and validated at time τ. After the deletions of the tuples t_2 and t_{10}, INDIBITS *generates the direct generalizations of φ_1 such as:*
φ_{11} : $heartrate_{(\leqslant 10)}, o2sat_{(\leqslant 1)}, sbp_{(\leqslant 20)} \rightarrow temperature_{(\leqslant 1)}$,
φ_{32} : $heartrate_{(\leqslant 10)}, sbp_{(\leqslant 20)}, pain_{(\leqslant 2)} \rightarrow temperature_{(\leqslant 1)}$,
and so forth. Thus, since φ_{32} is valid at time $\tau + 1$, then φ_1 is no longer minimal.

Algorithm 3 shows the procedure for updating similarity vectors upon each tuple deletion at time $\tau + 1$. In particular, it considers the set of already processed tuples in the relation r at time τ with the corresponding similarity vectors, and a tuple t^- deleted at time $\tau + 1$. The procedure first retrieves the id of the tuple to be deleted from r (Line 1), which corresponds to the id of the tuple assigned during its insertion. Then, for each similarity vector S^τ_B of $attr(R)$, and for each tuple t_k it updates the values in $S^\tau_B[t_k]$ by removing the bit corresponding to the tuple identifier id (Lines 2-8). In particular, for each value $S^\tau_B[t_k]$, the procedure extracts the bits from 0, i.e., the less significant bit, to $id-1$ (Line 5). Then, it extracts the bits from $id+1$ to $|r|-1$, and concatenates them to remove the bit in position id (Lines 6-7). The bit values are then converted to a decimal value, leading to $S^\tau_B[t_k]$ (Line 8). After performing this operation over all tuples and attributes, the procedure returns the similarity vectors (Lines 9-10).

Algorithm 4 shows the main procedure of INDIBITS executed for every batch of tuples deleted at time τ. It considers the set of all RFD$_c$s holding at time τ ($rfds_\tau$), the set of all RFD$_c$s that were not valid at time τ ($invalidRfds_\tau$), and the similarity vectors updated after the deletion of the tuples at time $\tau + 1$ ($S^{\tau+1}$). The procedure checks if there are still values left in $S^{\tau+1}$ after deleting the tuples at time $\tau + 1$. If $S^{\tau+1}$ is empty, the procedure returns an empty set of valid/invalid RFD$_c$s (Lines 1-3). On the contrary, the procedure starts by considering as candidate RFD$_c$s those that were not valid before the deletion, and it analyzes them from the most specialized to the general one (Lines 6-11). Then, it validates each candidate RFD$_c$ according to the approach described in Section VI-A (Line 8). If a candidate RFD$_c$ is valid, the procedure removes it from the not valid RFD$_c$s and adds its generalizations to the set of new candidate RFD$_c$s (Lines 9-11). Then, INDIBITS checks if the new valid RFD$_c$s

Algorithm 4 Main procedure of INDIBITS after deleting tuples

INPUT: A set $rfds_\tau$ of minimal RFD$_c$s; A set $invalidRfds_\tau$ of not valid RFD$_c$s; Similarity vectors $S^{\tau+1}$
OUTPUT: Updated $rfds_{\tau+1}$, updated $invalidRfds_{\tau+1}$

1: **if** $S^{\tau+1}$.*isEmpty()* **then**
2: $rfds_{\tau+1} \leftarrow \varnothing$
3: $invalidRfds_{\tau+1} \leftarrow \varnothing$
4: **else**
5: $validRfds \leftarrow \varnothing$
6: **for** $level \in m \dots 1$ **do**
7: **for** $\varphi \in invalidRfds_\tau.getLevel(level)$ **do**
8: **if** *validate*(φ, $S^{\tau+1}$) **then**
9: $invalidRfds_\tau \leftarrow invalidRfds_\tau \cup generalize(\varphi)$
10: $invalidRfds_\tau \leftarrow invalidRfds_\tau \backslash \{\varphi\}$
11: $validRfds \leftarrow validRfds \cup \{\varphi\}$
12: **for** $\varphi \in validRfds$ **do**
13: **if not** $rfds_\tau.containsGeneralizations(\varphi)$ **then**
14: $rfds_\tau.addAndCheckMinimality(\varphi)$
15: $rfds_{\tau+1} \leftarrow rfds_\tau$
16: $invalidRfds_{\tau+1} \leftarrow invalidRfds_\tau$
17: **return** $rfds_{\tau+1}$, $invalidRfds_{\tau+1}$

are minimal with respect to the already validated ones, and adds them to the set of valid RFD$_c$s after the deletion of tuples (Lines 12-14). Notice that, the *addAndCheckMinimality* function will remove all RFDs that are not minimal with respect to the added one. Finally, the procedure returns the sets of holding and invalid RFD$_c$s at time $\tau + 1$ (Lines 16-17).

