Visualizing Dependencies during Incremental Discovery Processes

Bernardo Breve
Department of Computer Science
University of Salerno
Fisciano (SA), Italy
bbreve@unisa.it

Loredana Caruccio
Department of Computer Science
University of Salerno
Fisciano (SA), Italy
lcaruccio@unisa.it

Stefano Cirillo
Department of Computer Science
University of Salerno
Fisciano (SA), Italy
scirillo@unisa.it

Vincenzo Deufemia
Department of Computer Science
University of Salerno
Fisciano (SA), Italy
deufemia@unisa.it

Giuseppe Polese
Department of Computer Science
University of Salerno
Fisciano (SA), Italy
gpolese@unisa.it

ABSTRACT

Functional dependencies (FDs), and their extensions relaxed functional dependencies (RFDs), represent an important semantic property of data. They have been widely used over the years for several advanced database operations. Thanks to the availability of discovery algorithms for inferring them from data, in the last years (relaxed) FDs have been exploited in many new application contexts, including data cleansing and query relaxation. One of the main problems in this context is the possible "big" number of RFDs that might hold on a given dataset, which might make it difficult for a user getting insights from them. On the other hand, one of the main challenges that has recently arisen is the possibility of monitoring how dependencies change during discovery processes run over data streams, also known as continuous discovery processes. To this end, in this paper we present a tool for visualizing the evolution of discovered RFDs during continuous discovery processes. It permits to analyze detailed results for different types of RFDs, and uses quantitative measures to monitor how discovery results evolve. Finally, in order to facilitate the analysis of results in long discovery processes, the tool enables the comparison among RFDs holding in different time-slots. The effectiveness of the proposed tool has been evaluated in a case study focused on dependencies discovered from streams of data associated to the tweets posted over the Twitter social network.

KEYWORDS

relaxed functional dependencies, visual analytics, continuous discovery, data profiling

1 INTRODUCTION

Metadata have been recognized as a fundamental mean to assess the quality and integrity of data. They range from basic statistics on domain distributions and cardinalities to more complex properties of data, such as data dependencies [18]. The necessity to collect metadata from big data collections is the goal of data profiling approaches. Among the different types of data profiling tasks, in this paper we focus on the discovery of functional dependencies (FDs), and their extensions relaxed functional dependencies (RFDs). They describe relationships among database attributes, which might be exploited in several advanced database operations, such as query optimization, data cleansing, and so forth. In particular, RFDs relax some constraints of canonical FDs by admitting the possibility for a functional dependency to hold on a subset of data (also referred to as RFDs relaxing on the *extent*), and/or by relying on approximate paradigms to compare pairs of tuples (also referred to as RFDs relaxing on the *attribute comparison*) [3].

FDs were traditionally specified at design time, based on the semantics of attributes, and could only change upon schema evolution operations [7]. RFDs can also be specified at design time, but due to the thresholds they require this would be a complex task. Moreover, since they are mainly used in the big data context, it is desirable to have means to automatically discover them from data, so as to be able to capture their evolutions upon changes to the database instances [2][16].

Although the problem of discovering FDs and RFDs from data is extremely complex, the recent definition of efficient algorithms enabled their discovery from "big" data collections. Among these, it is worth to mention [20, 21, 26] for FD discovery, and [4, 6, 15, 25] for RFD discovery. Moreover, recent proposals dealt with incremental or continuous data profiling scenarios [2, 22]. We particularly focus on such kind of scenarios, where the goal is to get holding FDs/RFDs even when the input data dynamically change over time, permitting the discovery of RFDs also from data streams. Nevertheless, in the context of continuous profiling, the possibly huge quantity of holding RFDs at each point in time yields the necessity to graphically visualize them. Indeed, a proper analysis of how RFDs change over time cannot be accomplished by looking at such a huge number of holding RFDs over millions of time instants. To the best of our knowledge, the literature does not offer solutions for mastering such a problem.

In this paper, we present DEVIS (DEpendency VISualizer), a tool for analyzing and comparing RFD discovery results. It permits to analyze the set of RFDs extracted from data streams and their evolution over time. Moreover, DEVIS enables the user to i) visualize the trend related to the number of holding RFDs over time, ii) compare the RFDs holding at different time-slots, and iii) interact with the discovery results by hiding details and/or attributes, and by filtering them according to a specific Right-Hand-Side (RHS) attribute. DEVIS and its visual components can be used over all the available incremental RFD discovery algorithms, also thanks to a parsing module enabling the standardization of discovery results.

The paper is organized as follows. Section 2 describes related visualization approaches and tools. Section 3 presents definitions and theoretical foundations of FDs and RFDs. Section 4 presents DEVIS, whereas Section 5 reports the experimental case study on streams of tweets. Finally, summary and concluding remarks are included in Section 6.

2 RELATED WORK

The availability of efficient RFD (or FD) discovery algorithms yields the necessity to manage big result sets of RFDs, most of which differing only for some values of relaxation parameters. However, only recently such algorithms are becoming capable to scale over big data sources, but there are no many solutions in the literature for handling the complexity related to the visualization of possibly huge numbers of discovered RFDs. Among these, one of the most effective platforms for data profiling is the Metanome project [19], which embeds several algorithms to automatically discover complex metadata, including functional and inclusion dependencies. Moreover, it embeds various result management techniques, such as list-based ranking techniques, and interactive diagrams of discovery results. To this end, a representative scalable platform for analyzing data profiles is Metacrate [14], which permits the storage of different meta-data and their integrations, enabling users to perform several ad-hoc analysis. In this context, the first proposal for visualizing large sets of RFDs is described in [5]. It presents several metaphors for representing RFDs at different levels of detail. Starting from a high-level visualization of attribute correlations, details are interactively revealed, also including details on the relaxation criteria.

