Relaxed Functional Dependency Discovery in Incremental Scenarios

Bernardo Breve[1], Loredana Caruccio[1], Stefano Cirillo[1], Vincenzo Deufemia[1] and Giuseppe Polese[1]

[1]University of Salerno, via Giovanni Paolo II, n.132, 84084 Fisciano (SA), Italy

Abstract

The extraction of metadata from dynamic data sources represents an extremely challenging task of the data profiling research area, since it requires to handle the update of the inferred metadata without processing the whole dataset from scratch upon modifications. This discussion paper presents IndiBits, an approach for discovering relaxed functional dependencies (RFDs for short), which represent data relationships relying on approximate matching paradigms. It exploits a binary representation of data similarities, a new validation method, and specific search methods, to dynamically update the set of RFDs, based on previously holding RFDs and the type of modifications performed over data. Experimental results demonstrate the effectiveness of IndiBits on real-world datasets, even in comparison with FD and RFD discovery algorithms in both static and dynamic scenarios.

Keywords

Data Profiling, Relaxed Functional Dependencies, Incremental Scenarios, Bitwise Similarities

1. Introduction

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

As for FDs, RFDs have been used in several application contexts, especially those in which data are collected from heterogeneous sources [6, 7, 8]. 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 [9, 10]. However, due to the

SEBD 2023: 31st Symposium on Advanced Database System, July 02–05, 2023, Galzignano Terme, Padua, Italy

*Corresponding author.

bbreve@unisa.it (B. Breve); lcaruccio@unisa.it (L. Caruccio); scirillo@unisa.it (S. Cirillo); deufemia@unisa.it (V. Deufemia); gpolese@unisa.it (G. Polese)

0000-0002-3898-7512 (B. Breve); 0000-0002-2418-1606 (L. Caruccio); 0000-0003-0201-2753 (S. Cirillo); 0000-0002-6711-3590 (V. Deufemia); 0000-0002-8496-2658 (G. Polese)

dynamic nature of many real-world datasets, it is necessary to adapt discovery processes in order to guarantee the update of the discovered metadata whenever the dataset changes [11]. Consequently, "incremental" discovery processes are demanded in many existing application domains for RFDs, like data imputation, when the underlying data continuously evolve [12].

Incremental scenarios make the RFD discovery problem more complex and challenging, especially for the representation and management of data and results. In fact, the number of FDs and RFDs holding on a given dataset can be exponential in the number of attributes, requiring the exploration of an extremely large and complex search space [1]. Re-executing the discovery process from scratch upon each change in the dataset is computationally burdensome [11]. Thus, solutions for dynamic scenarios should allow to efficiently (i) (re-)validate previously holding RFDs, (ii) discover new possibly holding RFDs, and (iii) guarantee properties like correctness and minimality for discovered RFDs. In this discussion paper, we present IndiBits, the first incremental discovery algorithm for RFDs relaxing on the attribute comparison (i.e., RFD$_c$s). It optimizes the RFD$_c$ discovery process through a binary representation that maps all the distances between the attribute values in a compact way, facilitating their modifications upon data updates through efficient bitwise operations. Furthermore, IndiBits adapts the refinement property used for validating RFD$_c$s to the dynamic context. Finally, a proper search strategy has been introduced for efficiently browsing the search space according to the specific data updates.

The paper is organized as follows. Section 2 presents the theoretical foundations of considered profiling metadata. Section 3 provides an overview of IndiBits, by also describing details of the validation method underlying it, and the discovery strategy to handle insertion and deletion data modifications. Section 4 reports experimental results to analyze the effectiveness of IndiBits on real-world datasets, also when compared with other FD and RFD$_c$ discovery algorithms. Finally, summary and future directions are included in Section 5.

2. Preliminaries

In this section, we formally introduce preliminary concepts related to RFD$_c$.

