Decentralized and Incremental Discovery of Relaxed Functional Dependencies Using Bitwise Similarity

Bernardo Breve, Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, *Member, IEEE*, and Giuseppe Polese, *Member, IEEE*

***Abstract*—Over the past decade, there have been numerous extensions to the definition of Functional Dependency (FD), culminating in the introduction of Relaxed Functional Dependency (RFD), offering more flexible constraints compared to traditional FDs. This increased flexibility makes RFDs well-suited for exploring and profiling data in datasets with lower data quality. However, efficiently identifying RFDs within dynamic data sources presents a significant challenge, as it requires processing an entire dataset from scratch whenever modifications occur. To tackle this problem, incremental discovery algorithms have been defined, but they often suffer when the frequency and the size of batches of updates increase. This article presents a new algorithm, namely D-INDIBITS, relying on a new decentralized architecture to balance the workload that drives the incremental discovery process of INDIBITS, which is based on bitwise operators for computing attribute similarities. Experiments demonstrate D-INDIBITS's effectiveness compared to FD and RFD discovery algorithms on both static and dynamic real-world data. With batches of modifications of sizes 10 k and 100 k, D-INDIBITS is capable of updating the set of RFDs in a few seconds, whereas all other approaches often employ more than 3 hours.**

***Index Terms*—Data profiling, relaxed functional dependencies, incremental scenarios, bitwise similarities.**

I. INTRODUCTION

THE availability of new types of Big Data sources, which dynamically change over time, has brought a completely innovative vision of data. However, in order to make data-intensive processes effective over dynamic data sources, it is necessary to guarantee a consistent characterization of data. This involves accommodating data errors and tracking their evolution over time. In this context, data profiling tasks play a pivotal role in examining datasets and generating metadata [1]. Among the metadata present in the literature, researchers have defined relaxed functional dependencies (RFDs) as extensions of the canonical functional dependency (FD) [2], [3]. RFDs represent data relationships relying on approximate matching paradigms (RFDs relaxing on the attribute comparison, RFD_cs for short), or tolerating possible violations for a limited number of tuples, according to an error measure (RFDs relaxing on the extent, RFD_es for short).

The importance of considering this kind of relationships is supported by their usefulness in several application contexts, where the constraints expressed by the canonical definition of FD make their application impractical for obtaining a correct representation of data, especially those in which the data are collected from heterogeneous sources [4]. 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 [5], [6]. To provide these approaches with the required set of RFDs, discovery algorithms have been proposed to automatically infer them 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 [7]. However, due to the dynamic nature of many real-world datasets, it is necessary to adapt discovery processes in order to make them capable of updating the set of discovered metadata whenever the dataset changes. Furthermore, incremental discovery processes are also demanded in emerging application domains for which the literature is questioning the best strategies and solutions to be adopted when dealing with dynamic scenarios [8], [9].

For instance, companies involved in resource planning or supply chain management may update data related to inventories, shipments, or resource availability several times in a day. This entails the application of batches of tuple changes of considerable sizes, which might alter the set of valid RFDs, hence invalidating many RFDs-based data management processes, such as those mentioned above. Thus, the set of holding RFDs should reflect the new data, but as shown in [10], even with incremental RFDs discovery processes this might require many hours, and new batches or updates might occur before the end of the process. As a consequence, in these contexts, it is necessary to have RFDs discovery processes capable of processing big batches of changes in relatively short times. To this aim, in this paper we present D-INDIBITS, a decentralized version of INDIBITS [10] that considerably speeds up the incremental update of the set of holding RFDs for bigger batches of tuple changes. D-INDIBITS relies on the same data structures underlying INDIBITS, i.e., a binary representation to characterize data similarities and track updates through bitwise operations, but it decentralizes both the pre-processing and the discovering steps through a computation strategy based on the MapReduce model. Both such steps are

Manuscript received 31 August 2023; revised 30 March 2024; accepted 11 May 2024. Date of publication 21 May 2024; date of current version 13 November 2024. This work was supported in part by Project SERICS (PE00000014) under the NRRP MUR Program funded by the EU - NGEU. Recommended for acceptance by L. Chen. *(Corresponding author: Stefano Cirillo.)*

The authors are with the Department of Computer Science, University of Salerno, 84084 Fisciano, Italy (e-mail: bbreve@unisa.it; lcaruccio@unisa.it; scirillo@unisa.it; deufemia@unisa.it; gpolese@unisa.it).

Digital Object Identifier 10.1109/TKDE.2024.3403928

managed by coordination processes balancing the workload according to the characteristics of the attributes of datasets.

The main novelties of this work with respect to [10] are:

- Extending the pre-processing and the discovering steps to work in a decentralized way, still guaranteeing the correctness and minimality of extracted RFD$_c$s.
- Introducing several coordination processes to suitably split the computations, and combine the results.
- Demonstrating through extensive experiments that D-INDIBITS outperforms existing approaches when dealing with larger datasets and significant batch 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 D-INDIBITS, whereas their underlying data representation method is presented in Section V. Then, Section VI details the validation method underlying D-INDIBITS, and the procedures to handle insertion and deletion data modifications. Section VII reports experimental results to analyze the effectiveness of D-INDIBITS on real-world datasets, also when compared with other FD and RFD$_c$ discovery algorithms. Finally, conclusions and future directions are included in Section VIII.

II. RELATED WORK

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 1980 s, when the first discovery algorithms were defined [11], [12]. 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 [12], 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 [13], 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 FSC [16], DepMiner [17], FastFD [18], and FDep [19]. Finally, in order to achieve good performances in high-dimensional datasets with many rows, the discovery algorithms HyFD [20] and DHyFD [21] combine 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$_e$s are validated through either a condition or a coverage measure [22], such as the $g3$-error [23], or the error measure of super keys [24], which permit to identify the portion of the relation instance on which an FD is not valid. In general, RFD$_e$s discovery algorithms rely on sampling [23], [25], [26], which consists in validating candidate RFD$_e$s on samples of tuples, meaning that they hold on a relation instance with a given probability. Instead, RFD$_c$s generalize the definition of FDs by introducing similarity- or distance- based comparisons between attribute values [2]. One of the most recent algorithms for RFD$_c$ discovery is DOMINO [7], which relies on the concept of dominance to automatically derive thresholds of attributes to compare their values through similarity functions. Lastly, in [27] 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$_c$s and RFD$_e$s is presented, whereas in [28] 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) [29]. 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 [30]. Examples of CFD discovery algorithms include CFDMiner, CTANE, FastCFD [29], and the greedy algorithm proposed in [31].

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 [32]. 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 [33], which extends the HyFD algorithm [20] by enabling the discovery and update of FDs in dynamic datasets. Finally, the author in [34] recently presented an approach, called DynASt, for the efficient maintenance of agree-sets to be exploited by discovery processes to manage dynamic scenarios.

In the context of RFD discovery, incremental algorithms for RFD$_e$s are AD-MINER [35] and BIRD [36]. The former exploits logical operations to automatically infer new RFD$_e$s upon the insertion of new tuples, whereas the latter exploits data partitions to continuously update the set of holding RFD$_e$s upon the insertion of new batches of data.

To the best of our knowledge, INDIBITS was the first solution for the incremental discovery of RFD$_c$s [10]. In fact, incremental algorithms existing in the literature are either suited for FDs [20], [32], [33] or RFD$_e$s [35], [36]. The former cannot be used to also discover RFD$_c$s, since they do not consider similarity constraints between attribute values. On the other hand, approaches for the incremental discovery of RFD$_e$s address a different problem with respect to the discovery of RFD$_c$s. Finally, employing static solutions for RFD$_c$s discovery [7], [27] 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. Regardless of the approach, large datasets are even more demanding in terms of computational resources and execution time, especially when considering significant batch updates. This has motivated the design of D-INDIBITS, which, in contrast,

TABLE I
A SUMMARY OF SYMBOLS USED THROUGHOUT THE PAPER

Symbol	Description
$\mathcal{R}$	Relational database schema
R	Relation schema
r	Relation instance
r_τ	Relation instance at time τ
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
$\|r\|$	Number of tuples in r
$rfds_\tau$	Set of minimal RFD$_c$s at time τ
$rfds_\tau[B]$	Set of minimal RFD$_c$s at time τ having the attribute B on the RHS

allows such batch updates to be managed while remaining within reasonable execution times due to its decentralized architecture.

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 [2]. Table I summarizes the notations used throughout the paper.

A similarity constraint is a predicate evaluating whether the distance or similarity between two values falls within predefined intervals. More formally, let r be an 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 threshold. For the 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 $i \in [1, h]$ ($j \in [1, k]$, 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, in all examples, we consider a more compact notation for the constraints, which are specified for each attribute, and directly show the comparison operator and the associated threshold.

TABLE II
A SNIPPET OF THE TRIAGE DATASET

	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

Example 1: In the snippet of the Triage dataset shown in Table II, 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. This is a kind of relationship that can be expressed by an RFD$_c$.

$$\varphi_1 : \textit{chiefcomplaint}_{(\leqslant 5)}, \textit{heartrate}_{(\leqslant 10)} \rightarrow \textit{acuity}_{(\leqslant 0)}$$

where under the similarity constraints expressed in the form $Levenshtein(chiefcomplaint1, chiefcomplaint2) \leqslant 5$ and $abs(heartrate1, heartrate2) \leqslant 10$, the RFD$_c$ establishes that whenever two tuples have the Levenshtein distance less than or equal to the threshold value 5 between attribute values of ***chiefcomplaint*** and the absolute difference is less than or equal to the threshold value 10 between attribute values of ***heartrate***, than the attribute values of ***acuity*** must be equal, i.e., $abs(acuity1, acuity2) \leqslant 0$. With respect to the dataset snippet, the constraints on the left-hand-side (LHS) of the RFD are only satisfied by the tuple pair (t_6, t_7), which also satisfies the equality of values on the right-hand-side (RHS).

One of the most important characteristics of an RFD$_c$ is its 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 for each attribute a fixed constraint is 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.

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.

Example 2: Let us consider the snippet of the Triage dataset shown in Table II and the RFD$_c$ φ_1 identified in the previous Example that describes the correlation between the values of the attributes ***heartrate*** and ***chiefcomplaint*** with those of attribute ***acuity***. We can state that the RFD$_c$ φ_1 holds on the considered snippet, but it is not minimal since the RFD$_c$s:

$$\varphi_2 : \textit{chiefcomplaint}_{(\leqslant 5)} \rightarrow \textit{acuity}_{(\leqslant 0)}$$

is also valid on the considered snippet of data. This means that the values of the attribute *chiefcomplaint* are related to those of attribute *acuity* with a certain degree of tolerance.

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 RFD$_c$s 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 involved in an RFD$_c$ [37]. Nevertheless, different from the equality, the similarity does not satisfy the transitivity property, preventing the possibility of adopting the validation methods of FDs. This leads to the necessity of conceiving new validation methods for evaluating candidate RFD$_c$s.

IV. The Proposed RFD Discovery Strategies

IncBits 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.