D. Theoretical analysis

In what follows, we provide proofs of correctness and minimality for INDIBITS.

Theorem 1 (Correctness)**.** *Each* RFD$_c$ *discovered by* INDIBITS *is valid.*

Proof. We proceed by contradiction. Let $\varphi : X_{\Phi_1} \rightarrow A_{\phi_2}$ be an RFD$_c$ discovered by INDIBITS from a relation instance r, with $X = \{X_1, \dots, X_n\}$ and $\Phi_1 = \phi_{X_1} \wedge \dots \wedge \phi_{X_n}$. Let us suppose that φ is not valid. This means that there exists a tuple pair $(t_k, t_l) \in r$ for which Φ_1 is satisfied, but not ϕ_A. According to the validation process defined in Section VI-A, if φ is discovered by INDIBITS then $||S^\tau_{X\cup A}|| = ||S^\tau_X||$. This means that $\sum_{t_i \in S^\tau_{X\cup A}} |S^\tau_{X\cup A}[t_i]| = \sum_{t_i \in S^\tau_X} |S^\tau_X[t_i]|$. From definition of similarity vectors on t_k, $|S^\tau_{X\cup A}[t_k]| = |S^\tau_X[t_k]|$, which corresponds to $|\bigwedge_{B\in X\cup A} S^\tau_B[k]| = |\bigwedge_{B\in X} S^\tau_B[k]|$. For the tuple pair (t_k, t_l), we have $|\bigwedge_{B\in X\cup A} M^\tau_B[t_k][t_l]| = |\bigwedge_{B\in X} M^\tau_B[t_k][t_l]|$. Since the pair (t_k, t_l) represents a violation of φ, we have that $\bigwedge_{B\in X} M^\tau_B[t_k][t_l] = 1$ and $\bigwedge_{B\in X\cup A} M^\tau_B[t_k][t_l] = 0$, leading to $||S^\tau_{X\cup A}|| \neq ||S^\tau_X||$. This contradicts the assumption that INDIBITS discovers φ. □

Theorem 2 (Minimality)**.** *Each* RFD$_c$ *discovered by* INDIBITS *is minimal according to Definition 2.*

Proof. We proceed by contradiction. Let $\varphi : X_{\Phi_1} \rightarrow A_{\phi_2}$ be an RFD$_c$ discovered by INDIBITS from a relation instance r, with $X = \{X_1, \dots, X_n\}$ and $\Phi_1 = \phi_{X_1} \wedge \dots \wedge \phi_{X_n}$. Let us suppose that φ is not minimal. This means that there exists a valid RFD$_c$ $\varphi' : X'_{\Phi'_1} \rightarrow A_{\phi_2}$, where $X' = X \backslash X_i$ and $\Phi'_1 = \phi_{X_1} \wedge \phi_{X_{i-1}} \wedge \phi_{X_{i+1}} \wedge \phi_{X_n}$ with $i \in [1, n]$. The search strategy used by INDIBITS to discover φ depends on the type of operation performed to update the dataset, leading the analysis of two separate cases:

Case 1 - Tuple insertions. According to Algorithm 2, at time $\tau + 1$ INDIBITS validates the set $rfds_\tau$ of minimal RFD$_c$s discovered at time τ. Let us suppose that φ is at level k of the lattice. If φ was minimal at time τ and is still valid at time $\tau + 1$, then φ is also minimal, since the insertion of new tuples can only invalidate previously discovered RFD$_c$s, i.e., the insertion of tuples guarantees that all generalizations of φ are still not valid (see Section IV). This contradicts the assumption on the existence of φ'. Conversely, if φ was valid but not minimal at time τ and is valid at time $\tau + 1$, then it has been discovered as specialization of an invalid RFD$_c$ at level $k - 1$. The minimality property of φ is guaranteed by lines 13-15 of Algorithm 2, which verify the absence of any other minimal RFD$_c$ generalizing φ. This refuses the existence of φ', contradicting the original assumption.