RFD$_c$. Given a relational database schema $\mathcal{R}$, and $R = \{A_1, \ldots, A_m\}$ one of its relation schemas, an RFD$_c$ φ on $\mathcal{R}$ is denote by

$$X_{\Phi_1} \rightarrow Y_{\Phi_2} \tag{1}$$

where

- $X, Y \subseteq attr(R)$;
- Φ_1 contains (for each attribute $X_i \in X$) a constraint $\phi_i[X_i]$ that can be used to determine whether pair of tuples with values in $dom(X_i)$ are "similar" enough (likewise for each attribute $Y_j \in Y$ with $\phi_j[Y_j] \in \Phi_2$). More specifically, each $\phi_i[X_i]$ ($\phi_j[Y_j]$ resp.) requires the specification of a similarity/distance function defined on the domain of X_i (Y_j, resp.), an operator, and a threshold setting the boundaries for the satisfaction of the constraint.

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: $X_{\Phi_1} \rightarrow A_{\phi_2}$. Moreover, in the following, we consider a more compact notation for the constraints.

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

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

3. IndiBits

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

IndiBits monitors changes 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 groups of change operations.

An overview of the discovery process underlying IndiBits is shown in Fig. 1. The latter describes how IndiBits iteratively performs the discovery of RFD$_c$s as data is updated over time.

To efficiently handle the representation of the similarity among tuples with respect to the input thresholds Φ, we devised an ad-hoc data structure mapping the satisfiability degree between attribute values, limited by the input thresholds, through a vector of integers, named similarity vector. In particular, similarity vectors are generated for each attribute, according to a theoretical concept called Binary Attribute Satisfiability (BAS) Distance Matrix. For a given attribute B, a BAS Distance Matrix M_B^τ represents a triangular matrix whose row and column indices correspond to the tuples of a relation instance at a given time τ. Each matrix entry will contain either the value 0 or 1 depending on whether the pair of tuples agree with the threshold associated with B. A BAS Distance Vector $M_B^\tau[t_k]$ describes the tuples similar to t_k on attribute B. As an example, in Figure 1, $M_{\mathsf{acuity}}^\tau[t_2]$ corresponds to the bit vector 011, indicating that at time $\tau = 0$, the tuple t_2 is similar, with respect to the attribute acuity, only with tuple t_3 as well as itself. Finally, the similarity vector denoted as S_B^τ, is a vector whose length is to the dataset size. Each element of the vector is characterized by the decimal conversion of the bit vector for a given tuple, where the more significant bit is the far-most right. As an example,

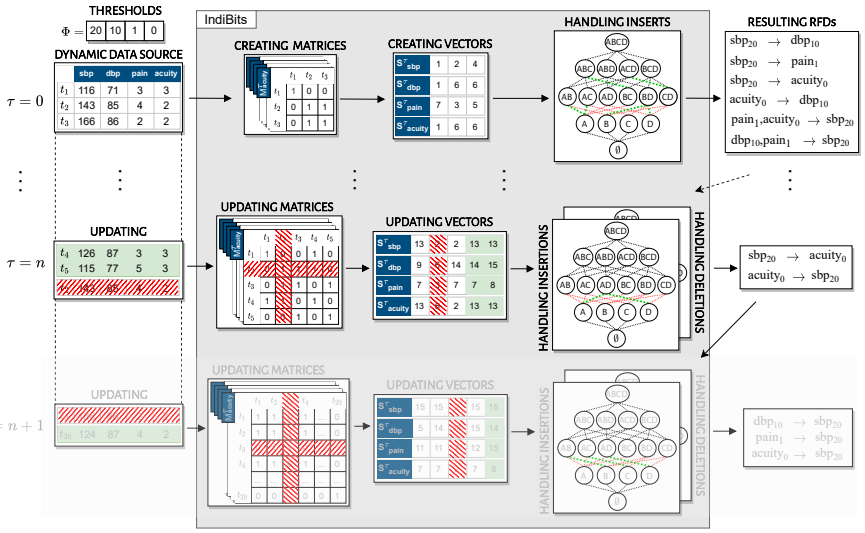

Figure 1: Overview of the process underlying IndiBits.

each element of similarity vector S^{τ}_{acuity} is the decimal conversion of the corresponding BAS Distance Vector produced for a certain tuple at time τ, i.e., $dec(M^{\tau}_{\text{acuity}}[t_1]) = dec(001) = 1$, $dec(M^{\tau}_{\text{acuity}}[t_2]) = dec(011) = 6$, and $dec(M^{\tau}_{\text{acuity}}[t_3]) = dec(011) = 6$. Thus, the similarity vector S^{τ}_{acuity} is 166. Further updates on the dataset over time are transposed to similarity vectors by means of bitwise operations, optimizing their management in an efficient way. A more detailed and formal description of similarity vectors is provided in the full article [13].