A related research context is represented by visual data mining. In the literature, there are many approaches and tools that aim to improve the understanding of data mining algorithms and results (see [10] for a survey). Among these, it is worth to mention Association Rules (ARs) visualization approaches, since the concept of AR is somehow related to that of RFD. Effective examples are the tool in [24], which provides multiple views to visually inspect the overall set of ARs; and the hierarchical matrix-based visualization technique presented in [8].

Although all of these approaches represent effective tools to visualize and explore properties and metadata after the execution of mining/discovery algorithms, our proposal goes beyond the only result visualization problem, since it allows users to explore how RFDs change over time, and to perform result comparisons among different time-slots. This meets the recent need of researchers to focus on more complex scenarios and issues, such as the possibility to explore how data change [1] and to perform continuous profiling [18]. For these reasons, even if in the literature several time-related visualization approaches/tools have been proposed [9, 12, 17, 23], they focus on different application scenarios. Thus, to the best of our knowledge DEVIS is the first proposal that enables the exploration and comparison of RFD discovery results in dynamic scenarios.

3 PRELIMINARES

Before describing the proposed tool we will review the definition of FD, and the general definition of RFD.

FD definition. Let $\mathcal{R}$ be a relational database schema defined over a set of attributes $attr(\mathcal{R})$, and r an instance of $\mathcal{R}$, then we use $\{A, B, C, \dots\}$ to denote single attributes in $attr(\mathcal{R})$, $\{X, Y, W, \dots\}$ sets of attributes in $attr(\mathcal{R})$, $t[A]$ and $t[X]$ the projection of t onto A and X, respectively. An FD over $\mathcal{R}$ is a statement $X \rightarrow Y$ (X implies Y), such that, given an instance r of $\mathcal{R}$, $X \rightarrow Y$ is satisfied in r if and only if for every pair of tuples (t_1, t_2) in r, whenever $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. X and Y represent the Left-Hand-Side (LHS) and Right-Hand-Side (RHS) of the FD, respectively.

Starting from the *canonical* definition of FD over 35 extended definitions have been provided in the literature, which have been generalized under the concept of *Relaxed Functional Dependency* (RFD) [3]. In particular, RFDs enable the consideration of two main relaxation criteria. The first one generalizes the equality constraint used into the FD definition. It requires that the projections of two tuples over a subset of attributes are compared by means of the equality function. Instead, RFDs consider a constraint as a predicate evaluating whether the distance or similarity between two values of an attribute satisfies a given threshold. Thus, it involves a similarity or difference function, such as the Jaro or the Edit distance [11], whose results are verified according to a comparison operator and a threshold.

The second relaxation criterion, defined as relaxation on the *extent*, admits the possibility that the dependency might hold for a subset rather than all the tuples. In particular, the satisfiability degree of an RFD can be formally specified by means of conditions restricting the application domain of the RFD or through a *coverage measure*. The latter measures the minimum number or percentage of tuples on which the RFD must hold. Examples of widely used coverage measures are the confidence and $g3$-error [13].

RFD definition. Consider a relational database schema $\mathcal{R}$, and a relation schema $R = (A_1, \dots, A_m)$ of $\mathcal{R}$. An RFD φ on $\mathcal{R}$ is denoted by

$$\mathbb{D}_c : X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2} \tag{1}$$

where

- $\mathbb{D}_c = \left\{ t \in dom(R) \mid \bigwedge_{i=1}^{m} c_i(t[A_i]) \right\}$ with $c = (c_1, \dots, c_m)$, and each c_i is a predicate on $dom(A_i)$ filtering the tuples on which φ applies;
- $X = B_1, \dots, B_h$ and $Y = C_1, \dots, C_k$, with $X, Y \subseteq attr(R)$ and $X \cap Y = \emptyset$;
- $\Phi_1 = \bigwedge_{B_i \in X} \phi_i[B_i]$ ($\Phi_2 = \bigwedge_{C_j \in Y} \phi_j[C_j]$, resp.), where ϕ_i (ϕ_j, resp.) is a conjunction of predicates on C_i (C_j, resp.) with $i = 1, \dots, h$ ($j = 1, \dots, k$, resp.). For any pair of tuples (t_1, t_2)$\in dom(R)$, the constraint Φ_1 (Φ_2, resp.) is true if $t_1[B_i]$ and $t_2[B_i]$ ($t_1[C_j]$ and $t_2[C_j]$, resp.) satisfy the constraint ϕ_i (ϕ_j, resp.) $\forall\, i \in [1, h]$ ($j \in [1, k]$, resp.).
- Ψ is a coverage measure defined on $dom(R)$, quantifying the amount of tuples violating or satisfying φ. It can be defined as a function $\Psi : dom(X) \times dom(Y) \rightarrow \mathbb{R}^+$, where $dom(X)$ is the cartesian product of the domains of attributes composing X.
- ε is a threshold indicating the upper bound (or lower bound in case the comparison operator is $\geq$) for the result of the coverage measure.

Given $r \subseteq \mathbb{D}_c$ a relation instance on R, r *satisfies* the RFD φ, denoted by $r \models \varphi$, if and only if: $\forall\, t_1, t_2 \in r$, if Φ_1 indicates true, then *almost always* Φ_2 indicates true. Here, *almost always* is expressed by the constraint $\Psi \leq \varepsilon$. A more general definition covering more types of RFDs has been provided in [3].