Case 2 - Tuple deletions. According to Algorithm 4, at time $\tau + 1$ INDIBITS validates the set $rfds_\tau$ by also considering the set of all invalid RFD$_c$s at time τ ($invalidRfds_\tau$). Let us suppose that φ is at level k of the lattice. If φ was minimal at time τ, it is still valid after the deletion of tuples at time $\tau + 1$. However, if φ' is validated at time $\tau + 1$, then φ' is added to the set $validRfds$ (Line 11) and the function *addAndCheckMinimality* at Line 14 would remove φ from $rfds_\tau$. Thus, INDIBITS would not discover φ contradicting the original assumption. Conversely, if φ was not valid at time τ but valid at time $\tau + 1$, then INDIBITS considers all its generalizations as candidate RFD$_c$s (Line 9), including φ'. However, the minimality check at Lines 12-14 removes φ from $rfds_\tau$ since it contains a generalization of φ (Line 13). Thus, INDIBITS would not discover φ contradicting the original assumption. □

VII. EXPERIMENTAL EVALUATION

We present experimental results concerning the performances of INDIBITS in terms of discovery results, execution time, and memory consumption, and compare them with those of DiMε[1] [25], an RFD discovery algorithm for static scenarios, and DYNFD[2] [31], an FD discovery algorithm for dynamic scenarios.

A. Experimental setup

We implemented INDIBITS in Java 17, using the Levenshtein distance for comparing textual attributes, the absolute difference for numerical attributes, and the alphabetic distance for individual characters. During the data pre-processing phase, INDIBITS constructs the similarity vectors according to user-defined distance thresholds. In particular, for each dataset we performed 5 experiments, using the same threshold ε for all of their attributes, i.e., 0, 1, 2, 4, and 8. Moreover, for each of them we analyzed INDIBITS performances by considering 4 batch size (e.g., the number of changes, in terms of insertion and/or deletion operations, to be considered at each time instant), i.e., $1, 10, 100, 1000$. Finally, we considered a time limit (TL) of 3 hours. All the experiments have been executed

[1]DiMε is available at https://dastlab.github.io/dime/
[2]DynFD is available at https://github.com/HPI-Information-Systems/dynfd

Dataset	#Cols	#Rows	#RFD$_c$s									
			Insertion					Deletion				
			$\varepsilon = 0$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	$\varepsilon = 0$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$
Abalone	9	4,177	137	34	56	64	64	191	34	56	64	64
Glass	10	214	124	27	22	38	73	168	34	28	38	73
Cmc	10	1,473	1	27	36	72	72	3	27	36	72	72
Poker-hand	11	1,025,010	1	0	0	50	50	1	0	0	50	50
Australian	14	690	535	71	78	78	91	2,352	88	78	78	91
Adult	15	32,561	78	81	63	38	33	361	738	292	111	104
Sgemm-gpu	18	241,600	4	68	68	68	119	4	68	68	68	119
Lymphography	19	148	2,730	342	342	342	308	86,558	342	342	342	308
Ncvoter	19	1,001	775	387	256	237	181	599	472	355	276	208
Flights	38	1,000	1,904	1,562	1,340	1,059	1,115	3,495	3,416	1,365	1,063	1,145
Sonar	60	208	75,086	3,540	3,540	3,540	3,540	7,462	3,540	3,540	3,540	3,540
Movement-Libras	91	360	348,215	5,787	5,787	5,787	5,787	436,576	5,787	5,787	5,787	5,787
Uniprot	223	1001	TL	TL	616,187	1,473,270	1,180,725	TL	TL	TL	4,747,193	3,974,670
Tuandromd	242	4,465	9,189	15,602	15,602	15,666	15,666	10,920	15,602	15,602	15,666	15,666

TABLE III: Details of the considered real-world datasets and number of holding RFD$_c$s.

on an iMac Pro with an Intel Xeon CPU at 3.20 GHz, 18-core, and 128GB of memory, running macOS Mojave and OpenJDK 17 as Java environment. The experiments were performed on different real-world datasets[3], previously used for evaluating FD and RFD discovery algorithms. Table III shows the characteristics of the datasets involved in the evaluation.

B. Performances on real-world datasets

Our first experiment measured the number of discovered RFD$_c$s (see Table III), the execution times, and the memory consumption of INDIBITS on 14 real-world datasets (see Fig. 5). The considered datasets have a different number of rows and columns, in order to highlight how the execution times of INDIBITS vary according to such parameters. Since these datasets have not been designed for incremental discovery, we simulated an incremental scenario in which tuples are firstly inserted, and then deleted. In particular, for each dataset, we have first performed the insertion operations of all tuples, and then we have randomly deleted 90% of them.