According to the similarity representation underlying IndiBits, we can now describe the overall process of IndiBits (see Figure 1). In particular, IndiBits reads the first batch of tuples at time $\tau = 0$ and creates the binary representation of the data according to the thresholds in Φ defined as input. Then, it performs the discovery process by considering only insertion operations and extracts the set of holding RFD$_c$s. For each time $\tau > 0$, IndiBits first processes all the deletions defined in a batch (represented in the image by the tuple crossed out in red), and then the insertion operations (green highlighted tuples), 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 Section 3.2). In particular, for each RFD$_c$ φ holding at time τ as a new candidate RFD$_c$ at time $\tau + 1$, it 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 the operation is a deletion, φ: $X_{\Phi'_1} \rightarrow A_{\phi_2}$ not minimal, then IndiBits 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, then it must generate and validate new candidate RFD$_c$s φ'' that are *specialization* of φ, that is, $\varphi'' : X''_{\Phi''_1} \rightarrow A_{\phi_2}$, with $X \subset X''$ and Φ''_1 is a conjunction of the similarity constraints defined on the attributes X''. Furthermore, IndiBits efficiently validates each candidate RFD$_c$ following the methodology described in Section 3.1.

3.1. RFD$_c$ Validation

Starting from the representation 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 [14].

More formally, given an attribute set X, it is possible to consider $S^\tau_X[t_k]$ computed as $\bigwedge_{B \in X} S^\tau_B[t_k]$, where each $S^\tau_B[t_k]$ is an element of S^τ_B and whose binary representation maps all tuples that are similar to t_k on the values of each attribute in X, according to the similarity constraints defined by Φ. Then, according to the refinement property, if we consider a set $X \cup A \supset X$, it is possible to say that $S^\tau_{X \cup A}$ always refines S^τ_X, since each tuple pair is similar on $X \cup A$ if and only if it is also similar on X. Thus, it is possible to count the number of tuples that are similar to each tuple t_k, by counting the number of bits having the value 1 in $S^\tau_X[k]$, 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. The value 1 is subtracted to exclude comparing the tuple with itself, whereas the division by 2 allows to consider each tuple pair only once.

Example. Let us suppose we have the following similarity vectors: $S^\tau_{\mathsf{pain}} = [65, 350, 350, 350, 350, 928, 95, 672, 830, 928]$, $S^\tau_{\mathsf{chiefcomplaint}} = [1, 130, 100, 280, 280, 612, 612, 130, 280, 608]$, and $S^\tau_{\mathsf{o2sat}} = [1021, 2, 1021, 1021, 1021, 1021, 1021, 1021, 1021, 1021]$ then, the following RFD$_c$ is valid: $\mathsf{pain}_{(\leq 2)}, \mathsf{chiefcomplaint}_{(\leq 8)} \rightarrow \mathsf{o2sat}_{(\leq 1)}$ since it is possible to compute the vectors $S^\tau_X = S^\tau_{\{\mathsf{pain,\ chiefcomplaint}\}} = [1, 2, 68, 280, 280, 544, 68, 128, 280, 544]$ and $S^\tau_{X \cup A} = S^\tau_{\{\mathsf{pain,\ chiefcomplaint,\ o2sat}\}} = [1, 2, 68, 280, 280, 544, 68, 128, 280, 544]$, yielding the same number of similar pairs: $||S^\tau_X|| = \frac{0+0+1+2+2+1+1+0+2+1}{2} = ||S^\tau_{X \cup A}||$.