In the following we consider FDs, RFDs relaxing on the tuple comparison only, RFDs relaxing only on the extent through a coverage measure, and a hybrid version of them, with the general

acronym RFD. In fact, they can be described according to the equation (1). In this study, we do not consider RFDs relaxing on the extent through a condition of domain values.

For RFDs relaxing on the tuple comparison only, when no couple of tuples yields an RFD violation, the expression $\Psi(X, Y) = 0$ is omitted from the RFD expression. Moreover, for sake of simplicity in what follows we always consider the $g3$-error as coverage measure, the operator $\leq$ for tuple comparison, and several distance functions, such as the edit distance for textual values, the absolute difference for numerical values, and so forth. Finally, without loss of generality, we can consider RFDs with a single value on the RHS.

As an example, in a database of tweets, it is likely to have the same number of followed accounts (#Friends) for accounts with the same name and location. Thus, an FD {Name, Loc}⟶ #Friends might hold. However, this property might also hold for names and locations stored using different abbreviations and/or similar numbers of friends, hence the following RFD might hold:

$$\{\mathsf{Name}_{\leq 2}, \mathsf{Loc}_{\leq 4}\} \longrightarrow \#\mathsf{Friends}_{\leq 10}$$

Moreover, since accounts might change location during their account's life, or there might be changes in the number of friends, the previous RFD should tolerate possible exceptions. This can be modeled by introducing a different coverage measure into the RFD:

$$\{\mathsf{Name}_{\leq 2}, \mathsf{Loc}_{\leq 4}\} \xrightarrow{\psi(Name,Loc,\#Friends)\leq 0.03} \#\mathsf{Friends}_{\leq 10}$$

Among all RFDs holding on a given relation database schema $\mathcal{R}$, only a subset of them can be considered as the most meaningful ones. In fact, in the context of FDs, Armstrong's inference rules have been theoretically defined in order to derive the minimal set of FDs. The latter represents the set of FDs from which all valid ones can be derived. Into the context of RFDs, an RFD $X_{\Phi_1} \xrightarrow{\psi(X,A)\leq\varepsilon} A_{\phi_2}$ is said to be *minimal* if and only if 1) it is non-trivial, i.e. no attributes are shared between the LHS and RHS; 2) it has the minimum possible number of LHS attributes; 3) it has LHS attributes with maximum possible threshold values; and 4) it has the RHS attribute with minimum possible threshold value.

Algorithms for discovering RFDs typically return the set of *minimal* RFDs satisfied by the database instance provided in the input. However, as said before, the number of holding RFDs can be huge, especially when relaxation criteria settings increase. This prevents a proper analysis of RFDs, and mostly their evolution over time. Aiming to solve this problem, we propose DEVIS, which is described in the next section.

4 A TOOL FOR ANALYZING AND COMPARING RFD DISCOVERY RESULTS

In this section, we describe DEVIS. It enables monitoring of RFD discovery results over time. In particular, we first present the system architecture (Section 4.1), and then provide details on i) the visual interface (Section 4.2), ii) the feature enabling the comparison between two different time-slots (Section 4.3), and iii) the interactions that a user can perform (Section 4.4).

4.1 System architecture

Visualizing RFD discovery results during the execution of incremental algorithms is a quite complex problem. There are several issues related to discovery processes that lead to specific choices for the system architecture: 1) the amount of RFDs processed at each point in time can be significantly big, 2) the presence of several visualization components could require frequent updates in short time, and 3) discovery algorithms rely on different implementation technologies. To this end, DEVIS has been designed aiming to enable users monitor results during the execution of RFD discovery algorithms through a responsive visual interface. Moreover, it is based on a client-server architecture, whereas each of its modules is standalone and shares information with other modules by using the JSON standard. High component modularity and maintainability allow for the substitution of any back-end component, provided that its output is formatted according to the JSON standard defined for the interaction.

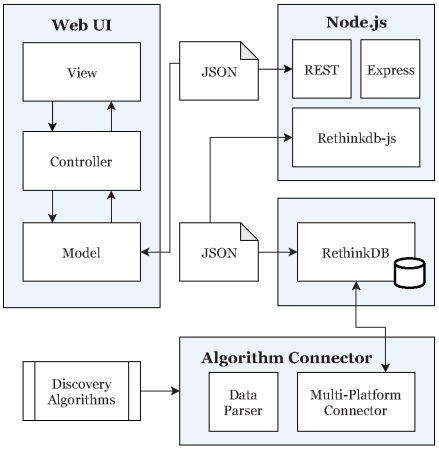


Figure 1: The system architecture.

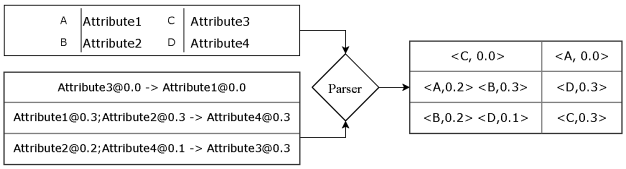


Figure 2: Examples of RFDs processed by the parser module.

Figure 1 shows the architecture of DEVIS. In particular, the client is a web application based on the model-view-controller (MVC) architecture that communicates with the back-end modules through live queries. The back-end server is a long-lived Node.js application running distributed jobs while keeping a live connection with the database. Back-end modules receive data from live queries, process and represent them on the DEVIS's dashboard.