IncBits monitors changes that 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 sets of change operations.

The decentralized version of IncBits, namely D-IncBits, applies vertical decompositions over the attributes. This approach delegates the tasks related to RFD$_c$s discovery to multiple decentralized components, making them suitable for execution within parallelization and distribution scenarios.

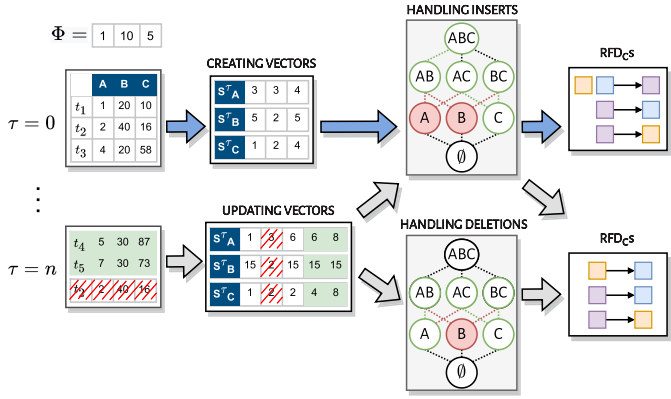


Fig. 1. Overview of the process underlying IncBits.

In this section, we first provide an overview of the process underlying IncBits and then we present the decentralized architecture enabling the execution of pre-processing and discovery steps on multiple components.

A. IncBits Discovery Process

An overview of the discovery process underlying IncBits is shown in Fig. 1. As we can see, IncBits 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. The definition of the thresholds generally depends on the nature of the analyzed data and is not necessarily linked to the distribution or statistics of the data. The optimal value for thresholds is generally specified by the designer or can be evaluated empirically based on the application context of the RFD$_c$s. By manually setting thresholds, IncBits deals with this issue independently, making it applicable across various domains.

Starting from the thresholds defined as input, it performs the discovery process by considering only insertion operations and extracts the set of holding RFD$_c$s. Then, for each batch, IncBits 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 IncBits 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 IncBits 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

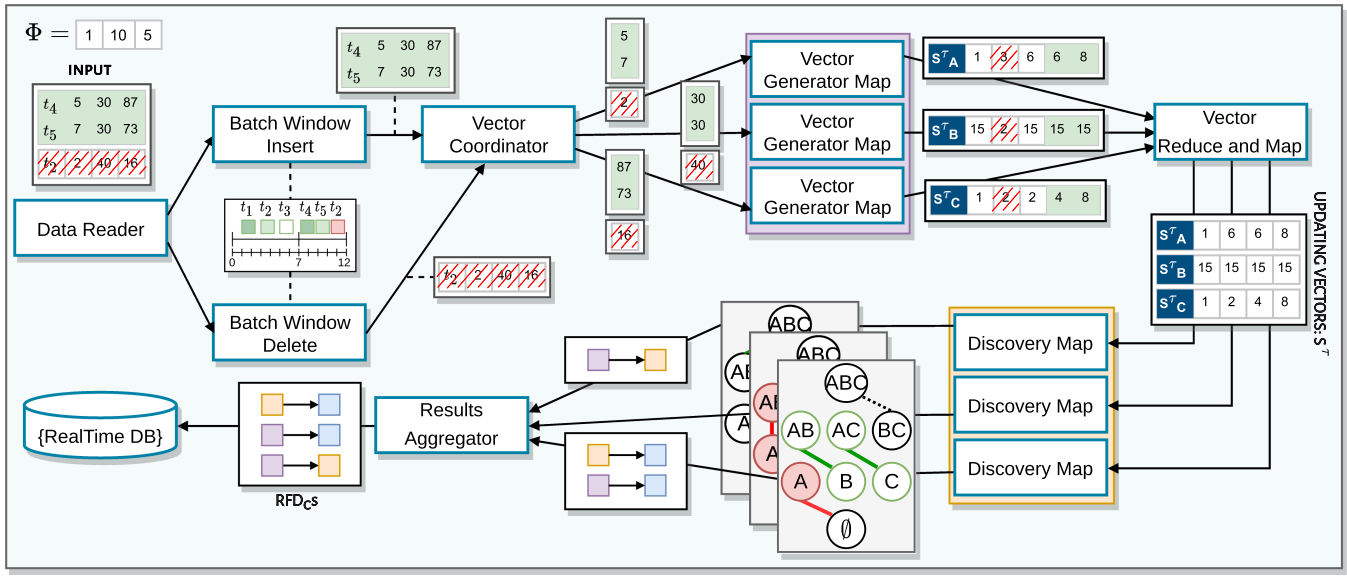


Fig. 2. Overview of the decentralized and destructured architecture of D-INDIBITS at a generic time τ.

the similarity constraints defined on the attributes X''. INDIBITS first processes all the deletions contained 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$_c$, INDIBITS relies on a compact representation of data similarities (see Section V), from which an efficient validation method has been defined (see Section VI-A).

B. D-INDIBITS: Decentralized INDIBITS

INDIBITS, as introduced in [10], adopts a monolithic architecture that does not allow for leveraging component independence to decompose the underlying processes. To address this limitation, we have re-designed the architecture of INDIBITS by decomposing each component to harness the potential of the MapReduce model. Fig. 2 shows an overview of the decentralized and destructured version of INDIBITS, which we now term D-INDIBITS.

The first component, namely *Data Reader*, reads at time τ a batch of tuples containing changes to be applied to the original dataset. Its role involves managing the distribution of these tuples to subsequent components based on the nature of changes within the batch—specifically, whether they involve insertion or deletion.

Upon reading the data changes, the *Batch Window Insert* and *Batch Window Delete* components allow D-INDIBITS to process changes at time τ in the dataset independently. These components rely on a sliding window mechanism capable of grouping tuples into a single batch based on a maximum change count set during the algorithm's configuration stage, i.e., the batch size. In particular, *Batch Window Insert* and *Batch Window Delete* receive from *Data Reader* components only the tuples relating to insertions and deletions, respectively. After they receive a number of changes equal to the batch size, they will emit these updates to the *Vector Coordinator* component.

Upon receiving the new batch of tuples from the previous components, the *Vector Coordinator* takes charge of allocating computations by assigning at each worker a projection of the tuples onto a single attribute $G \in attr(R)$, i.e., the *Vector Generator Map* component. This allocation ensures that each worker consistently manages changes related to the same attribute projections, which are assigned during the first time instant. According to this strategy, each worker generates the representation of similarities S^{τ}_{G} related to G each time it receives a new batch of tuples. This division adheres to the MapReduce model based on the maximum worker count defined during configuration.

The *Vector Reduce and Map* component handles the Reduce phase, waiting until all prior workers conclude their computations of the binary representations S^{τ}_{G}. This component relies on a control and caching mechanism, which temporarily retains data structures received from earlier workers until an equal number of tuples is received for each attribute projection. The *Vector Reduce and Map* is also responsible for coordinating the execution of the discovery step. As we can see from Fig. 2, this component distributes the collection of all S^{τ}_{G} computed from previous stages, which we now term S^{τ}, to each worker in charge of executing the discovery step, namely *Discovery Map*. Moreover, at each *Discovery Map* worker, the *Vector Reduce and Map* assigns an attribute G for the discovery step while simultaneously communicating the updated representation S^{τ}. The attribute assignment allows the *Discovery Map* worker to exclusively handle the extraction and validation of RFD$_c$s where attribute G is on the RHS. This approach enables the exploration of the search space concurrently in different parts,

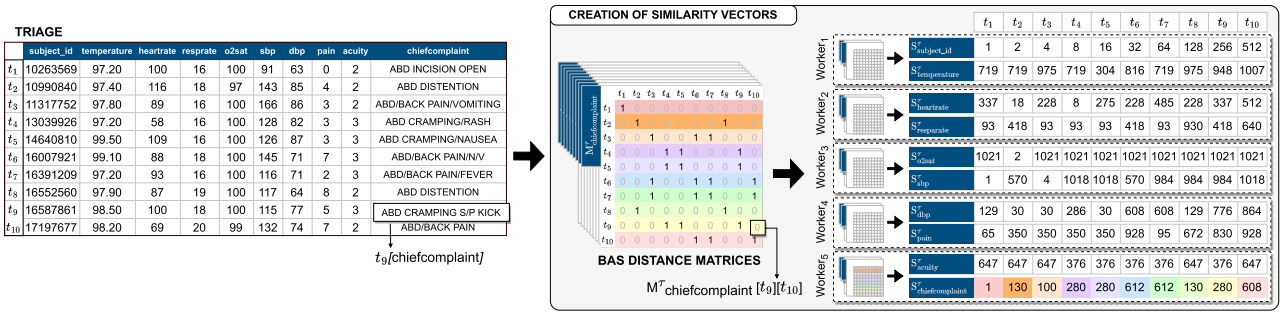


Fig. 3. Similarity Vectors created for the snippet of the *Triage* dataset from *Mimic*-III database.

decentralizing the discovery process across multiple workers. The RFD$_c$s obtained by each worker are collected by a *Results Aggregator* component, which centralizes and stores them in a real-time database.

V. Bitwise Similarity

As mentioned above, we handled the complexity in representing data similarities, also according to possible data modifications, founded over the usage of ad-hoc data structures, i.e., *similarity vectors* [10]. In particular, similar to INDIBITS, D-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 recall the theoretical concept of Binary Attribute Satisfiability (BAS) Distance Matrix introduced in [10]:

Definition 3 (BAS Distance Matrix and Vector): Let r_τ 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_B^τ, is a triangular matrix defined as:

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

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 $M_B^\tau[t_k]$, whose values are $\langle M_B^\tau[t_k][t_1], \ldots, M_B^\tau[t_k][t_k], M_B^\tau[t_{k+1}][t_k], \ldots, M_B^\tau[t_{|r|}][t_k]\rangle$.

A BAS Distance Vector $M_B^\tau[t_k]$ describes the tuples similar to t_k on attribute B. As an example, $M^\tau_{chiefcomplaint}[t_2]$ in Fig. 3 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 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_τ 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 3: Fig. 3 provides an overview of the process for creating similarity vectors from the snippet of the *Triage* dataset considering $\{0, 1, 10, 1, 1, 20, 5, 2, 0, 8\}$ as thresholds. The vectors are constructed balancing the workload among different workers, according to the new decentralized architecture underlying D-INDIBITS. The outputs provided by each worker are then merged following the strategy discussed in Section IV-B.

The Similarity Vectors S^τ summarize the similarities between the tuples of the input relation, providing a lightweight representation that can be efficiently updated upon changes. In particular, both the insertion and deletion of tuples yield the necessity to update the similarity vectors for the validation process. Different from INDIBITS, D-INDIBITS shares updates with all workers to allow each of them to update the similarity vectors they are responsible for. More specifically, the insertion of new tuples requires computing the similarity of each new tuple 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 introduced in [10].

Definition 5 (BAS Matrix Upon Tuple Insertion): Let r_τ 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:

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

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 S^τ. 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.