3.2. Handling Insertions and Deletions

During the discovery process, the insertion operations can or cannot confirm the validity of RFD$_c$s already validated at time τ. This strategy enables IndiBits to avoid re-executing the discovery process from scratch, by keeping track of the previously holding RFD$_c$s. Notice that during the first execution of IndiBits, the most general RFD$_c$s in the search space are considered as starting points. Then, IndiBits performs the validation from the most general to the most specialized RFD$_c$s. More specifically, IndiBits validates each candidate RFD$_c$ and prunes the search space according to the strategy proposed in [2], enabling IndiBits to greatly reduce the number of candidate RFD$_c$s. Instead, if there exists at least one candidate RFD$_c$ that is no longer valid at time $\tau + 1$, 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$_c$, IndiBits checks if there exists at least one valid RFD$_c$ at time $\tau + 1$ that generalizes it. If so, the specialization is a valid and not minimal RFD$_c$, since an RFD$_c$ that generalizes it has already been validated at time $\tau + 1$.

On the other hand, the deletion of one or more tuples always confirms, at time $\tau + 1$, the validity of the previously holding RFD$_c$s, but it could lead to the validation of some RFD$_c$s that were not valid at time τ. In the last case, it is necessary to check the minimality of previously valid RFD$_c$s with respect to those newly validated at time $\tau + 1$. In particular, IndiBits starts by considering the minimal RFD$_c$s holding on a relation instance r at time τ and for each of them considers their direct generalizations as new candidate RFD$_c$s at time $\tau + 1$. If none of these

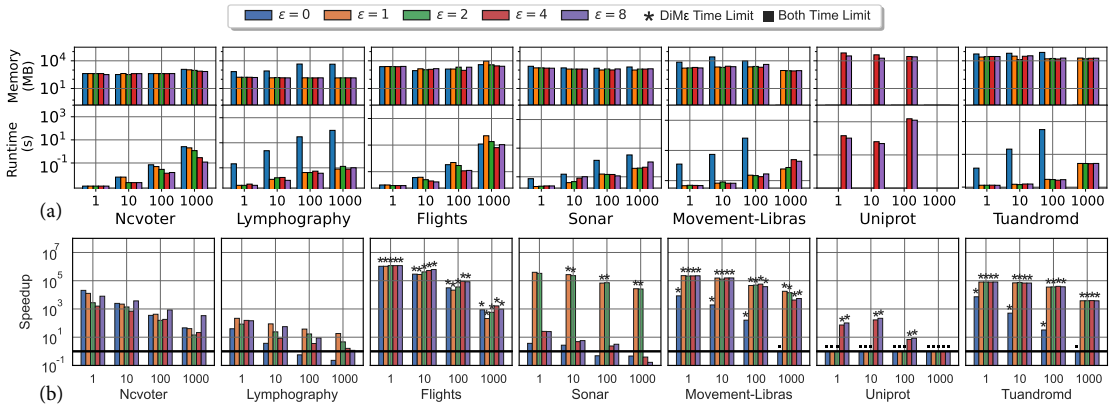

Figure 2: Performances of IndiBits over real-world datasets (a), speedup comparison with DiMε (b).

are valid, IndiBits confirms the validity of the RFD$_c$ from which the generalizations have been produced. On the contrary, for each candidate RFD$_c$ validated at time $\tau + 1$, IndiBits removes all the RFD$_c$s that are specializations of it, and it continues to generalize them as long as possible.

4. Experimental Evaluation

We present experimental results concerning the performances of IndiBits and compare them with those of DiMε [14], an RFD discovery algorithm for static scenarios, and DynFD [15], an FD discovery algorithm for dynamic scenarios.

We implemented IndiBits in Java 17, using the Levenshtein distance for comparing textual attributes, the absolute difference for numerical attributes, and the alphabetic distance for individual characters. We evaluated IndiBits on different real-world datasets whose characteristics are publicly available on the official repository[1]. For each dataset, we used the same threshold ε for all of their attributes, i.e., 0, 1, 2, 4, and 8, and we considered 4 batch size (e.g., the number of changes, in terms of insertion and/or deletion operations, to be considered at each time instant), i.e., $1, 10, 100, 1000$. Finally, we considered a time limit (TL) of 3 hours.

4.1. Performances on real-world datasets

Our first experiment measured the execution times and the memory consumption of IndiBits on the top-6 datasets in terms of attributes (see Fig. 2 (a)). Since these datasets have not been designed for incremental discovery, we simulated an incremental scenario in which tuples are first inserted and then deleted. In particular, for each dataset, we first performed the insertion operations of all tuples, and then we randomly deleted 90% of them.