Although the architecture is flexible, to make DEVIS compatible with most FD and RFD discovery algorithms, it is necessary to use a parsing module that uniforms the syntax of the dependencies regardless of possible thresholds. Figure 2 shows an example of a parsing operation. In particular, it shows three RFDs on its left-bottom part. The first one is an FD, and the other two are RFDs relaxing on the attribute comparison, but with different thresholds. The module receives the dependencies, manipulates their syntax, and extracts a compact version so as to store it in the real-time database RethinkDB[1]. The latter guarantees high scalability, REST API, and real-time monitoring for the storage operations. It allowed us to build multi-language connection

[1]https://rethinkdb.com

modules to integrate different discovery algorithms into DEVIS. As said before, DEVIS focuses on continuous data profiling algorithms [18] and therefore it requires to maximize fluidity and to minimize processing times within the visual interface. Thus, all selected technologies for both client- and server-side support real-time updating of data.

4.2 RFDs visualization

Systems for data relationship visualization vary in what they display and how users interact with them. However, most of these data visualization systems are not intended for the dynamic elaboration of data, restricting themselves to a static representation of large sets of data. In the case of RFD discovery algorithms, a static representation only allows for the visualization of results. However, in the context of continuous data profiling it provides much more useful insights studying how dependencies evolve over time.

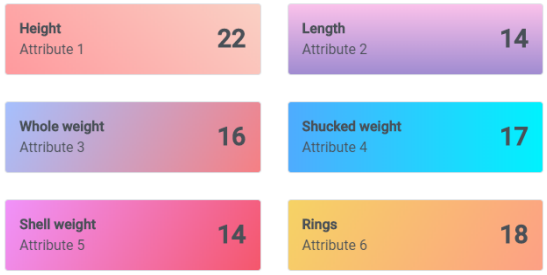
The dynamic representation of a large portion of data requires the application of interactive graphs, capable of highlighting the arrival of new information without losing track of pre-existent information. Hence, a dynamic visual representation has been defined and implemented through a line plot, showing the variation on the number of dependencies being discovered or invalidated. Moreover, a dependency table has been used to highlight further details concerning discovered dependencies.

Figure 4 shows the line plot, which is responsible to display information about the number of valid dependencies discovered over time. The line plot block is divided into two sections, the upper one is the main line plot, showing the number of valid RFDs on its y-axis and the time-stamp on its x-axis; the line plot at the bottom of the figure is a replica of the upper one, but it enables the interaction with the user, allowing him/her to select an interval of time through a brush in order to visualize details on how the number of discovered RFDs changed during the selected period. Both these line plots are updated, so that users can see how the trend changes at any time instant, as the RFD discovery process progresses.

The dependency table displayed below the line plot is structured so that each row represents an RFD, whereas each column represents an attribute of the dataset. More specifically, the first column labeled "RHS" informs the user about the RHS attribute of the dependency. Instead, the other columns display details on all the attributes that can appear in a dependency.

We provided DEVIS with statistical counters displaying the number of RFDs that are valid, invalid and minimal, on a given time instant (Figure 3(a)). Furthermore, at the bottom of the interface, the number of minimal dependencies are grouped by each RHS attribute appearing in the discovered RFDs (Figure 3(b)).

(a) Dependency validation counters.



(b) Minimal dependencies on single RHS attributes.

Figure 3: Statistical counter included into DEVIS.

4.3 Comparing dependencies over time

An important feature of DEVIS is the possibility to monitor the variation of the dependency status in two different time intervals. This process has no impact on the discovery algorithm, which continues to provide new status variations while the user is focused on the analysis.

A comparison between discovery results at different time intervals by processing a dedicated button placed in the side menu, which duplicates both the line-plot and the dependency table. Both the line plots and the dependency tables are continuously updated, but they react differently to the interaction of the user.

Figure 5 shows the disposition of the line plots. By interacting with the bottom line-plots, the user can select the time intervals to be analyzed, and visually compare the dependencies through the two associated dependency tables.

4.4 Interaction in depth

As mentioned above, users interact with the interface through the bottom line plot(s) by selecting time intervals. To this end, the user places the mouse pointer in correspondence with the starting point of the interval and drags it by holding the left mouse button up to the end point of the time interval. The selection is highlighted with a grey rectangle on the bottom line plot. During the process the top line plot reacts consequently, reducing the scale on the x-axis in order to adapt the range boundaries specified by the user. Nevertheless, the discovery process is still in progress, and the user can verify this through the UI when s/he stops the brushing process by pressing the left mouse button at any point of the bottom line plot.

Concerning the dependency table, Figure 6 highlights all the interaction functionalities offered by DEVIS. In order to allow the user to reduce the table content, we added two different types of filters: a search text field (Figure 6, yellow rectangle) allowing for a global search over the attributes in the table, and a column-based filter (Figure 6, green rectangle). Based on the text the user inserts in the search text field, the table content is adapted showing only the rows containing that specific value. If the user writes a value in the column-based filter, the table content shows only the rows having that value for the attribute corresponding to the specific column. Instead, if more column filters get compiled, only the rows having those values for all corresponding columns are shown. We planned a different behavior in the case the user decides to filter out the dependency on the table typing a value for the column "RHS". Indeed, for this particular column only, as the user provides information about an attribute name, the query also affects the top line plot, presenting a new blue line showing information about how the number of dependencies having that attribute on their RHS has changed (Figure 7).