The number of discovered RFD$_c$s at the end of insertion and deletion operations is detailed in Table III. Concerning the insertion operations, we observe that the number of RFD$_c$s is not necessarily greater when the distance threshold increases. In fact, often a huge number of RFD$_c$s is obtained with threshold 0, i.e., when considering canonical FDs, such as in the case *Abalone*, *Glass*, *Australian*, *Lymphography*, *Ncvoter*, *Sonar*, and *Movement-Libras* datasets; whereas for *Cmc* and *Sgemm-gpu* datasets the number of RFD$_c$s increases with higher distance thresholds. According to results on the last group of datasets, we can state that the impact of higher distance thresholds on the number of RFD$_c$s is due to the fact that when only few FDs hold, the algorithm is capable to catch significant relationships among data, based on the relaxation of the attribute comparison constraints. However, few considerations can be made on the number of RFD$_c$s discovered for the *Uniprot* dataset, since INDIBITS has reached the time limit for lower similarity thresholds, while it was able to discover a huge number of RFD$_c$s for the larger thresholds.

Concerning the deletion operations, the number of RFD$_c$s holding on the considered datasets is typically greater than the number of discovered RFD$_c$s upon the insertion operations. Exceptions hold for *Poker-hand*, *Sgemm-gpu*, *Sonar*, and *Movement-Libras* datasets, and in part for *Tuandromd* dataset. In general, for both insertion and deletion operations, the number of RFD$_c$s tends to stabilize when the attribute comparison threshold flattens the distribution of distances among tuple pairs. This permits not only to have a similar or equal number of resulting RFD$_c$s, but they are typically characterized by a low number of attributes on their LHSs as the comparison threshold increases.

Concerning time and memory performances, we report the average runtimes and memory peaks in Fig. 5, by grouping the results according to batch sizes and distance constraints. In particular, INDIBITS almost always required less than 10^4 MB of memory, except for *Poker-hand*, *Adult*, *Sgemm-gpu*, *Movement-Libras*, *Uniprot*, and *Tuandromd*, in which the resulting memory peaks never exceed 10^5 MB. In general, the low memory consumption of INDIBITS are mainly due to the lightweight representation of data and distances that makes the memory requirements not severely affected by the dimensionality of datasets and the number of holding RFD$_c$s.

In general, we can notice that runtimes are quite stable or slighly grow when the batch size increases. Moreover, almost always the average times are less than 10 ms, except for *Lymphography*, *Flights*, *Movement-Libras*, *Uniprot*, and *Tuandromd* datasets. The last three datasets also exceeded the TL in some configurations. Although such datasets represent the three biggest datasets in terms of attributes (see Table III), for the *Movement-Libras* and *Tuandromd* datasets, INDIBITS reached a time limit only with threshold 0 in the last batch size, but the average runtimes for other configurations do not exceed 10^3 ms (i.e., 1 sec). Instead, concerning the *Uniprot* dataset, INDIBITS completed the discovery process with the two highest attribute comparison thresholds, considering batch sizes equal to 1, 10, and 100. On the other hand, by considering *Adult*, *Sgemm-gpu*, and *Poker-hand*, representing the most challenging datasets in terms of number of rows, INDIBITS seems not to be affected in terms of time performances. In fact, although such datasets are characterized by a considerable amount of tuples to be processed, the average runtimes are comparable to the ones obtained on smaller datasets, such as *Cmc* or *Australian*, suggesting that INDIBITS might have scalability potential over large-size datasets.