Concerning deletion, 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

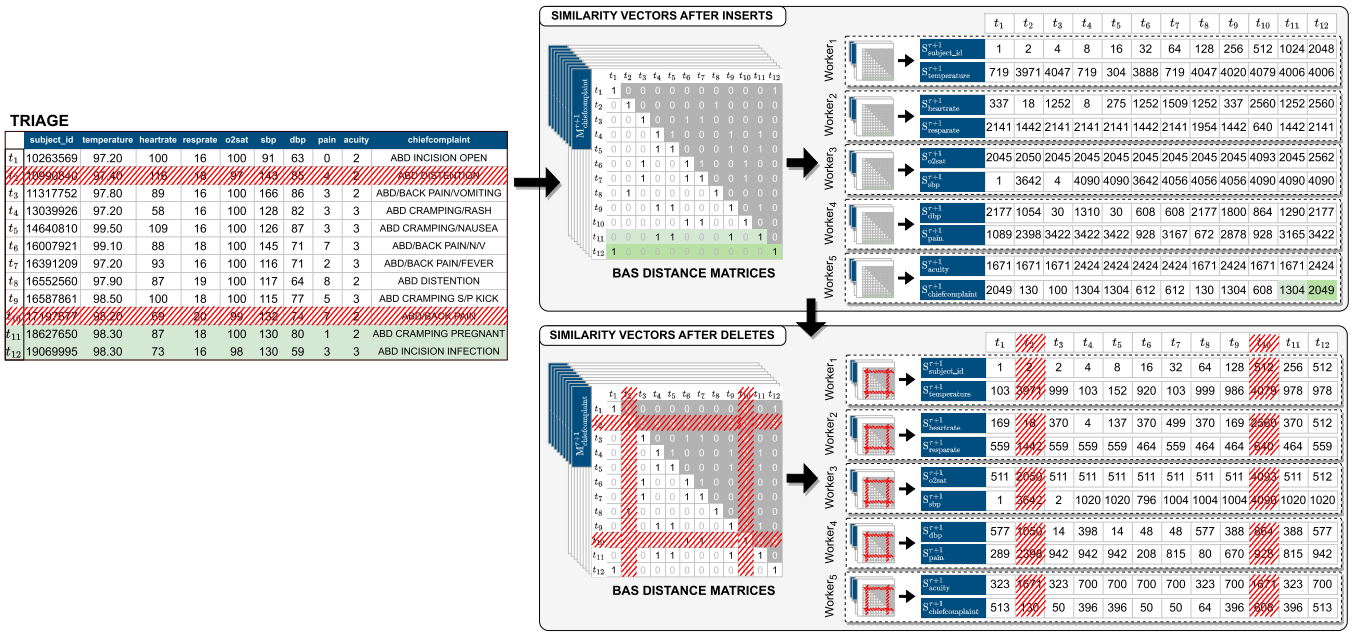


Fig. 4. Updating vectors after the insertion and deletion of tuples over the snippet of the *Triage* dataset.

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

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: Fig. 4 shows the results achieved by the insertion of the tuples t_{11} and t_{12}, which led to the creation of two columns and rows to the BAS matrices of each attribute. The changes in the dataset are shared with each worker to enable them to update the similarity vectors for which they are responsible. The insertion of new tuples leds to the definition of new values in each similarity vector. For instance, let us consider the attribute *chiefcomplaint*, the new values $1304_{(10)}$ and $2049_{(10)}$ correspond to the decimal representation of $10100011000_{(2)}$ and $100000000001_{(2)}$, respectively. The deletion of tuples t_2 and t_{10} has led to a drop of two rows/columns from the BAS Matrices of each attribute. This means that each worker drops two values from the similarity vectors and updates the remaining values accordingly. Let us consider the BAS matrix of the attribute *chiefcomplaint*, after the deletion of 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.

VI. Incremental RFD$_c$ Discovery

Section V presented the lightweight data structure underlying INDIBITS and D-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 [27].

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. By computing the similarity vectors for any attribute set, it is possible to count the number of tuples that are similar to each tuple t_k by counting the number of bits having

value 1 in $S^\tau_X[t_k]$, according to the following formula:

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

where $|S^\tau_X[t_k]|$ is the number of bits equal to 1 in the binary representation of $S^\tau_X[t_k]$, and represents the number of similar tuples in S^τ_X. 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. As we have seen, every BAS matrix is symmetric because the similarity function δ respects the commutative property, i.e., $M_B[t_i][t_j] = M_B[t_j][t_i]$ for each pair of tuples $(t_i, t_j) \in r$ with $i \leqslant j$. Moreover, for each tuple $t_i \in r$, we can consider $M_B[t_i][t_i] = 1$ in each matrix, since $\delta(t_i[B], t_i[B]) = 0$. These enable us to not consider the diagonal of the matrix, which represents the similarity of each value with itself, and consider only one time the similarities between pairs of tuples.

Since for an attribute set $X \cup A \supset X$, we can say that $S^\tau_{X \cup A}$ always refines S^τ_X, it is possible to state that $||S^\tau_{X \cup A}||$ is always lower than or equal to $||S^\tau_X||$. However, given an RFD$_c$ $\varphi : X_{\Phi_1} \rightarrow A_{\phi_2}$, if $||S^\tau_{X \cup A}||$ is lower than $||S^\tau_X||$, 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 $||S^\tau_{X \cup A}|| = ||S^\tau_X||$.

Example 5: According to the similarity vectors shown in Fig. 3, the following RFD$_c$ is valid:

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

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

$$||S^\tau_X|| = \frac{0 + 0 + 1 + 2 + 2 + 1 + 1 + 0 + 2 + 1}{2} = ||S^\tau_{X \cup A}||$$

B. Handling Insertions

In this section, we discuss the strategies to maintain RFD$_c$s upon the insertion of tuples according to the new decentralized and destructured architecture underlying D-INDIBITS. The main procedure of D-INDIBITS for handling changes relies on the procedures described in [10]. However, as discussed in Section IV-B, the new architecture has required dividing the vector update and discovery processes in different components that work independently relying on the MapReduce model.

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, D-INDIBITS considers as starting points the minimal RFD$_c$s validated on a relation r at time τ. Different from INDIBITS that maintains a single centralized set of RFD$_c$s, D-INDIBITS ensures that each discovery component only maintains a smaller set of RFD$_c$s with the same attribute B on the RHS. In this way, each component keeps track of a smaller subset of RFD$_c$s to use as starting points upon data changes, enabling each component of D-INDIBITS to avoid re-executing the discovery process from scratch. During the first execution of the algorithm, 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, D-INDIBITS performs the validation from RFD$_c$s with the lowest number of the attributes on the LHS to those with the highest one, i.e., from the most general to the most specialized RFD$_c$s, according to the validation strategy discussed in Section VI-A. Differently from the version of INDIBITS proposed in [10] which reduces the search space according to the pruning strategy proposed in [12] whenever an RFD$_c$ is valid, D-INDIBITS does not involve this pruning rule. 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 $X \cap A = \varnothing$, the pruning rules states that for each attribute $B \in attr(R)$, with $B \notin \{X \cup A\}$, then $X \cup A \rightarrow B$ is not minimal at time $\tau + 1$. However, since the discovery components of D-INDIBITS have been conceived for maintaining updated only a subset of RFD$_c$s with the same attribute A_{Φ_2} on the RHS, such a pruning strategy would require a synchronization strategy among all discovery components, reducing their independence and affecting the discovery performances of D-INDIBITS.

On the other hand, if there exists at least one candidate RFD$_c$ that is no longer valid at time $\tau + 1$, D-INDIBITS specializes it and generates new candidate RFD$_c$s. To ensure the minimality of the resulting RFD$_c$s at time $\tau + 1$, before analyzing each newly specialized RFD$_c$, D-INDIBITS checks if there exists at least one valid RFD$_c$ at time $\tau + 1$ that generalizes it. If so, the specialization is valid and not minimal RFD$_c$, since an RFD$_c$ that generalizes it has already been 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 $\tau + 1$, a new tuple t inserted at time $\tau + 1$, an attribute B for which the component must update the associated vector S^τ_B. The procedure first reads the size of the stored tuples, which corresponds to the id of the newly inserted tuple t (Line 2). Then, it updates the similarity vectors S^τ_B associated with B according to the strategy defined in Section V (Lines 3-7). More specifically, the procedure compares the value of the new tuple over the attribute B with those

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; An attribute B; A similarity vector S^τ_B associated with B
OUTPUT: Updated similarity vector $S^{\tau+1}_B$

```
1: function UPDATEVECTORINSERT(B)
2:     id ← |r|
3:     S^τ_B[t_id] ← 2^id
4:     for k ∈ 0 ... |r| do
5:         if δ(t[B], t_k[B]) satisfies φ[B] then
6:             S^τ_B[t_k] ← S^τ_B[t_k] + 2^id
7:             S^τ_B[t_id] ← S^τ_B[t_id] + 2^k
8:     S^{τ+1}_B ← S^τ_B
9:     return S^{τ+1}_B
```

Algorithm 2: Procedures for Checking and Pruning.

```
1:  prunedRfds ← ∅
2:  function PRUNEDRFDS(φ)
3:      rhs ← φ.getRhs()
4:      lhs ← (φ.getLhs() ∪ rhs)
5:      for B ∈ attr(R)\lhs do
6:          prunedRfds.add(lhs → B)
7:  function ISPRUNED(φ)
8:      if not prunedRfds.isEmpty() then
9:          rhs ← φ.getRhs()
10:         for each φ' ∈ prunedRfds.getRfds(rhs) do
11:             if φ'.isGeneralizations(φ) then
12:                 return true                    ▷ φ is pruned
13:     return false                               ▷ φ is not pruned
```

Algorithm 3: Main Procedure of D-INDIBITS After Inserting Tuples.

INPUT: An attribute B; A set $rfds_\tau[B]$ of minimal RFD$_c$s with th attribute B on the RHS; A set $invalidRfds_\tau[B]$ of RFD$_c$s not valid at time τ with the attribute B on the RHS; Similarity vectors $S^{\tau+}$ at time $\tau+1$

OUTPUT: Updated $rfds_{\tau+1}[B]$, Updated $invalidRfds_{\tau+1}[B]$

```
1:  function DISCOVERYONINSERT(B)
2:      for level ∈ 0...m do
3:          rfdsInvalidated ← ∅
4:          for φ ∈ rfds_τ[B].getLevel(level) do
5:              if not validate(φ, S^{τ+1}) then
6:                  rfdsInvalidated ← rfdsInvalidated ∪ {φ}
7:                  invalidRfds_τ[B] ← invalidRfds_τ[B] ∪ {φ}
8:          for each φ ∈ rfdsInvalidated do
9:              rfds_τ ← rfds_τ\{φ}
10:             spec ← specialize(φ)
11:             for each φ' ∈ spec do
12:                 if not rfds_τ[B].containsGeneralizations(φ') then
13:                     rfds_τ[B] ← rfds_τ[B] ∪ {φ'}
14:     rfds_{τ+1}[B] ← rfds_τ[B]
15:     invalidRfds_{τ+1}[B] ← invalidRfds_τ[B]
16:     return rfds_{τ+1}[B], invalidRfds_{τ+1}[B]
```

of the tuples t_k already processed (Lines 5-7). 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 vector $S^{\tau+1}_B$ associated with B at time $\tau+1$ (Lines 8-9).