The dependencies are sorted by default in a way that the most recently altered ones get added on top of the table. However, users can also define column-based sorting criteria by clicking on the

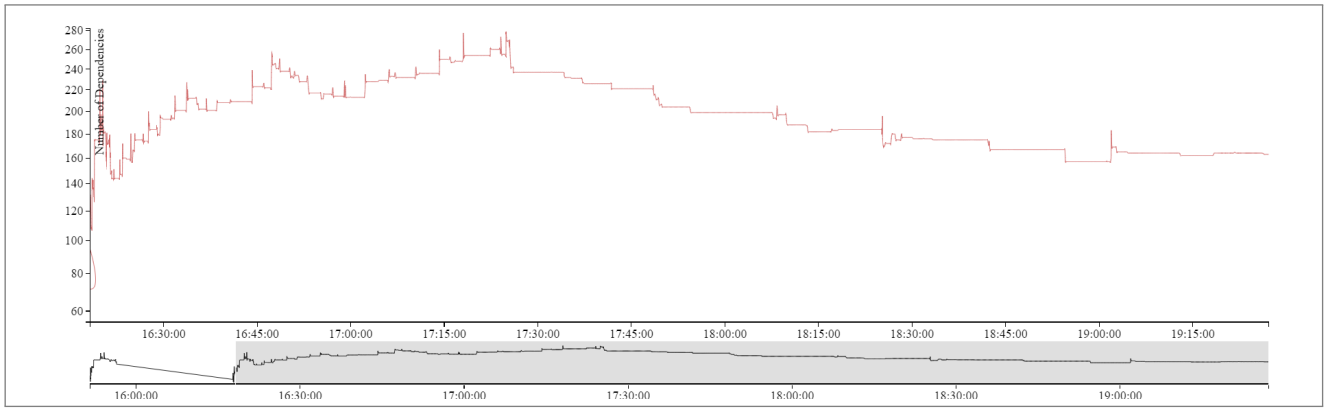


Figure 4: Line plot visualization.

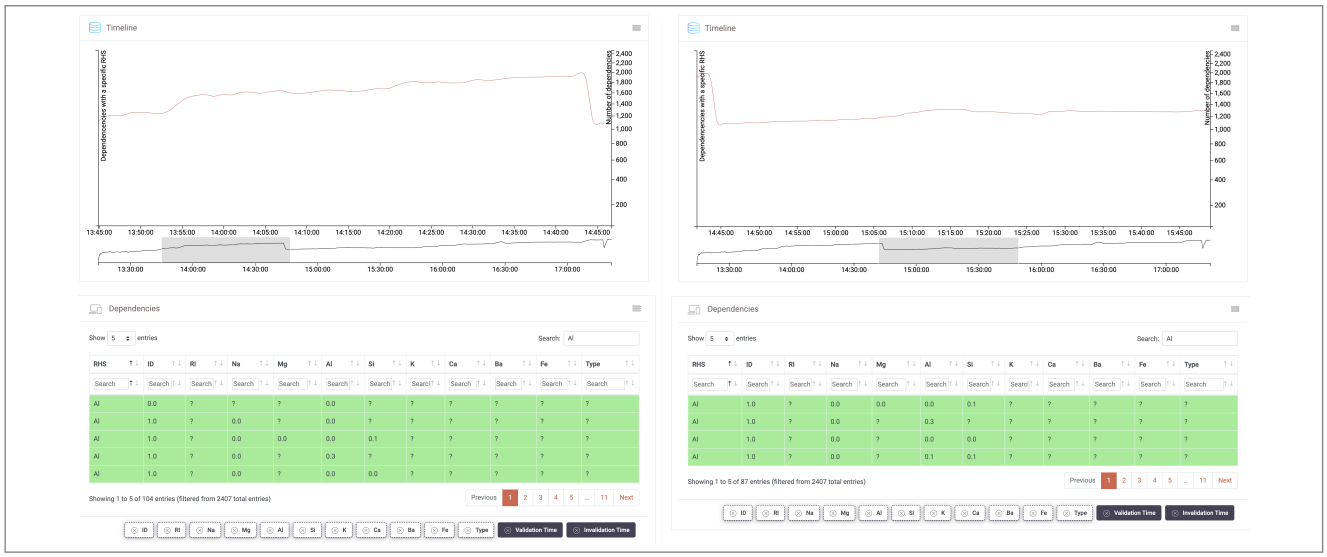


Figure 5: Comparing dependencies between two different time-slots.

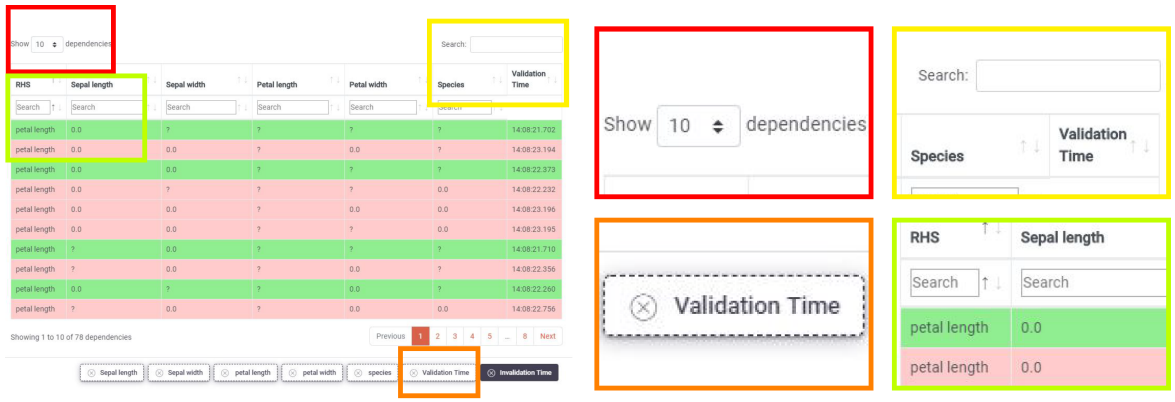


Figure 6: Interacting with the dependency table.

chosen column and selecting either an ascending or descending order.