Summarizing, for the biggest considered datasets (i.e.,

[3]https://github.com/DastLab/TestDataset

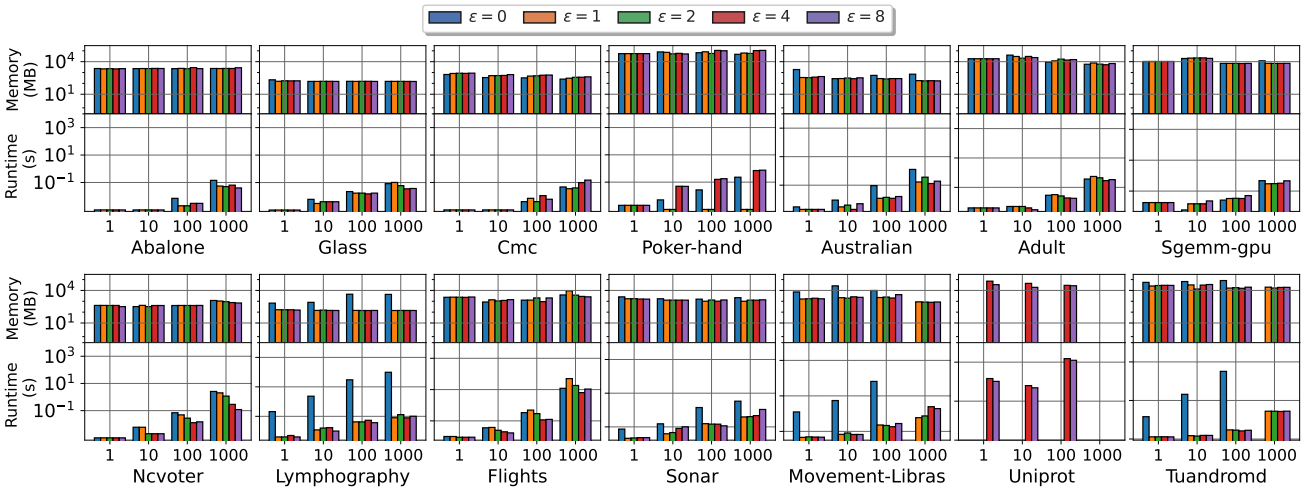


Fig. 5: Performances of INDIBITS over real-world datasets.

Adult, *Sgemm-gpu*, *Poker-hand*, *Movement-Libras*, *Uniprot*, and *Tuandromd*) INDIBITS achieved good time performances with respect to both number of tuples and attributes.

Overall, we cannot identify a strict correlation between the dimensionality of the datasets and both the number of RFD_cs and the INDIBITS runtimes. In particular, although for datasets with high dimensionality INDIBITS can potentially discover more RFD_cs, due to the wider range of candidate RFD_cs, there are some exceptions, such as the *Lymphography* dataset, which presents a high number of FDs despite having 19 attributes only. Similarly, the high dimensionality of datasets does not imply a trend of exponential growth in the number of holding RFD_cs. Such a gap is even less predictable when higher thresholds are involved, since for many datasets runtimes are proven to be faster. This can be due to the reduced variability of resulting RFD_cs in accordance with data changes when higher thresholds are considered.

C. Comparative evaluation

As a second experiment, we compared the performances of INDIBITS to those of DiMε [25] and DYNFD [31].

DiMε is a static algorithm relying on a column-based strategy to discover RFDs relaxing on the attribute comparison (RFD_cs), and/or the extent (RFD_es). In this experiment, we will focus on the discovery of RFD_cs by analyzing in which conditions INDIBITS under- or out-performs DiMε. To this end, we gradually scale up the size of the dataset according to a batch size, each time executing DiMε on an increased dataset. Then, we plot the average runtimes of INDIBITS against those of DiMε, by considering the speedup measure. A speed-up of 10 indicates that INDIBITS has been 10 times faster than DiMε, 1 indicates that they obtained the same runtime, while a value lower than 1 indicates that DiMε was faster. Notice that, when DiMε reaches the TL, we set the value 3-hours as its runtime, and the speed-up bar is marked with a start on the top. Instead, when both algorithms reach the TL, the speed-up bar is marked with a black square on the top.

Fig. 6 shows the results of the comparative evaluation. We can notice that INDIBITS is almost always faster than DiMε, especially on the *Abalone*, *Cmc*, and *Ncvoter* datasets, where INDIBITS has been more than 1000 times faster than DiMε in most configurations. Conversely, only a few times DiMε outperforms INDIBITS, i.e., on the *Lymphography* dataset with threshold 0 and batch sizes set to 100 and 1000, and on the *Sonar* dataset with threshold 0 and batch sizes set to 100 and 1000, and with thresholds 4 and 8 and batch size 1000. As mentioned above, this can be due to the fact that INDIBITS performs worse when there are many invalidations in each batch. Moreover, for the *Sonar* dataset, a static approach like DiMε performs better because its discovery strategy exploits RFD_cs discovered on lowest lattice levels to reduce the execution of validation processes. Nevertheless, we can notice that DiMε suffers when executing on the five datasets with the highest number of attributes and the three with the highest number of rows, since it reaches the TL in many configurations. Instead, INDIBITS reaches the TL only in few cases of the three datasets with the highest number of columns.

Concerning the comparative evaluation between INDIBITS and DYNFD, we set up insertion and deletion operations by considering the experimental configuration introduced in Section VII-B, but limiting the analysis to only the similarity threshold 0, i.e., the FDs. This is due to the fact that DYNFD focuses only on holding FDs upon the insertion and deletion of batches of tuples, but not on RFD_cs.