Algorithm 2 shows the procedures for pruning RFD$_c$s (i.e., *pruneRfds*) and for checking if a candidate RFD$_c$ is pruned (i.e., *isPruned*). It considers a global set of all pruned RFD$_c$s (i.e., *prunedRfds*) that is updated whenever a valid RFD$_c$ is discovered (Line 1). In particular, the first procedure aims to update the set of pruned RFD$_c$s considering a valid RFD$_c$ φ as input. Starting from φ, it first extracts the attribute on the RHS and then defines a new set of attributes composed of the ones on the RHS and LHS of φ (Lines 2-4). This new set of attributes represents the LHS of the RFD$_c$s to prune. For each attribute that is not contained on the LHS, the procedure adds to *prunedRfds* the RFD$_c$s composed by the new LHS defined and all the other attributes on the RHS, according to the pruning strategy defined in [12] (Lines 5-6).

The second procedure, i.e., *isPruned*, checks if an RFD$_c$ φ has been pruned from other RFD$_c$s already validated (Line 7). It first checks if there are some RFD$_c$s already pruned (Line 8) and if true, extracts the attribute on the RHS of φ (Line 9). The attribute on the RHS allows the procedure to optimize the search of pruned RFD$_c$s by filtering only those that have the same attribute on the RHS of φ. For each RFD$_c$ φ' contained in *prunedRfds*, the procedure checks if φ' is a generalization of φ, and if so, it means that φ has been pruned (Lines 10-12). Otherwise, the procedure returns *false* (Line 13).

Algorithm 3 shows the main procedure of D-INDIBITS executed for every batch of new tuples inserted at time $\tau+1$. Differently from INDIBITS, the discovery procedure of D-INDIBITS aims to update the subset of RFD$_c$s valid at time τ having B on the RHS, i.e., $rfds_\tau[B]$. More specifically, it considers an attribute B, the set of all RFD$_c$s holding at time τ having the attribute B on the RHS ($rfds_\tau[B]$), the set of all RFD$_c$s that were not valid at time τ with B on the RHS ($invalidRfds_\tau[B]$), 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 2). Then, the procedure validates each candidate RFD$_c$ according to the approach described in Section VI-A (Lines 4-5). Thus, if the analyzed RFD$_c$ is not valid at time $\tau+1$ it is added to the set of invalid RFD$_c$s (Lines 5-7). Then, the procedure removes the invalid RFD$_c$s from $rfds_\tau[B]$, and it generates their specializations (Lines 9-10). 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 at time $\tau+1$ (Lines 11-13). 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 $\tau+1$ having the attribute B on the RHS (Lines 14-16).

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 $\tau+1$. Similar to insertion operations, the main procedure of D-INDIBITS for handling changes relies on the procedures described in [10]. However, as discussed in Section IV-B, D-INDIBITS divides the updating of the vectors and the discovery among different components that work independently with the MapReduce model. In particular, D-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. Because each discovery component maintains only a subset of RFD$_c$s with the same attribute on the RHS, the discovery process generalizes each RFD$_c$ with the same

attribute B on the RHS 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, D-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 $\tau+1$, D-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, D-INDIBITS generalizes $X_{\Phi_1} \rightarrow A_{\Phi_2}$, by introducing all the direct generalizations $\{X_{\Phi_1} \backslash C\} \rightarrow A_{\Phi_2}$ for each $C \in X$ 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.

Example 6: 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}, the algorithm 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 4 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, a tuple t^- deleted at time $\tau+1$, an attribute B for which the component must update the associated vector S_B^τ. The procedure first retrieves the id of the tuple to be deleted from r (Line 2), which corresponds to the id of the tuple assigned during its insertion. Then, for the similarity vector S_B^τ and for each tuple t_k it updates the values in $S_B^\tau[t_k]$ by removing the bit corresponding to the tuple identifier id (Lines 3-8). In particular, for each value $S_B^\tau[t_k]$, 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 $S_B^\tau[t_k]$ (Line 8). After performing this operation over all tuples, the procedure returns the similarity vectors of the attribute B updated after the deletion operation (Lines 9-10).

Algorithm 5 shows the main procedure of D-INDIBITS executed for every batch of tuples deleted at time τ. Similar to the main procedure for insertion operations (Algorithm 3), the discovery procedure of D-INDIBITS upon deletion of tuples aims to update the subset of RFD$_c$s valid at time τ having B on the RHS. In particular, the procedure considers an attribute B, the set of all RFD$_c$s holding at time τ having B on the RHS ($rfds_\tau[B]$), the set of all RFD$_c$s that were not valid at time τ ($invalidRfds_\tau[B]$), and the similarity vectors updated after the deletion of the tuples at time $\tau+1$ ($S^{\tau+1}$). The procedure

Algorithm 4: Updating Vectors After Deleting a Tuple.

INPUT: A relation r at time τ; A tuple t^- deleted at time $\tau+1$; An attribute B; A similarity vector S_B^τ associated with B
OUTPUT: Updated similarity vector $S_B^{\tau+1}$

```
1:  function UpdateVectorDelete(B)
2:      id ← getTupleId(t⁻)
3:      for k ∈ 0 ... |r| do
4:          if k ≠ id then
5:              v1 ← getBits(S_B^τ[t_k], 0, id − 1)
6:              v2 ← getBits(S_B^τ[t_k], id + 1, |r| − 1) ≫ 1
7:              v ← v1.concat(v2)
8:              S_B^τ[t_k] ← convertBinaryNumber(v)
9:      S_B^{τ+1} ← S_B^τ
10:     return S_B^{τ+1}
```

Algorithm 5: Main Procedure of D-INDIBITS After Deleting Tuples.

INPUT: An attribute B; A set $rfds_\tau[B]$ of minimal RFD$_c$s with the attribute B on the RHS; A set $invalidRfds_\tau[B]$ of RFD$_c$s not valid at time τ with the attribute B on the RHS; Similarity vectors $S^{\tau+1}$
OUTPUT: Updated $rfds_{\tau+1}[B]$, Updated $invalidRfds_{\tau+1}[B]$

```
1:  function DiscoveryOnDelete(B)
2:      if S^{τ+1}.isEmpty() then
3:          rfds_{τ+1} ← ∅
4:          invalidRfds_{τ+1} ← ∅
5:      else
6:          validRfds ← ∅
7:          for level ∈ m ... 1 do
8:              for φ ∈ invalidRfds_τ[B].getLevel(level) do
9:                  if validate(φ, S^{τ+1}) then
10:                     invalidRfds_τ ← invalidRfds_τ[B] ∪ generalize(φ)
11:                     invalidRfds_τ[B] ← invalidRfds_τ[B]\{φ}
12:                     validRfds ← validRfds ∪ {φ}
13:             for φ ∈ validRfds do
14:                 if not rfds_τ[B].containsGeneralizations(φ) then
15:                     rfds_τ[B].addAndCheckMinimality(φ)
16:     rfds_{τ+1}[B] ← rfds_τ[B]
17:     invalidRfds_{τ+1}[B] ← invalidRfds_τ[B]
18:     return rfds_{τ+1}[B], invalidRfds_{τ+1}[B]
```

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 2-4). 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 5-12). Then, it validates each candidate RFD$_c$ according to the approach described in Section VI-A (Line 9). 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 10-12). Then, D-INDIBITS checks if the new valid RFD$_c$s 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 13-15). 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 16-18).

D. Coordination of the Discovery Process and Vector Updating

After discussing the details of the vector update procedures and discovery process underlying D-INDIBITS, it is necessary to provide details on the processes for coordinating decentralized components. As introduced in Section IV-B, D-INDIBITS relies on the MapReduce model which allows dividing the vector update and discovery processes in several independent components that work simultaneously. However, these processes required the definition of new components capable of coordinating the processes and aggregating results.

Algorithm 6 shows the map and reduce procedures defined to coordinate the updating of vectors and RFD$_c$s. For the sake of clarity, we consider only a single tuple t received at time $\tau+1$, but the approach can process different tuples at a time. In particular, the function VECTORCOORDINATOR aims to divide the process of updating vectors into different tasks, assigning a task to each component according to an attribute B. The procedure starts by considering a tuple t processed at time $\tau+1$ (Line 1) and for each attribute B in $attr(R)$, selects a component (i.e., a worker) W for assigning the task of updating the vector S_B^τ based on the value of t for B (Lines 3-4). The method *getWorkerForAttribute* is responsible for assigning to each worker the task of always updating the same vectors S_B^τ using the function VECTORGENERATORMAP. The latter represents the Map procedure for updating similarity vectors. In particular, the VECTORGENERATORMAP considers an attribute B and a tuple t and first checks the type of changes that D-INDIBITS are processing, i.e., insertion or deletion (Lines 5-6). If the operation is an insertion, the function updates S_B^τ following the procedure defined in Algorithm 1 (Lines 6-7). Otherwise, it executes the function for updating S_B^τ after deleting a tuple, i.e., Algorithm 4 (Lines 8–9).

After the mapping step, the function VECTORREDUCERANDMAP is responsible for collecting the updated vectors from the map procedures (Lines 10-16). In particular, it first waits for all the map procedures to be completed, i.e., for all the vectors to have been updated upon new data changes (Line 13). After that, the function collects from the results the vectors associated with each attribute $C \in attr(R)$ updated at time $\tau+1$ (Lines 14-16).

The function VECTORREDUCERANDMAP is also responsible for coordinating the map procedures of the discovery step of D-INDIBITS. In fact, after collecting $S_B^{\tau+1}$, the function assigns a discovery task to different components according to a single attribute C (Lines 18-23). Similar to the VECTORCOORDINATOR function, it assigns each worker a single discovery task at a time, i.e., limiting the discovery for each worker to only RFD$_c$s with a single attribute on the RHS. The resulting RFD$_c$s from each worker will be collected from the RESULTAGGREGATOR function, which waits for all the discovery procedures to be completed (Lines 24-27) and stores the resulting RFD$_c$s obtained after processing data changes (Lines 28-30).

Algorithm 6: Coordination Procedures Underlying D-INDIBITS.