Due to the conspicuous amount of dependencies that might be discovered, we also added a paging functionality (Figure 6, red rectangle), allowing the user to decide how many rows should be displayed in a single table page, preventing DEVIS's UI to be overfilled with too many elements.

Finally, the buttons below the table allow the user to decide which column to hide or show in the table (Figure 6, orange rectangle). Pressing on the "x" symbol hides the column and

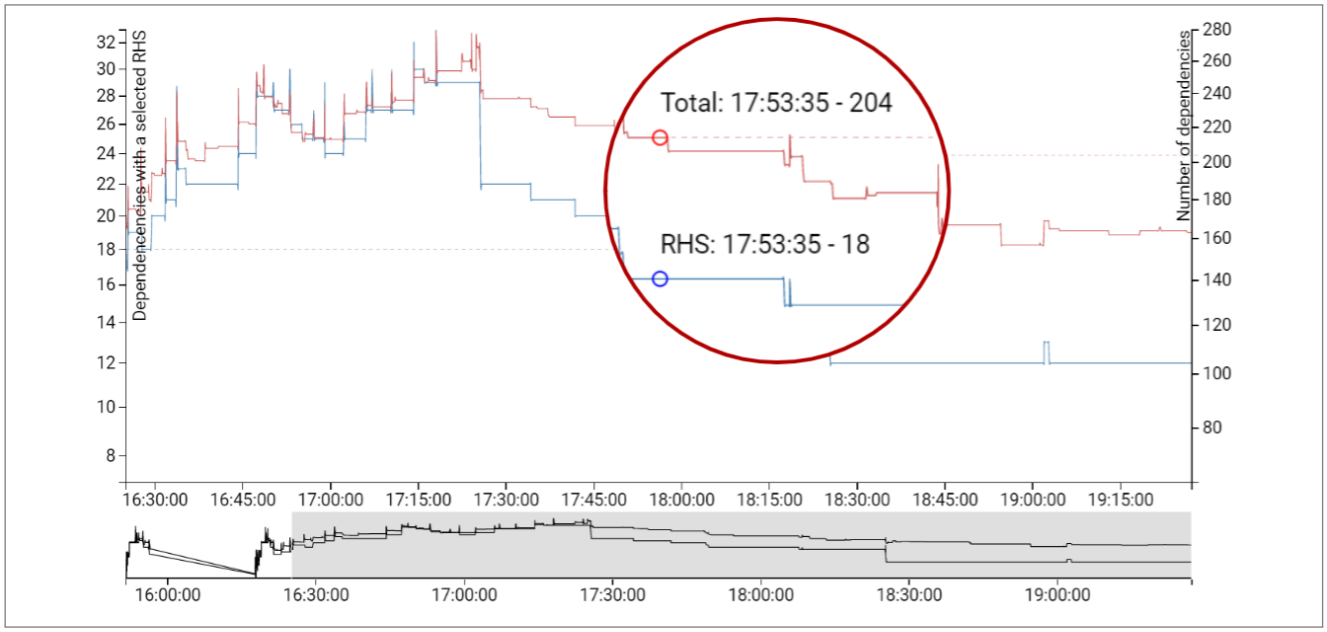


Figure 7: Interacting with the line plot.

ID	#Tweet	Initial Time	End Time	#Validations	#Analyzed FDs	#Minimal FDs
A	1347	11:42	16:51	967077	3731	150
B	1450	19:00	23:59	876810	3600	175
C	1053	11:40	17:05	604652	3503	172
D	1150	19:38	23:47	472650	2885	159

Table 1: Statistics on the different evaluation sessions.

∩	**A** 150	**B** 175	**A∩B** 90
C 172	**A∩C** 87	**B∩C** 114	**A∩B∩C** 76
D 159	**A∩D** 117	**B∩D** 85	**A∩B∩D** 81
C∩D 79	**A∩C∩D** 73	**B∩C∩D** 70	**A∩B∩C∩D** 68

Table 2: Statistics about the minimal FDs obtained from the tweet streams of the two experimental sessions.

makes the corresponding rectangle turn to black. Another click on the "x" symbol reverses this process.

5 ANALYZING FDS FROM TWITTER STREAMS: A CASE STUDY

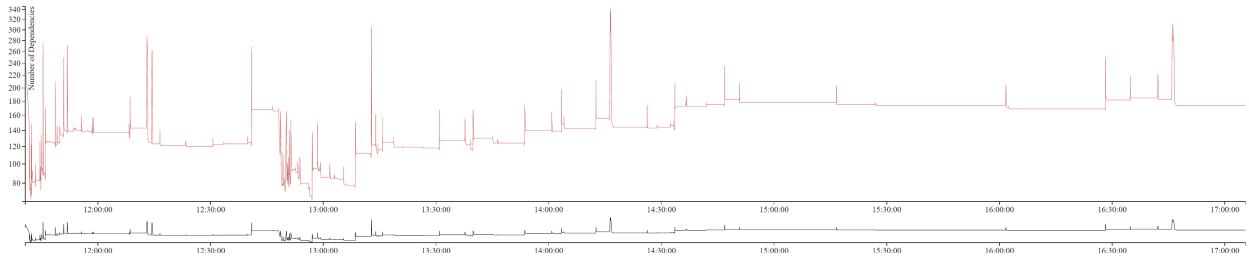
In order to verify the effectiveness of DEVIS on a real-world scenario we analyzed discovery results on the data stream of Twitter. We selected this scenario for its stressful data load, and the possibility to monitor FD and RFD discovery algorithms for long time. In particular, we monitored the results of the incremental FD discovery algorithm described in [2], extended to discover FDs in data streams.