Figg. 7a and 7b show the results of the comparative evaluation. Notice that, although INDIBITS is not optimized for the discovery of FDs, it is able to achieve competitive runtimes with respect to one of the most efficient incremental FD discovery algorithms. In fact, the results show that in many cases INDIBITS outperforms or achieves average execution times similar to DYNFD. In particular, concerning the insertion operations, we notice that INDIBITS typically outperforms DYNFD with smaller batches of tuples, i.e., 1, 10, and 100, except for *Lymphography* and *Flights* datasets. Moreover, as we can see for batches with sizes 1000, although it seems that DYNFD outperforms INDIBITS for *Glass* and *Australian* datasets, the average execution times are of the

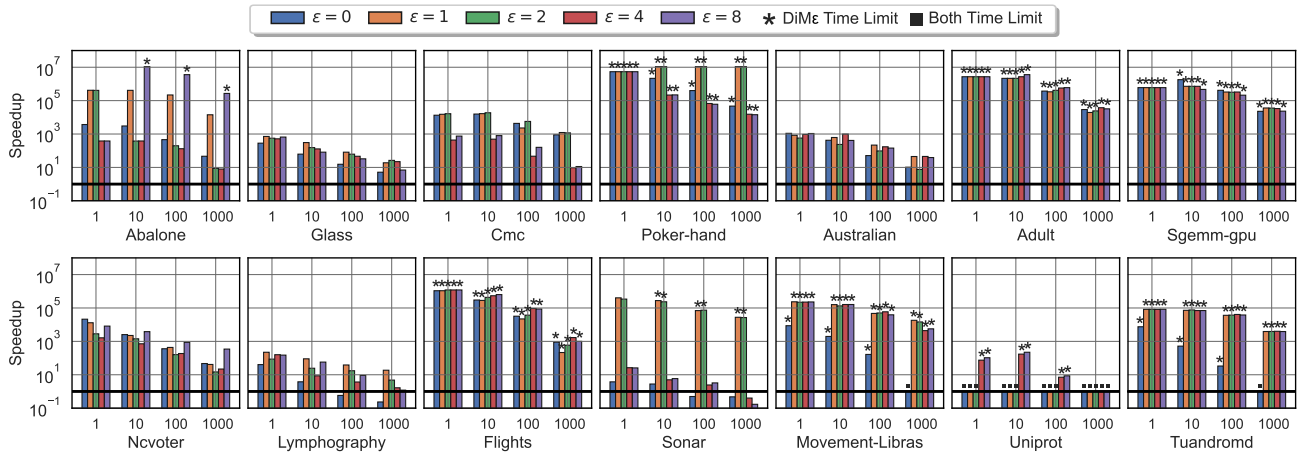
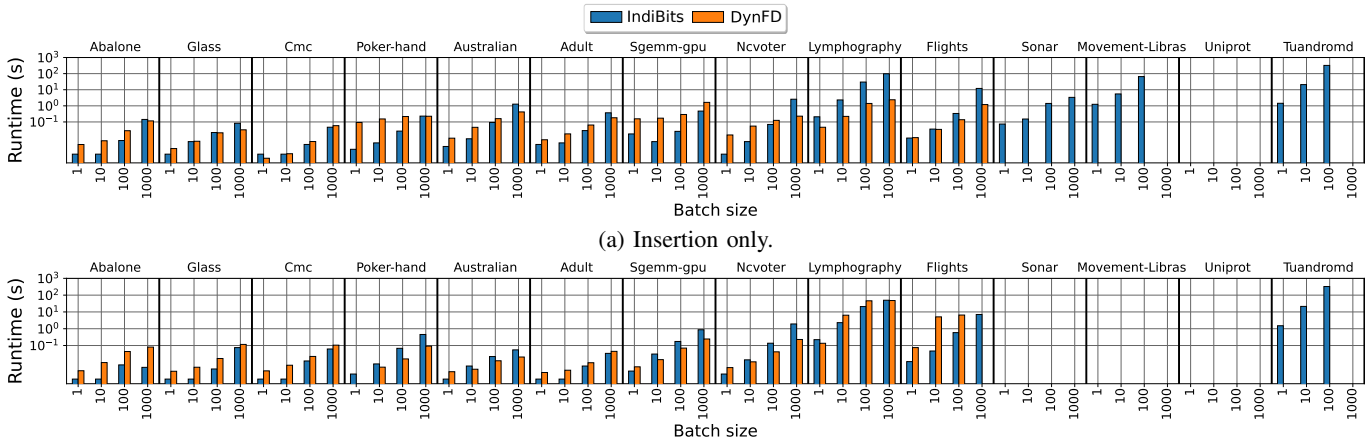


Fig. 6: Speedup with respect to DiMε.