INPUT: A tuple t at time $\tau+1$

1: **function** VECTORCOORDINATOR(t)
2: **for each** $B \in attr(R)$ **do**
3: W←*getWorkerForAttribute*(B)
4: ASSIGNTASK(W, VECTORGENERATORMAP(B, t))
5: **function** VECTORGENERATORMAP(B, t)
6: **if** isIsertion(t) **then**
7: **return** B, UPDATEVECTORINSERT(B, t)
8: **else**
9: **return** B, UPDATEVECTORDELETE(B, t)
10: **function** VECTORREDUCERANDMAP(B, *updatedVectors*)
11: $S^{\tau+1} \leftarrow \varnothing$
12: ▷ Reduce Step
13: *waitAllWorkers()*
14: **for each** $C \in attr(R)$ **do**
15: $S_C^{\tau+1} \leftarrow$ *updatedVectors.getVectorObject*(C)
16: $S^{\tau+1} \leftarrow S^{\tau+1} \cup S_C^{\tau+1}$
17: ▷ Discovery Map Step
18: **for each** $C \in attr(R)$ **do**
19: W←*getWorkerForAttribute*(C)
20: **if** *isInsertion*(t) **then**
21: ASSIGNTASK(W, DISCOVERYONINSERT(C))
22: **else**
23: ASSIGNTASK(W, DISCOVERYONDELETE(C))
24: **function** RESULTAGGREGATOR(B, *discoveryResults*)
25: $rfds_{\tau+1} \leftarrow \varnothing$
26: $invalidRfds_{\tau+1} \leftarrow \varnothing$
27: *waitAllDiscoveryWorkers()*
28: **for each** $C \in attr(R)$ **do**
29: $rfds_{\tau+1}$←*discoveryResults.getRfds*(C)
30: $invalidRfds_{\tau+1}$←*discoveryResults.getInvalidRfds*(C)

E. Theoretical Analysis

The proofs presented for INDIBITS in [10] remain valid also for the decentralized version of INDIBITS, i.e., D-INDIBITS. In fact, for what concerns correctness, during the discovery process, all components consider a complete, up-to-date, representation of similarity vectors, resulting from the reduce phase performed in the *Partition/Vector Reduce*. Furthermore, regarding the minimality, each component of the *Discovery Map* is configured to deal with all possible candidate RFD$_c$s having the same RHS attribute. Thus, as the property of minimality requires to consider all and only candidate RFD$_c$s having the same attribute on the RHS (see Definition 2), this also leads to the decentralized architecture of D-INDIBITS to be compatible with the proof of minimality.

VII. EXPERIMENTAL EVALUATION

In this section, we first present the results achieved from INDIBITS, by extending the previous evaluation presented in [10]. Then, we discuss experimental results concerning the performances of D-INDIBITS and INDIBITS in terms of discovery results and execution time. Moreover, we compare the results of D-INDIBITS with those of DiMε[1] [27], an RFD discovery algorithm for static scenarios, and of DYNFD[2] [33], an FD discovery algorithm for dynamic scenarios.

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

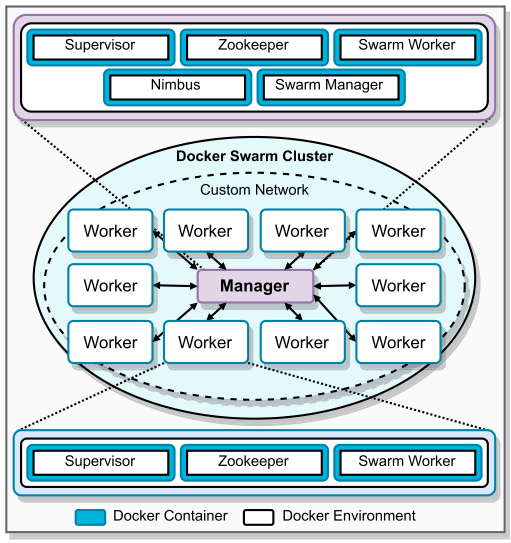


Fig. 5. Environment configuration for D-INDIBITS.

A. Experimental Setup

We implemented D-INDIBITS and 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, the algorithms compute similarity vectors according to user-defined distance thresholds. In every evaluation concerning RFD discovery algorithms, we performed 5 experiments, using the same threshold ε for all of their attributes, i.e., 0, 1, 2, 4, and 8, and 4 different batch sizes (e.g., the number of changes, in terms of insertion and/or deletion operations, to be considered at each time instant), i.e., $1, 10, 100$, and 1,000. Concerning the evaluation with D-INDIBITS, we have considered two larger batch sizes, i.e., 10,000, and 100,000, and the same thresholds defined above. Finally, we considered a time limit (TL) of 3 hours.

All the experiments of INDIBITS have been executed on an iMac Pro with an Intel Xeon CPU at 3.20 GHz, 18-core, and 128 GB of memory, running macOS Mojave and OpenJDK 17 as Java environment. D-INDIBITS has required configuring an ad-hoc environment capable of executing the version of the discovery algorithm relying on the decentralized architecture. Fig. 5 shows an overview of the environment for D-INDIBITS. As we can see, we have designed it considering different *Workers*, all coordinated by a single worker *Manager*. The entire environment relies on a microservice architecture, where each component of the environment is a Docker container,[3] deployed in a Docker Swarm Cluster.[4]

The Manager component consists of 5 different containers: a *Zookeeper* container that exposes a service used by a group of components and coordinates the distribution of the work between all zookeeper servers through robust synchronization techniques; a *Nimbus* container that is responsible for the execution of D-INDIBITS on all the workers in the environment; a *Supervisor* container that is the component following the instructions provided by the Nimbus; a *Swarm Manager* container that is responsible to coordinate all the nodes in the environment, and a *Swarm Worker* container that aims to execute worker containers in the shared environment. The Worker components consist of 3 different containers, a *Zookeeper*, a *Swarm Worker*, and a *Superivor* container. All the components rely on the Apache Storm framework,[5] which enables D-INDIBITS to combine the decentralized architecture with a pipeline processing, in order to make all the processes more efficient and reduce the waiting times between components. Through this environment, it is possible to execute D-INDIBITS, by maximizing the decomposition of the components.

Dataset Details: The experiments were performed on 21 real-world datasets,[6] previously used for evaluating FD and RFD discovery algorithms. Table III shows the characteristics of the datasets involved in the evaluation. These datasets have different numbers of rows, and columns, and contain data from different domains, which enable us to evaluate the applicability of the proposed approach across different scenarios. Indeed, some of the datasets contain data collected from real sensors, such as *Sonar* to *Sgemm-gpu*, while others originate from the healthcare domain, such as *Cmc* and *Lymphography*, and still others are commonly used in predictive scenarios, such as *Iris*, *Poker-hand*, or *Movement-Libras*. As illustrated in Table III, most datasets contain both categorical and numeric attributes, with only a few containing numeric attributes. This diversity allows us to examine how algorithm performance varies based on these parameters.

B. Performances on Real-World Datasets

Our first experiment measured the number of discovered RFD_cs (see Table III), the execution times, and the memory consumption of INDIBITS algorithm on the considered real-world datasets. Since these datasets have not been designed to be employed in incremental discovery processes, we simulated an incremental scenario in which we first insert all tuples, and then randomly delete 90% of them.

The number of discovered RFD_cs at the end of insertion and deletion operations is detailed in Table III. Concerning the insertion operations, we observe that the number of RFD_cs is not necessarily larger when the distance threshold increases. In fact, often a huge number of RFD_cs is obtained with threshold 0, i.e., when considering canonical FDs, such as in the case *Abalone*, *Glass*, *Australian*, *Lymphography*, *Ncvoter*, *Biodegradation*, *Sonar*, and *Movement-Libras* datasets; whereas for *Cmc* and *Sgemm-gpu* datasets the number of RFD_cs 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_cs is due to the fact that when only a few FDs hold, the algorithm is capable to catch significant

[3] www.docker.com
[4] www.docs.docker.com/engine/swarm/
[5] www.storm.apache.org
[6] www.github.com/DastLab/TestDataset

TABLE III
DETAILS OF THE CONSIDERED REAL-WORLD DATASETS AND NUMBER OF HOLDING RFD_cS

Dataset	#Cols	#Rows	#RFD_cs										Number of Attributes per Type	
			Insertion					Deletion						
			$\varepsilon = 0$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	$\varepsilon = 0$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	Categorical	Numerical
Iris	5	150	4	0	2	13	20	9	0	9	13	50	1	4
Har70	8	200,000	279	242	56	56	56	586	456	56	56	56	1	7
Tic-tac-toe	9	958	9	72	72	72	72	9	72	72	72	72	9	0
Yeast	9	1,484	37	64	64	72	72	70	64	64	72	72	1	8
Abalone	9	4,177	137	34	56	64	64	191	34	56	64	64	1	8
Bitcoin	9	400,000	245	198	72	72	72	440	224	72	72	72	1	8
Glass	10	214	124	27	22	38	73	168	34	28	38	73	0	10
Cmc	10	1,473	1	27	36	72	72	3	27	36	72	72	0	10
Poker-hand	11	1,025,010	1	0	0	50	50	1	0	0	50	50	0	11
Australian	14	690	535	71	78	78	91	2,352	88	78	78	91	0	14
Adult	15	32,561	78	81	63	38	33	361	738	292	111	104	9	6
Sgemm-gpu	18	241,600	4	68	68	68	119	4	68	68	68	119	0	19
Lymphography	19	148	2,730	342	342	342	308	86,558	342	342	342	308	0	19
Ncvoter	19	1,001	775	387	256	237	181	599	472	355	276	208	17	2
Flights	38	1,000	1,904	1,562	1,340	1,059	1,115	3,495	3,416	1,365	1,063	1,145	17	21
Biodegradation	41	1,055	40,655	1,824	921	680	929	52,043	2,243	1,058	826	1,134	0	41
Sonar	60	208	75,086	3,540	3,540	3,540	3,540	7,462	3,540	3,540	3,540	3,540	0	60
Movement-Libras	91	360	348,215	5,787	5,787	5,787	5,787	436,576	5,787	5,787	5,787	5,787	0	91
Dota2-Games	116	100,000	TL	TL	TL	7,350	7,413	TL	TL	TL	10,452	10,793	0	116
Uniprot	223	1001	TL	TL	616,187	1,473,270	1,180,725	TL	TL	TL	4,747,193	3,974,670	219	4
Tuandromd	242	4,465	9,189	15,602	15,602	15,666	15,666	10,920	15,602	15,602	15,666	15,666	1	241

relationships among data, based on the relaxation of the attribute comparison constraints. However, few considerations can be made on the number of RFD_cs discovered for the *Dota2-Games* and *Uniprot* datasets, since both D-INDIBITS and INDIBITS have reached the time limit for lower similarity thresholds, while it was able to discover many RFD_cs with larger thresholds.

Concerning the deletion operations, the number of RFD_cs holding on the considered datasets is typically greater than the number of discovered RFD_cs upon the insertion operations. Exceptions hold for *Tic-tac-toe*, *Poker-hand*, *Sgemm-gpu*, and *Sonar*, and in part for the *Yeast*, *Movement-Libras* and *Tuandromd* datasets. For both types of operations, the number of RFD_cs 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_cs, but characterized by a low number of attributes on their LHSs as the comparison threshold increases.