We report the average runtimes and memory peaks in Fig. 2 (a), by grouping the results according to batch sizes and distance constraints. In particular, IndiBits almost always required less than 10^4 MB of memory, except for *Movement-Libras*, *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 makes the

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

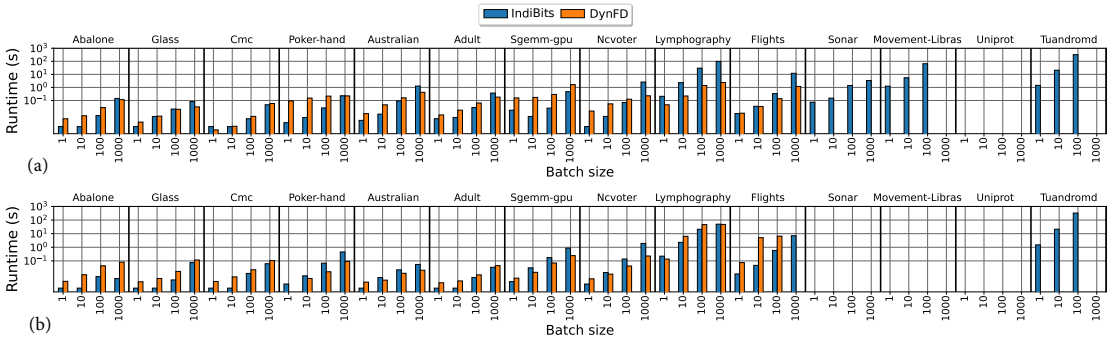

Figure 3: Runtimes evaluation with respect to DynFD on insertions (a), and deletions (b) operations.

memory requirements not severely affected by the dimensionality of datasets and the number of holding RFD_cs.

In general, we can notice that runtimes are quite stable or slightly grow when the batch size increases. Moreover, the average times are over a few seconds, except for particularly borderline configurations, where IndiBits still requires no more than 1 sec. Only on last three datasets IndiBits exceeded the TL in some configurations. Although such datasets represent the three biggest datasets in terms of attributes, for the *Movement-Libras* and *Tuandromd* datasets, IndiBits reached a time limit only with threshold 0 in the last batch size, but the average runtimes for other configurations do not exceed 10^3 sec. Instead, concerning the *Uniprot* dataset, IndiBits completed the discovery process with the two highest attribute comparison thresholds, considering batch sizes equal to 1, 10, and 100.

Summarizing, for the biggest considered datasets (i.e., *Movement-Libras*, *Uniprot*, and *Tuandromd*) IndiBits achieved good time performances with respect to both the number of tuples and attributes. Overall, we cannot identify a strict correlation between the dimensionality of the datasets and both the number of RFD_cs and the IndiBits runtimes.

4.2. Comparative evaluation

We also compared the performances of IndiBits to those of DiMε [14] and DynFD [15].

In this experiment, we will focus on the discovery of RFD_cs by analyzing in which conditions IndiBits under- or out-performs DiMε. To this end, we gradually scale up the size of the dataset according to a defined batch size, each time executing DiMε on an increased dataset. Then, we plot the average runtimes of IndiBits against those of DiMε, by considering the speedup measure. A speed-up of 10 indicates that IndiBits has been 10 times faster than DiMε, 1 indicates that they obtained the same runtime, while a value lower than 1 indicates that DiMε is faster. Fig. 2 (b) shows the results of the comparative evaluation. We can notice that IndiBits is almost always faster than DiMε. Conversely, only a few times DiMε outperforms IndiBits, i.e., on the *Lymphography* dataset with threshold 0 and batch sizes set to 100 and 1000, and on the *Sonar* dataset with threshold 0 and batch sizes set to 100 and 1000, and with thresholds 4 and 8 and batch size 1000. As mentioned above, this can be due to the fact that IndiBits performs worse when there are many invalidations in each batch. Moreover, for the *Sonar* dataset, a static approach like DiMε performs better because its discovery strategy exploits RFD_cs discovered on lowest lattice levels to reduce the execution of validation processes. In

general, we notice that DiMε suffers when executing datasets with a high number of attributes since it reaches the TL in many configurations. Instead, IndiBits reaches the TL only in a few cases of the three datasets with the highest number of columns.