(a) Insertion only.

(b) Deletion only.

Fig. 7: Runtimes evaluation with respect to DynFD.

same order of magnitude. The gap in execution times is greater for the *Lymphography* dataset, which represented an extremely challenging dataset for IndiBits when the similarity threshold is set to 0. This is probably due to the large amount of FDs with a high number of attributes on the LHS. Furthermore, Fig. 7a highlights that IndiBits is capable of completing the discovery process without reaching the TL also when DynFD exceeded it, as in the case of *Sonar*, *Movement-Libras*, and *Tuandromd* datasets. On the other hand, Fig. 7b highlights that the task of updating FDs after deletion operations resulted more challenging for both algorithms, which on average required more time for completing the discovery process. More specifically, we can notice that the gap between the algorithms in terms of average execution times was reduced with respect to insertion operations, and it seems to be independent of the batch sizes. Finally, both algorithms reached the TL when processing datasets with a higher number of columns, except for the *Tuandromd*, dataset for which IndiBits was able to complete the discovery process for 1, 10, and 100 batch sizes.

Overall, the comparative evaluation proved that in most cases the incremental strategy underlying IndiBits considerably reduced execution times by several orders of magnitude with respect to static discovery algorithms. Although it is not optimized for the discovery of FDs, IndiBits turned out to be competitive with respect to one of the most efficient algorithms for FDs discovery in dynamic scenarios.

VIII. Conclusion

In this paper, we presented IndiBits, an RFD$_c$s discovery algorithm for incremental scenarios. To the best of our knowledge, IndiBits represents the first incremental discovery algorithm for RFD$_c$s. It relies on a novel method for representing similarities between tuple pairs, which permits to efficiently update RFD$_c$s holding at a given time instant, starting from those holding at a previous time instant. Experimental results show that IndiBits considerably reduces execution times for discovering RFD$_c$s with respect to a static discovery algorithm and turns out to be competitive also for the discovery of FDs in dynamic scenarios.

In the future, we would like to extend IndiBits in order to enable the discovery of RFD$_c$s also from data streams. Another interesting issue concerns the possibility of updating RFD$_c$s together with thresholds forming similarity constraints. Finally, we would like to investigate the meaningfulness of the discovered RFD$_c$s in different application domains.

REFERENCES

[1] F. Naumann, "Data profiling revisited," *ACM SIGMOD Record*, vol. 42, no. 4, pp. 40–49, 2014.

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

[3] H. Yao, H. J. Hamilton, and C. J. Butz, "FD_Mine: Discovering functional dependencies in a database using equivalences," in *Proceedings of 2002 IEEE International Conference on Data Mining*, ser. ICDM '02, 2002, pp. 729–732.

[4] L. Caruccio, V. Deufemia, and G. Polese, "Relaxed functional dependencies — A survey of approaches," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 147–165, 2015.

[5] S. Song, F. Gao, R. Huang, and C. Wang, "Data dependencies over big data: A family tree," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 10, pp. 4717–4736, 2022.

[6] R. Hai, C. Quix, and D. Wang, "Relaxed functional dependency discovery in heterogeneous data lakes," in *Conceptual Modeling*, A. H. F. Laender, B. Pernici, E.-P. Lim, and J. P. M. de Oliveira, Eds. Cham: Springer International Publishing, 2019, pp. 225–239.

[7] S. Song, A. Zhang, L. Chen, and J. Wang, "Enriching data imputation with extensive similarity neighbors," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1286–1297, 2015.

[8] B. Breve, L. Caruccio, V. Deufemia, and G. Polese, "RENUVER: A missing value imputation algorithm based on relaxed functional dependencies," in *Proceedings of the 25th International Conference on Extending Database Technology*, ser. EDBT '22. OpenProceedings.org, 2022, pp. 1:52–1:64.

[9] L. Caruccio, V. Deufemia, F. Naumann, and G. Polese, "Discovering relaxed functional dependencies based on multi-attribute dominance," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 9, pp. 3212–3228, 2021.