More specifically, we selected 11 attributes concerning tweet's account details from the items of Twitter, avoiding all the key attributes, images and so on, (i.e., Name, Email, Description, #Followers, #FriendsCount, CreatedAt, #Favourites, TimeZone, Lang, #Statuses, #Listed), and filtered tweets by considering only those containing specific keywords ("2020", "newyear", "leapyear", "happy"). Tests have been performed by considering 4 execution sessions of the algorithm, totally lasting about 4 hours, during which DEVIS continuously updated the set of discovered dependencies as new tweets were read.

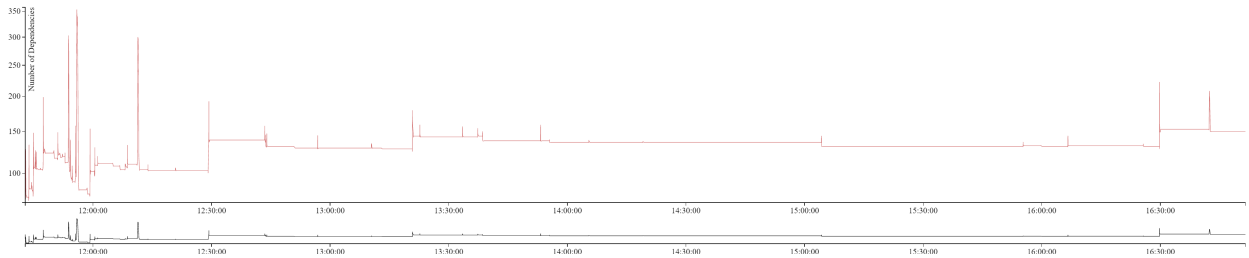
Table 1 shows some statistics concerning the performed evaluation, such as the initial and the end-time of each session, the number of the tweets caught in the given period, and some details on the number of validations performed by the FD discovery algorithm and the number of distinct FDs involved in the total session time. Last column reports the number of minimal FDs discovered at the end of the given session. We can notice that at the end the number of minimal FDs are quite similar, even if the number of performed validations is not necessarily close. This is due to the fact that we had no influence on tweets caught on the stream, and therefore each session may yield different validation/invalidation processes.

To get some knowledge on the FD validation trend, DEVIS enabled us to analyze line plots. As an example, Figure 8 shows the line plots of two executions, both performed in the morning. As expected, the two plots present a high variation at the beginning, and then tend to be stable. However, by comparing the two plots we can notice that the two trends present several differences, especially for the first hour of execution.

As said above, the final number of minimal FDs discovered in the different sessions is quite similar. Thus, DEVIS allowed us to compare the holding and minimal FDs at the end of the single execution. Quantitative comparison results between all considered sessions are reported in Table 2, which show that the different sessions shared many FDs. This result is not obvious, since in most cases the tweets of the sessions are created by different accounts.



(a) Session A.



(b) Session B.

Figure 8: Line plots of two execution sessions.

Some examples of minimal FDs shared between the two considered sessions are shown in Table 3. The first FD is quite intuitive, whereas the other ones require some explanation. As an example, the second FD states that given two tweets, if they are created by the same account (Name) and the number of followers is the same (#Followers), then the number of the account's friends does not change (#Friends). This might be due to the fact that in the considered period of time the users focused on writing tweets (this is the baseline of our scenario) without following new accounts. The breadth of the time interval also affected the other three FDs. In particular, by comparing the last two FDs we can deduce that the number of likes (#Favourites) changes more frequently wrt. the number of lists to which the account owner is registered (#Listed). In fact, #Favourites required another attribute #Lang on the LHS to be determined.

From these experiments of Twitter data DEVIS allowed us to figure out the monitored characteristics of different accounts obey to similar FDs. This was an interesting insight, because such findings might be obvious for fake accounts, but not for the genuine accounts we used in the experiments.

6 CONCLUSION AND VISION

Continuous profiling permits to consider extremely complex scenarios, where data are dynamically produced. In particular, each modification to data might produce the evolution of many FDs and/or RFDs. In some cases, due to both the quantity of holding FDs and RFDs and the high-frequency of updates, it is unthinkable to analyze the minimal FDs and RFDs for each time instant. Thus, to facilitate the interpretation of discovered dependencies over time, we have proposed the tool DEVIS, which relies on several visual components to let users actively analyze and explore the evolution of holding RFDs discovered through incremental FD or RFD discovery algorithms.

In the future, we would like to investigate the analysis tasks involved during the incremental discovery of RFDs by conducting interviews to potential users. The latter can belong to many different categories, depending on the application context in which RFDs are exploited. So far RFDs have been mainly destinated at data engineers and scientists. The aim of such interviews will also be to identify new filtering criteria for comparing RFDs, and to embed them within DEVIS. Finally, based on the results of users interviews, we would like to design new visual metaphors to graphically represent the general evolution of holding RFDs, so as to better support the identified analysis tasks.