Concerning time and memory performances achieved by INDIBITS, we report the average runtimes and memory peaks in Fig. 6, by grouping the results according to batch sizes and distance thresholds. In particular, INDIBITS almost always required less than 10^4 MB of memory, except for some configurations of *Har70*, *Bitcoin*, *Poker-hand*, *Adult*, *Sgemm-gpu*, *Movement-Libras*, *Dota2-Games*, *Uniprot*, and *Tuandromd*, in which the resulting memory peaks never exceed 10^5 MB. In general, the low memory consumption of INDIBITS is mainly due to the lightweight representation of data and distances that make the memory requirements not severely affected by the dimensionality of datasets and the number of holding RFD_cs.

Overall, we can notice that runtimes are quite stable or slightly grow when the batch size increases. Moreover, almost always the average times are less than 10 sec, except for some configurations of *Lymphography*, *Flights*, *Biodegradation*, *Movement-Libras*, *Uniprot*, and *Tuandromd* datasets. The last four datasets also exceeded the TL in some configurations. Although such datasets represent the four 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 did not exceed 10^3 ms (i.e., 1 sec). Instead, concerning the *Dota2-Games* and *Uniprot* datasets, INDIBITS completed the discovery process with the two highest attribute comparison thresholds, considering batches of size 1, 10, and 100 for *Uniprot*, and for *Dota2-Games* with threshold 8 on batch size 10 and with threshold 4 on batch size 100. On the other hand, by considering *Har70*, *Bitcoin*, *Poker-hand*, *Adult*, and *Sgemm-gpu* representing the most challenging datasets in terms of rows, INDIBITS seems not to be affected in runtimes. In fact, although such datasets are characterized by a large number of tuples to be processed, the average runtimes are comparable to the ones of smaller datasets, such as *Cmc* or *Australian*, suggesting that INDIBITS might have scalability potential over large-size datasets.

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 fact, although for high dimensionality datasets 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 many 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.

C. Comparison Between D-INDIBITS and INDIBITS

In this section, we present the experimental results concerning the performances of D-INDIBITS and INDIBITS in the pre-processing and discovery steps, considering 0, 1, 2, 4, and 8 thresholds for all the attributes and 1, 10, 100, 1,000, 10,000, and 100,000 as batch sizes. The two larger batch sizes allowed us to compare the performances of both algorithms when considering significant batch updates at each time instant.

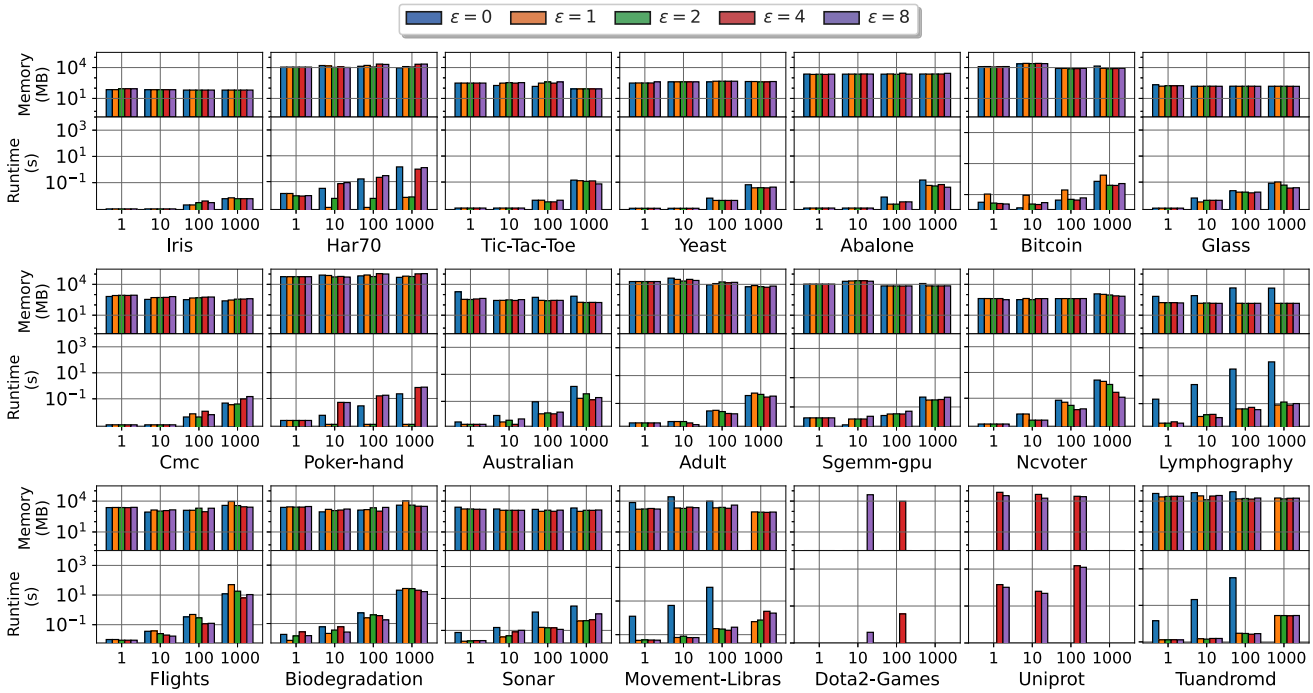


Fig. 6. Performances of INDIBITS over real-world datasets.

1) Pre-Processing Evaluation: In this experimental evaluation, we measured the performances of INDIBITS and D-INDIBITS for computing similarity vectors. In particular, we considered the 6 larger datasets in terms of the number of rows, i.e., *Har70*, *Bitcoin*, *Poker-hand*, *Adult*, *Sgemm-gpu*, and *Dota2-Games*. Table III shows the characteristics of the datasets involved in our evaluation.

Fig. 7 shows the performances of both INDIBITS and D-INDIBITS for a batch size of 10,000. The blue and the yellow lines represent the average runtimes achieved by both algorithms to compute similarity vectors. Specifically, each point results from averaging runtimes between the current processed batch with respect to all the previous ones. As we can see, D-INDIBITS consistently achieves lower times than INDIBITS as the number of changes increases. This improvement can be attributed to the decentralized architecture of D-INDIBITS, enabling individual computation and updates of similarity vectors through the MapReduce model. As a result, D-INDIBITS reduced the runtimes to compute and update similarity vectors by approximately an order of magnitude for each dataset under consideration.

Concerning the computation of the similarity vectors for a batch size of 100,000, similar to the previous results, the average execution times of D-INDIBITS are always faster than INDIBITS (Fig. 8). Notice that, the considered batch size is greater than and equal to the number of rows in the *Adult* and *Dota2-Games* datasets, respectively. Therefore, the evaluation was conducted by starting from a small sample of rows within the original dataset, upon which changes were applied in accordance with the specified batch size.

As a general insight, from Figs. 7 and 8, it is evident that the trends of average times for both INDIBITS and D-INDIBITS are similar across all analyzed datasets and batch sizes. D-INDIBITS improves the average performances of the computation of the vectors of about 1 order of magnitude for each considered dataset and for both sizes of updating. Although these improvements do not seem to be so evident, it is necessary to consider that D-INDIBITS preserves the specific strategy for comparing values underlying INDIBITS, destructuring the overall vector calculation into different parts. This led to an overall performance improvement in D-INDIBITS preprocessing allowing us to process large batches efficiently.

2) Discovery Evaluation: We conducted a comparison between INDIBITS and D-INDIBITS based on the time required to accomplish the discovery step. In this experimental assessment, we continue to utilize the 6 datasets with the highest number of rows, maintaining the same configuration as previously detailed.

Fig. 9 illustrates the results of the comparison between the two versions of INDIBITS by showing the achieved runtimes in terms of speedup. This analysis revealed that INDIBITS can achieve similar, and at times better, results compared to D-INDIBITS in configurations involving smaller batch sizes. Specifically, for batch sizes of 1, 10, 100, and 1,000, the outcomes of both versions on the *Har70* dataset tend to be very close, with only a few exceptions. Notably, INDIBITS outperforms its decentralized version for certain scenarios, such as *Poker-hand* with a batch size of 1, *Adult* with batch sizes of 1 and 10, and *Sgemm-gpu* and *Bitcoin* with a batch size of 10. This disparity is likely attributed to the fact that, for smaller batch sizes, aggregating

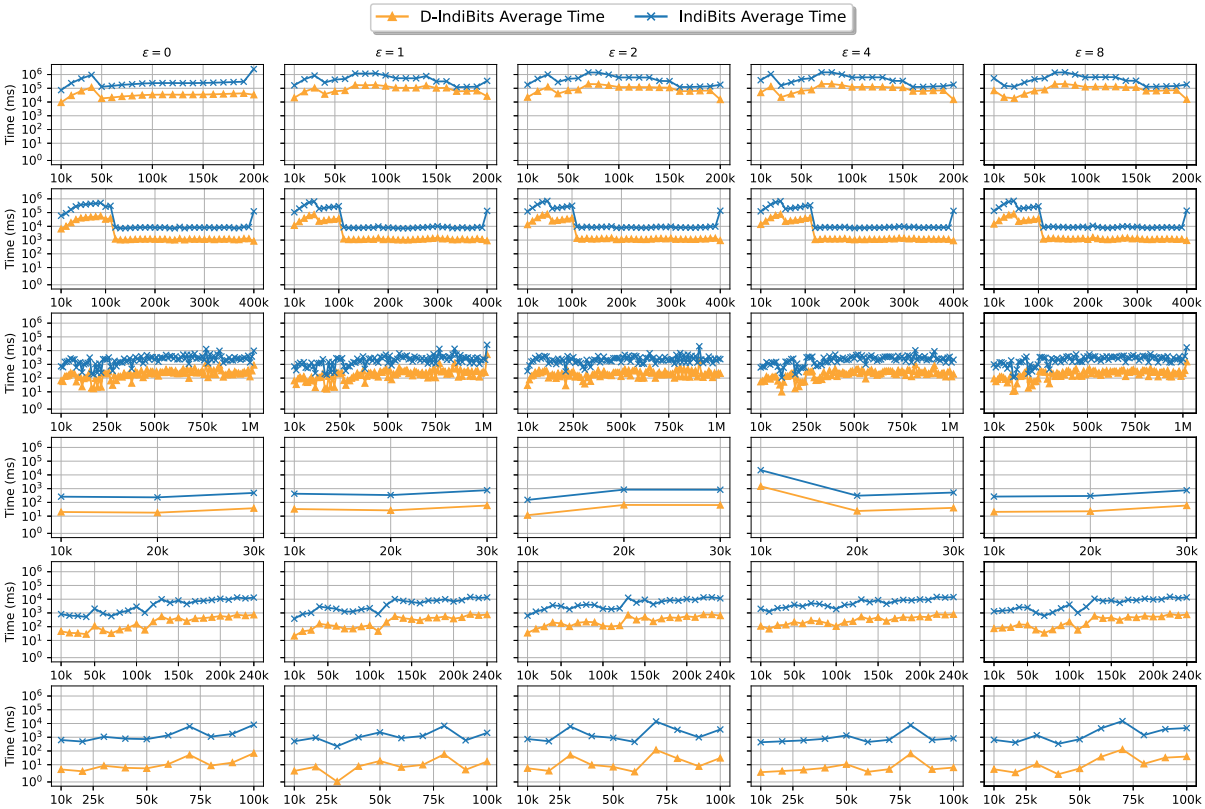


Fig. 7. Results of the pre-processing of INDIBITS and D-INDIBITS considering a batch size of 10,000 rows.