[10] L. Caruccio, S. Cirillo, V. Deufemia, and G. Polese, "Incremental discovery of functional dependencies with a bit-vector algorithm," in *Proceedings of the 27th Italian Symposium on Advanced Database Systems*, ser. CEUR Workshop Proceedings, vol. 2400. CEUR-WS.org, 2019.

[11] M. Khayati, A. Lerner, Z. Tymchenko, and P. Cudré-Mauroux, "Mind the gap: An experimental evaluation of imputation of missing values techniques in time series," *Proceedings of the VLDB Endowment*, vol. 13, no. 5, pp. 768–782, 2020.

[12] H. Mannila and K.-J. Raiha, "Dependency inference," in *Proceedings of the 13th International Conference on Very Large Data Bases*, ser. VLDB '87, 1987, pp. 155–158.

[13] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen, "Efficient discovery of functional and approximate dependencies using partitions," in *Proceedings 14th IEEE International Conference on Data Engineering*, ser. ICDE '98. IEEE, 1998, pp. 392–401.

[14] N. Novelli and R. Cicchetti, "Fun: An efficient algorithm for mining functional and embedded dependencies," in *Proceedings of 8th International Conference Database Theory*, ser. ICDT '01, 2001, pp. 189–203.

[15] Z. Abedjan, P. Schulze, and F. Naumann, "DFD: Efficient functional dependency discovery," in *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*, ser. CIKM '14, 2014, pp. 949–958.

[16] S. Lopes, J.-M. Petit, and L. Lakhal, "Efficient discovery of functional dependencies and Armstrong relations," in *Proceedings of the 7th International Conference on Extending Database Technology*, ser. EDBT '00, 2000, pp. 350–364.

[17] C. Wyss, C. Giannella, and E. Robertson, "FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances," in *Proceedings of International Conference on Data Warehousing and Knowledge Discovery*, ser. DaWaK '01, 2001, pp. 101–110.

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

[19] T. Papenbrock and F. Naumann, "A hybrid approach to functional dependency discovery," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '16. ACM, 2016, pp. 821–833.

[20] C. Giannella and E. Robertson, "On approximation measures for functional dependencies," *Information Systems*, vol. 29, no. 6, pp. 483–507, 2004.

[21] J. Kivinen and H. Mannila, "Approximate inference of functional dependencies from relations," *Theoretical Computer Science*, vol. 149, no. 1, pp. 129–149, 1995.

[22] R. King and J. Oil, "Discovery of functional and approximate functional dependencies in relational databases," *Journal of Applied Mathematics and Decision Sciences*, vol. 7, no. 1, pp. 49–59, 2003.

[23] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga, "CORDS: Automatic discovery of correlations and soft functional dependencies," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '04, 2004, pp. 647–658.

[24] S. Kruse and F. Naumann, "Efficient discovery of approximate dependencies," *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 759–772, 2018.

[25] L. Caruccio, V. Deufemia, and G. Polese, "Mining relaxed functional dependencies from data," *Data Mining and Knowledge Discovery*, vol. 34, no. 2, pp. 443–477, 2020.

[26] ——, "Evolutionary mining of relaxed dependencies from big data collections," in *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '17, 2017, pp. 1–10.

[27] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 5, pp. 683–698, 2010.

[28] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *Proceedings of 23rd IEEE International Conference on Data Engineering*, ser. ICDE '07. IEEE, 2007, pp. 746–755.

[29] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 376–390, 2008.

[30] S.-L. Wang, J.-W. Shen, and T.-P. Hong, "Incremental discovery of functional dependencies using partitions," in *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, vol. 3. IEEE, 2001, pp. 1322–1326.

[31] P. Schirmer, T. Papenbrock, S. Kruse, D. Hempfing, T. Meyer, D. Neuschafer-Rube, and F. Naumann, "DynFD: Functional dependency discovery in dynamic datasets," in *Proceedings of the 22nd International Conference on Extending Database Technology*, ser. EDBT '19, 2019, pp. 253–264.

[32] S. M. Fakhrahmad, M. Sadreddini, and M. Z. Jahromi, "AD-Miner: A new incremental method for discovery of minimal approximate dependencies using logical operations," *Intelligent Data Analysis*, vol. 12, no. 6, pp. 607–619, 2008.

[33] L. Caruccio and S. Cirillo, "Incremental discovery of imprecise functional dependencies," *Journal of Data and Information Quality*, vol. 12, no. 4, pp. 1–25, 2020.

[34] 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," *Proceedings of the VLDB Endowment*, vol. 8, no. 10, pp. 1082–1093, 2015.