Examples of FDs
TimeZone → Lang
Name, #Statutes → #Friends
Descr, Lang, #Statuses → CreatedAt
Descr, #Followers, CreatedAt → #Listed
Descr, #Followers, Lang, CreatedAt → #Favourites

Table 3: Some meaningful FDs shared into the sessions.

REFERENCES

[1] Tobias Bleifuß, Leon Bornemann, Theodore Johnson, Dmitri V Kalashnikov, Felix Naumann, and Divesh Srivastava. 2018. Exploring change: a new dimension of data analytics. *Proceedings of the VLDB Endowment* 12, 2 (2018), 85–98.

[2] Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. 2019. Incremental Discovery of Functional Dependencies with a Bit-vector Algorithm. In *Proceedings of the 27th Italian Symposium on Advanced Database Systems*.

[3] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2016. Relaxed Functional Dependencies – A Survey of Approaches. *IEEE Transactions on Knowledge and Data Engineering* 28, 1 (2016), 147–165.

[4] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2019. Mining relaxed functional dependencies from data. *Data Mining and Knowledge Discovery* (2019), To appear.

[5] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2019. Visualization of (multimedia) dependencies from big data. *Multimedia Tools and Applications* 78, 23 (2019), 33151–33167.

[6] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2020. Discovering Relaxed Functional Dependencies based on Multi-attribute Dominance. *IEEE Transactions on Knowledge and Data Engineering* (2020), To appear.

[7] Loredana Caruccio, Giuseppe Polese, and Genoveffa Tortora. 2016. Synchronization of queries and views upon schema evolutions: A survey. *ACM Transactions on Database Systems (TODS)* 41, 2 (2016), 1–41.

[8] Wei Chen, Cong Xie, Pingping Shang, and Qunsheng Peng. 2017. Visual analysis of user-driven association rule mining. *Journal of Visual Languages & Computing* 42 (2017), 76–85.

[9] Luca Corcella, Marco Manca, Fabio Paternò, and Carmen Santoro. 2018. A Visual Tool for Analysing IoT Trigger/Action Programming. In *International Conference on Human-Centred Software Engineering*. Springer, 189–206.

[10] MC Ferreira De Oliveira and Haim Levkowitz. 2003. From visual data exploration to visual data mining: a survey. *IEEE Transactions on Visualization and Computer Graphics* 9, 3 (2003), 378–394.

[11] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering* 19, 1 (2007), 1–16.

[12] Daniel A Keim, Jörn Schneidewind, and Mike Sips. 2004. CircleView: a new approach for visualizing time-related multidimensional data sets. In *Proceedings of the working conference on Advanced visual interfaces*. ACM, 179–182.

[13] Jyrki Kivinen and Heikki Mannila. 1995. Approximate inference of functional dependencies from relations. *Theoretical Computer Science* 149, 1 (1995), 129–149.

[14] Sebastian Kruse, David Hahn, Marius Walter, and Felix Naumann. 2017. Metacrate: Organize and analyze millions of data profiles. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 2483–2486.

[15] Sebastian Kruse and Felix Naumann. 2018. Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment* 11, 7 (2018), 759–772.

[16] Chien-Min Lin, Yu-Lung Hsieh, Kuo-Cheng Yin, Ming-Chuan Hung, and Don-Lin Yang. 2013. ADMiner: An Incremental Data Mining Approach Using a Compressed FP-tree. *JSW* 8, 8 (2013), 2095–2103.

[17] Adam Marcus, Michael S Bernstein, Osama Badar, David R Karger, Samuel Madden, and Robert C Miller. 2012. Processing and visualizing the data in tweets. *ACM SIGMOD Record* 40, 4 (2012), 21–27.

[18] Felix Naumann. 2014. Data profiling revisited. *ACM SIGMOD Record* 42, 4 (2014), 40–49.

[19] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data profiling with Metanome. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1860–1863.

[20] Thorsten Papenbrock and Felix Naumann. 2016. A hybrid approach to functional dependency discovery. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 821–833.

[21] Hemant Saxena, Lukasz Golab, and Ihab F Ilyas. 2019. Distributed Discovery of Functional Dependencies. In *IEEE 35th International Conference on Data Engineering (ICDE '19)*. IEEE, 1590–1593.

[22] Philipp Schirmer, Thorsten Papenbrock, Sebastian Kruse, Dennis Hempfing, Torben Meyer, Daniel Neuschäfer-Rube, and Felix Naumann. 2019. DynFD: Functional Dependency Discovery in Dynamic Datasets. In *Proceedings of the 22nd International Conference on Extending Database Technology (EDBT '19)*. 253–264.

[23] Vinícius Segura and Simone DJ Barbosa. 2017. Historyviewer: Instrumenting a visual analytics application to support revisiting a session of interactive data analysis. *Proceedings of the ACM on Human-Computer Interaction* 1, EICS (2017), 11.

[24] Yoones A. Sekhavat and Orland Hoeber. 2013. Visualizing Association Rules Using Linked Matrix, Graph, and Detail Views. *International Journal of Intelligence Science* 3 (2013), 34–49.

[25] Shaoxu Song and Lei Chen. 2013. Efficient discovery of similarity constraints for matching dependencies. *Data & Knowledge Engineering* 87 (2013), 146–166.

[26] Ziheng Wei and Sebastian Link. 2019. Discovery and ranking of functional dependencies. In *IEEE 35th International Conference on Data Engineering (ICDE '19)*. IEEE, 1526–1537.