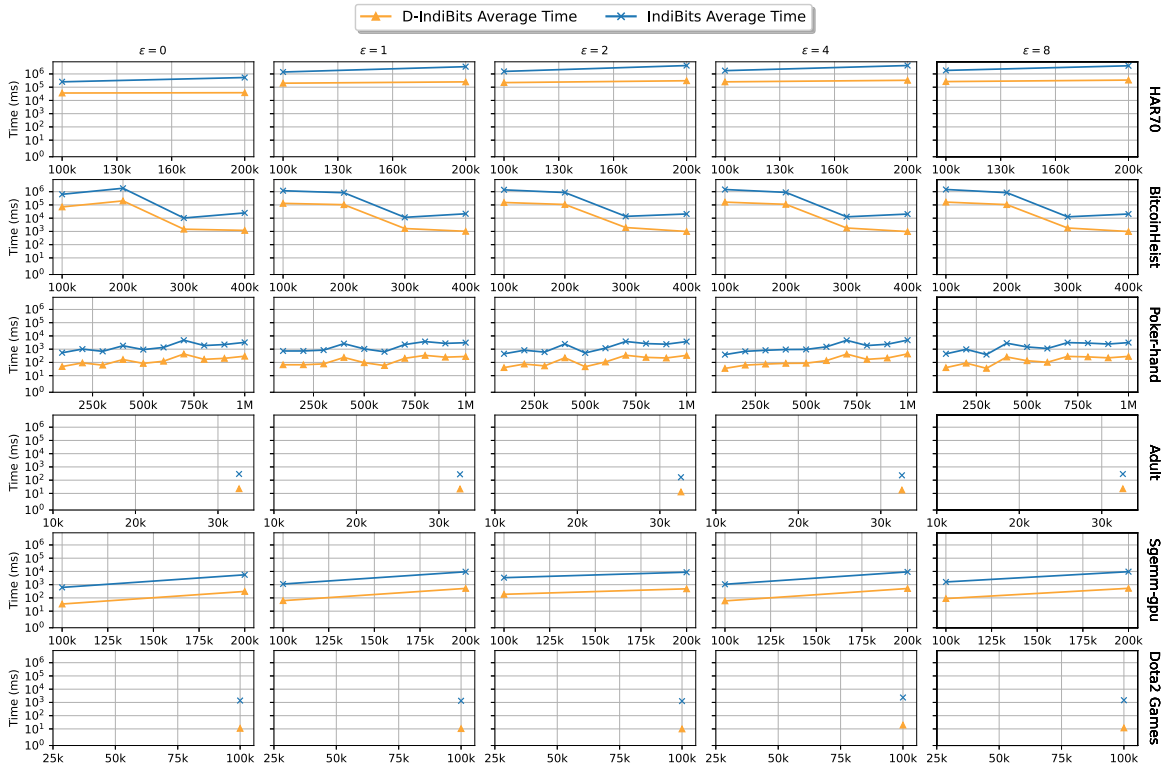


Fig. 8. Results of the pre-processing of INDIBITS and D-INDIBITS considering a batch size of 100,000 rows.

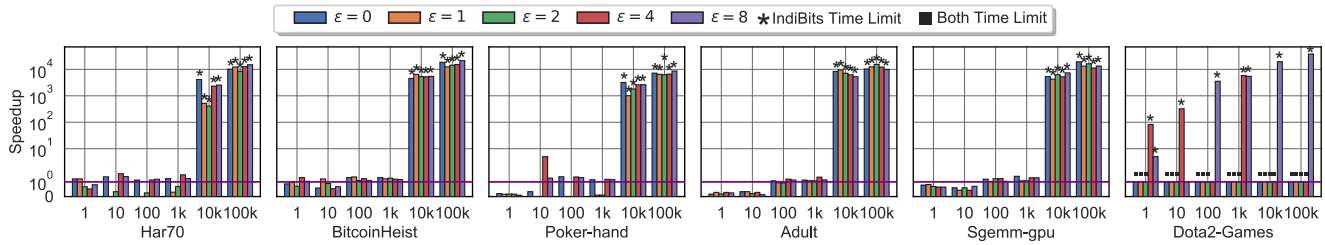


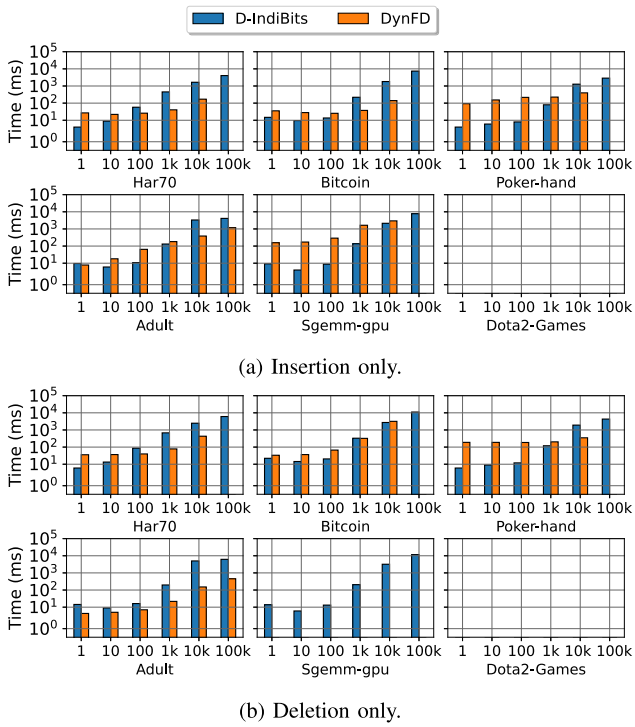
Fig. 9. Speedup of D-INDIBITS with respect to INDIBITS.

results in accordance with the MapReduce model, is more time-consuming than processing an entire batch sequentially. However, this outcome is reversed when larger batch sizes are taken into account. Specifically, across all setups involving batch sizes of 10,000 and 100,000, INDIBITS reached the TL for all considered thresholds, resulting in an inability to complete the whole discovery process. Conversely, D-INDIBITS managed to successfully conclude the entire process, showcasing significant speedup values, as indicated by the speedup bars reaching magnitudes in the order of 10^4. A case in point, however, is the speedup for the *Dota2-Games* dataset. In fact, the size of this dataset made it difficult for both D-INDIBITS and INDIBITS, to succeed in completing the discovery process. In fact, the two versions reached the TL in different configurations of both thresholds and batch sizes. Once again, there are specific configurations in which D-INDIBITS succeeded in completing the discovery process when high thresholds are considered. This is due to the fact that, in such cases, the number of RFD_cs reached a stable state as the threshold caused the flattening of distance distributions, leading to RFD_cs being validated at the lowest levels of the search space. The same type of motivation also applies in the two cases of thresholds 8 and 4 on batches of size 10 and 100, respectively, where both INDIBITS and D-INDIBITS managed to complete the discovery processes with very close runtimes. Overall, D-INDIBITS has proven capable of outperforming INDIBITS in both the preprocessing and discovery steps, especially for critical scenarios where large batches of operations need to be processed. The new decentralized architecture has demonstrated the potential to handle large amounts of changes using multiple components, overcoming some of the technical limitations underlying INDIBITS.

D. Comparative Evaluation of D-INDIBITS Against DiMϵ and DynFD

As a third experiment, we compared the performances of D-INDIBITS to those of DiMε [27] and DYNFD [33].

DiM ε is a static algorithm relying on a column-based strategy to discover RFDs relaxing on the attribute comparison (RFD_c s), and/or the extent (RFD_es). In this experiment, we focus on the discovery of RFD_cs by analyzing in which conditions D-INDIBITS under- or out-performs DiMε. To this end, we gradually scale up the size of the dataset according to batch sizes, each time executing DiMε on an increased dataset. Then, we plot the average runtimes of D-INDIBITS against those of DiMε, by considering the speedup measure. A speed-up of 10 indicates that D-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 star on top. Instead, when both algorithms reach the TL, the speed-up bar is marked with a black square on top.



(a) Insertion only.

(b) Deletion only.

Fig. 10. Runtimes evaluation with respect to DYNFD.

Fig. 11 shows the results of the comparative evaluation on the largest datasets in terms of number of rows. We can notice that D-INDIBITS is almost always faster than DiMε for all datasets and all configurations, especially on the *Har70*, *Bitcoin*, *Poker-hand*, *Adult*, and *Sgemm-gpu* datasets, where D-INDIBITS has been more than 1,000 times faster than DiMε in all configurations. Concerning *Dota2-Games*, which represents the largest dataset in terms of rows, we can see that only D-INDIBITS has completed the discovery process with all batches when considering the higher similarity thresholds, i.e., 4, 8. However, both DiMε and D-INDIBITS reached the time limit

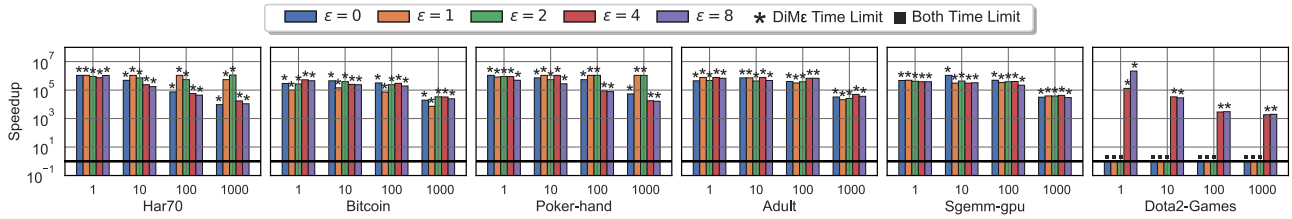


Fig. 11. Speedup of D-INDIBITS with respect to DIMε.

for the lower thresholds, i.e., 0, 1, 2, and with all batches. This can be due to the fact that D-INDIBITS performs worse when there are many invalidations in successive batches, leading to the discovery process to consider a larger number of candidates to be validated.

Concerning the comparative evaluation between D-INDIBITS and DYNFD, we set up insertion and deletion operations by considering the experimental configuration introduced in Section VII-A, 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$_c$s.