Concerning the comparative evaluation between IndiBits and DynFD, we set up insertion and deletion operations by considering the experimental configuration introduced in Section 4.1, 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. In Fig. 3 we show the execution times achieved considering insertions only (Fig. 3 (a)) and deletion only (Fig. 3 (b)). Notice that, although IndiBits is not optimized for the discovery of FDs, it is able to achieve competitive runtimes with respect to one of the most efficient incremental FD discovery algorithms. In fact, the results show that in many cases IndiBits outperforms or achieves average execution times similar to DynFD. In particular, concerning the insertion operations, we notice that IndiBits typically outperforms DynFD with smaller batches of tuples, i.e., 1, 10, and 100, while for batches with sizes 1000, although it seems that DynFD outperforms IndiBits for *Glass* and *Australian* datasets, the average execution times are of the same order of magnitude. The gap in execution times is greater for *Lymphography*, which represented an extremely challenging dataset for IndiBits when the similarity threshold is set to 0 due to the large amount of FDs with a high number of attributes on the LHS.

Fig. 3 (a) highlights that IndiBits is capable of completing the discovery process without reaching the TL also when DynFD exceeded it, as in the case of *Sonar*, *Movement-Libras*, and *Tuandromd*. On the other hand, Fig. 3 (b) highlights that the task of updating FDs after deletion operations is more challenging for both algorithms, which on average required more time for completing the discovery process, as can be seen by the gap between the algorithms in terms of average execution times being significantly reduced. Finally, both algorithms reached the TL when processing datasets with a high number of columns, except for the *Tuandromd*, dataset for which IndiBits was able to complete the discovery process for 1, 10, and 100 batch sizes.

5. Conclusion

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

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

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

References

[1] F. Naumann, Data profiling revisited, ACM SIGMOD Record 42 (2014) 40–49.

[2] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen, TANE: An efficient algorithm for discovering functional and approximate dependencies, The Computer Journal 42 (1999) 100–111.

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

[4] L. Caruccio, V. Deufemia, G. Polese, Relaxed functional dependencies — A survey of approaches, IEEE Transactions on Knowledge and Data Engineering 28 (2015) 147–165.

[5] S. Song, F. Gao, R. Huang, C. Wang, Data dependencies over big data: A family tree, IEEE Transactions on Knowledge and Data Engineering 34 (2022) 4717–4736.

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

[7] S. Song, A. Zhang, L. Chen, J. Wang, Enriching data imputation with extensive similarity neighbors, Proceedings of the VLDB Endowment 8 (2015) 1286–1297.

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

[9] L. Caruccio, V. Deufemia, G. Polese, On the discovery of relaxed functional dependencies, in: Proceedings of the 20th International Database Engineering & Applications Symposium, IDEAS '16, ACM, 2016, pp. 53–61.

[10] L. Caruccio, V. Deufemia, G. Polese, Evolutionary mining of relaxed dependencies from big data collections, in: Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics, WIMS '17, 2017, pp. 1–10.

[11] L. Caruccio, S. Cirillo, V. Deufemia, G. Polese, Incremental discovery of functional dependencies with a bit-vector algorithm, in: Proceedings of the 27th Italian Symposium on Advanced Database Systems, volume 2400 of *CEUR Workshop Proceedings*, 2019.

[12] M. Khayati, A. Lerner, Z. Tymchenko, P. Cudré-Mauroux, Mind the gap: An experimental evaluation of imputation of missing values techniques in time series, Proceedings of the VLDB Endowment 13 (2020) 768–782.

[13] B. Breve, L. Caruccio, S. Cirillo, V. Deufemia, G. Polese, Indibits: Incremental discovery of relaxed functional dependencies using bitwise similarity, in: 39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, California, USA, April 3–7, 2023., IEEE, 2023.

[14] L. Caruccio, V. Deufemia, G. Polese, Mining relaxed functional dependencies from data, Data Mining and Knowledge Discovery 34 (2020) 443–477.

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