Fig. 10(a) and (b) show the results of the comparative evaluation. Notice that, although D-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 D-INDIBITS outperforms or achieves average execution times similar to DYNFD. In particular, concerning the insertion operations, we notice that D-INDIBITS typically outperforms DYNFD with smaller batches of tuples, i.e., 1, 10, and 100, except for the *Har70* dataset. Moreover, as we can see for batch sizes 1,000 and 10,000, D-INDIBITS outperforms DYNFD on the *Sgemm-gpu* dataset with both sizes and on *Poker-hand* with size 1,000. Concerning the largest batch of tuples, i.e., 100,000, we can notice that both algorithms reach the TL for *Dota2-Games*. However, D-INDIBITS is capable of completing the discovery process without reaching the TL or ML also when DYNFD exceeded them, as in the case of *Har70*, *Bitcoin*, *Poker-hand*, and *Sgemm-gpu* datasets. The gap in execution times is greater for the *Adult* dataset, which represents an extremely challenging dataset for D-INDIBITS when the similarity threshold is set to 0. This is probably due to the considerable amount of FDs with a high number of attributes on the LHS. On the other hand, Fig. 10(b) highlights that the task of updating FDs after deletion operations resulted in more challenges for both algorithms, which on average required more time to complete 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. We observed many cases like for the *Har70*, *Bitcoin*, and *Poker-Hand* with batch sizes of 100,000, and *Sgemm-gpu* with all batch sizes, where only D-INDIBITS was capable of completing the discovery, while DYNFD cannot. Finally, both algorithms reached the TL when processing two of the datasets with the largest number of columns, i.e., *Dota2-Games* and *Uniprot*.

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 article, we presented a new algorithm for the discovery of RFD$_c$s, namely D-INDIBITS, which splits the pre-processing and the discovery steps of the INDIBITS algorithm [10] by means of the MapReduce model, aiming to enhance performances on modification batches of large sizes. Experimental results showed that D-INDIBITS considerably reduces execution times for discovering RFD$_c$s with respect to a static RFD$_c$s discovery algorithm, and turned out to be competitive also with respect to best FD discovery algorithms for dynamic scenarios. Furthermore, D-INDIBITS is capable of updating the set of RFDs with batches of modifications of sizes 10 k and 100 k in a few seconds, whereas all other approaches often employ more than 3 hours.

In the future, we would like to extend D-INDIBITS's architecture to encompass the discovery of RFD$_c$s also from data streams, which represent a dynamic context of great complexity, which might considerably benefit from similar algorithms. For example, in the case of data streams containing sensor data, it is often necessary to correct wrong data items, whereas when using online algorithms on them, RFDs might be useful for explainability purposes, since they can help to detect the features mostly affecting predicted values. Another interesting issue concerns the update of RFD$_c$s together with thresholds forming similarity constraints.

REFERENCES

[1] F. Naumann, "Data profiling revisited," *ACM SIGMOD Rec.*, vol. 42, no. 4, pp. 40–49, 2014.
[2] 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, Jan. 2016.
[3] S. Song, F. Gao, R. Huang, and C. Wang, "Data dependencies over big data: A family tree," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 10, pp. 4717–4736, Oct. 2022.
[4] R. Hai, C. Quix, and D. Wang, "Relaxed functional dependency discovery in heterogeneous data lakes," in *Proc. Int. Conf. Conceptual Model.*, A. H. F. Laender, B. Pernici, E.-P. Lim, and J. P. M. de Oliveira, Eds., Springer, Cham, 2019, pp. 225–239.
[5] S. Song, A. Zhang, L. Chen, and J. Wang, "Enriching data imputation with extensive similarity neighbors," *Proc. VLDB Endowment*, vol. 8, no. 11, pp. 1286–1297, 2015.

[6] B. Breve, L. Caruccio, V. Deufemia, and G. Polese, "RENUVER: A missing value imputation algorithm based on relaxed functional dependencies," in *Proc. 25th Int. Conf. Extending Database Technol.*, 2022, pp. 1:52–1:64.
[7] L. Caruccio, V. Deufemia, F. Naumann, and G. Polese, "Discovering relaxed functional dependencies based on multi-attribute dominance," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 9, pp. 3212–3228, Sep. 2021.
[8] M. Khayati, A. Lerner, Z. Tymchenko, and P. Cudr é-Mauroux, "Mind the gap: An experimental evaluation of imputation of missing values techniques in time series," *Proc. VLDB Endowment*, vol. 13, no. 5, pp. 768–782, 2020.
[9] L. Caruccio, S. Cirillo, V. Deufemia, and G. Polese, "Incremental discovery of functional dependencies with a bit-vector algorithm," in *Proc. 27th Italian Symp. Adv. Database Syst.*, 2019.
[10] B. Breve, L. Caruccio, S. Cirillo, V. Deufemia, and G. Polese, "IndiBits: Incremental discovery of relaxed functional dependencies using bitwise similarity," in *Proc. IEEE 39th Int. Conf. Data Eng.*, 2023, pp. 1393–1405.
[11] H. Mannila and K.-J. Raiha, "Dependency inference," in *Proc. 13th Int. Conf. Very Large Data Bases*, 1987, pp. 155–158.
[12] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.
[13] H. Yao, H. J. Hamilton, and C. J. Butz, "FD_Mine: Discovering functional dependencies in a database using equivalences," in *Proc. IEEE Int. Conf. Data Mining*, 2002, pp. 729–732.
[14] N. Novelli and R. Cicchetti, "FUN: An efficient algorithm for mining functional and embedded dependencies," in *Proc. 8th Int. Conf. Database Theory*, 2001, pp. 189–203.
[15] Z. Abedjan, P. Schulze, and F. Naumann, "DFD: Efficient functional dependency discovery," in *Proc. 23rd ACM Int. Conf. Inf. Knowl. Manage.*, 2014, pp. 949–958.
[16] X. Wan, X. Han, J. Wang, and J. Li, "Efficient discovery of functional dependencies on massive data," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 1, pp. 107–121, Jan. 2024.
[17] S. Lopes, J.-M. Petit, and L. Lakhal, "Efficient discovery of functional dependencies and Armstrong relations," in *Proc. 7th Int. Conf. Extending Database Technol.*, 2000, pp. 350–364.
[18] 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. Data Warehousing Knowl. Discov.*, 2001, pp. 101–110.
[19] P. A. Flach and I. Savnik, "Database dependency discovery: A machine learning approach," *AI Commun.*, vol. 12, pp. 139–160, 1999.
[20] T. Papenbrock and F. Naumann, "A hybrid approach to functional dependency discovery," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 821–833.
[21] Z. Wei and S. Link, "Towards the efficient discovery of meaningful functional dependencies," *Inf. Syst.*, vol. 116, 2023, Art. no. 102224.
[22] C. Giannella and E. Robertson, "On approximation measures for functional dependencies," *Inf. Syst.*, vol. 29, pp. 483–507, 2004.
[23] J. Kivinen and H. Mannila, "Approximate inference of functional dependencies from relations," *Theor. Comput. Sci.*, vol. 149, no. 1, pp. 129–149, 1995.
[24] R. King and J. Oil, "Discovery of functional and approximate functional dependencies in relational databases," *J. Appl. Math. Decis. Sci.*, vol. 7, no. 1, pp. 49–59, 2003.
[25] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga, "CORDS: Automatic discovery of correlations and soft functional dependencies," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 647–658.
[26] S. Kruse and F. Naumann, "Efficient discovery of approximate dependencies," *Proc. VLDB Endowment*, vol. 11, no. 7, pp. 759–772, 2018.
[27] L. Caruccio, V. Deufemia, and G. Polese, "Mining relaxed functional dependencies from data," *Data Mining Knowl. Discov.*, vol. 34, no. 2, pp. 443–477, 2020.
[28] L. Caruccio, V. Deufemia, and G. Polese, "Evolutionary mining of relaxed dependencies from big data collections," in *Proc. 7th Int. Conf. Web Intell. Mining Semantics*, 2017, pp. 1–10.
[29] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 5, pp. 683–698, May 2011.
[30] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, 2007, pp. 746–755.
[31] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 376–390, 2008.
[32] S.-L. Wang, J.-W. Shen, and T.-P. Hong, "Incremental discovery of functional dependencies using partitions," in *Proc. Joint 9th IFSA World Congr.*, 2001, pp. 1322–1326.
[33] P. Schirmer et al., "DynFD: Functional dependency discovery in dynamic datasets," in *Proc. Int. Conf. Extending Database Technol.*, 2019, pp. 253–264.
[34] K. Belhajjame, "DynASt: Efficient maintenance of agree-sets against dynamic datasets," in *Proc. 26th Int. Conf. Extending Database Technol.*, 2023, pp. 14–26.
[35] S. M. Fakhrahmad, M. Sadreddini, and M. Z. Jahromi, "AD-Miner: A new incremental method for discovery of minimal approximate dependencies using logical operations," *Intell. Data Anal.*, vol. 12, no. 6, pp. 607–619, 2008.
[36] L. Caruccio and S. Cirillo, "Incremental discovery of imprecise functional dependencies," *J. Data Inf. Qual.*, vol. 12, no. 4, pp. 1–25, 2020.
[37] T. Papenbrock et al., "Functional dependency discovery: An experimental evaluation of seven algorithms," *Proc. VLDB Endowment*, vol. 8, no. 10, pp. 1082–1093, 2015.

Bernardo Breve received the MSc (cum laude) and PhD degrees in computer science from the University of Salerno, in 2019 and 2023, respectively. He is currently a post-doc research fellow with the Department of Computer Science, University of Salerno. He regularly serves as a reviewer for many international journals, a member of international conference committees, and has served as a lead guest editor for a special issue in the *Multimedia Tools and Applications* journal. His research interests include artificial intelligence, data imputation, and natural language processing.

Loredana Caruccio received the BSc (cum laude), MSc (cum laude), and PhD degrees in computer science from the University of Salerno, in 2009, 2012, and 2018, respectively. She has been a visiting researcher with the Hasso Plattner Institute, and an adjunct professor with the Université C. Bernard Lyon 1, France. She is currently a tenure track assistant professor with the University of Salerno. Her research interests include data profiling, artificial intelligence, and data privacy. She is a reviewer for international journals, member of international conference committees, and an Associate Editor for international journals.

Stefano Cirillo received the PhD degree in computer science, in March 2022 with the maximum degree. He is an assistant professor with the Department of Computer Science, University of Salerno. He is a member of the IEEE Computer Society and ACM. His current research interests include data profiling algorithms, particularly on the discovery of relaxed functional dependencies, data privacy, artificial intelligence, and social network analysis. He is a member of the program committee of several international conferences and is an editorial board member and associate editor of several prestigious journals.

Vincenzo Deufemia (Member, IEEE) received the laurea (cum laude) and PhD degrees in computer science from the University of Salerno, in 1999 and 2003, respectively. He has been a visiting researcher with the University of Western Sydney, Australia, in 2013. He is currently an associate professor with the University of Salerno. He serves as a reviewer for several international journals and has been a member of international conference committees. He is a member of various associations, including IEEE and ACM. Recently, he has focused on topics related to natural language processing, data profiling, and usable security in IoT.

Giuseppe Polese (Member, IEEE) received the laurea (cum laude) degree in computer science from the University of Salerno, the MSc degree in computer science from the University of Pittsburgh, USA, and the PhD degree in applied mathematics and computer science from the University of Naples. He is a full professor with the University of Salerno. He has been visiting researcher with the University of Svizzera Italiana and the University of Pittsburgh. His research interests focus on data science and machine learning. He is a member of the ACM. Moreover, he is a member of the Editorial Board of several international journals.