Journal of Visual Language and Computing

journal homepage: www.ksiresearch.org/jvlc

Monitoring Evolution of Dependency Discovery Results

Loredana Caruccio, Stefano Cirillo

University of Salerno, via Giovanni Paolo II, 132 84084 Fisciano (SA), Italy

ARTICLE INFO

Article History:
Submitted 15.10.2020
Revised 11.29.2020
Accepted 12.4.2020

Keywords:
Data stream profiling
Big data visualization
Metadata visualization
Continuous discovery
Relaxed functional dependencies

ABSTRACT

The automatic discovery from data of Functional Dependencies (FDs), and their extensions Relaxed Functional Dependencies (RFDs), represents one of the main tasks in the data profiling research area. Several algorithms that deal with the "complex" problem of discovering RFDs have been recognized as a fundamental tool to automatically collect them starting from data. Moreover, the characteristics of scenarios involving "big" data require also profiling tasks to evolve towards continuous ones, which must be capable to dynamically collect and update the set of holding RFDs on the analyzed data. In this context, one of the most critical scenarios is represented by the possibility to discover RFDs over data streams. Nevertheless, although the main goal of discovery algorithms is allowing for fast execution processes, to enable the analysis of the resulting RFDs, it is necessary to also devise methods to continuously monitor discovery results. Thus, one of the main goals is to reduce the users' effort in moving in and out the possible huge quantity of holding RFDs. To this end, in this paper, we present DEVICE, a tool for continuously monitoring resulting RFDs during the execution of discovery processes. In particular, it permits to analyze the evolution of results by using a lattice representation of the search space. Moreover, zooming and filtering functionalities enable the user to focus the analysis on a specific portion of the search space. The effectiveness of the proposed tool has been evaluated in a scenario studying the application of different discovery strategies over a well-known and real-world dataset.

1. Introduction

Collecting metadata from big datasets is the goal of the data profiling research area, in which the discovery of functional dependencies (FDs), and their extensions relaxed functional dependencies (RFDs) represents one of its fundamental tasks. This kind of semantic property describes relationships among database attributes, which might be exploited in several advanced database operations, such as query optimization, data cleaning, and so forth. In particular, RFDs relax some constraints of canonical FDs by admitting the possibility for a 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*) [7].

Although the problem of discovering FDs and RFDs from data is extremely complex, the recent definition of efficient algorithms enabled their discovery from "big" datasets. Among these, it is worth to mention [20, 26, 22] for FD discovery, and [15, 25, 4, 7] for RFD discovery. Moreover, recent proposals dealt with incremental or continuous data profiling scenarios [23, 3, 2]. The latter are particularly useful in current application scenarios, such as big data analytic tasks, in which the possibility to learn predictive models from data requires to dynamically profile data streams and learn models from them. To this end, it is necessary to devise methods and tools capable of visualizing the dynamic evolution of the discovery results characterizing the profile of a data stream, and/or of the predictive models of interest. Indeed, a proper analysis of how RFDs change over time cannot be accomplished by looking at such a huge number of holding RFDs that change very rapidly.

We particularly focus on such kind of scenarios, where the goal is to get holding RFDs even when the input data dynamically change over time, permitting the discovery of RFDs also from data streams. The latter scenario imposes different emerging research challenges, such as the necessity of enabling user i) to continuously monitor and rapidly visualize discovery results, ii) to analyze how a discovery process browses the search space according to data are get-

lcaruccio@unisa.it (L. Caruccio); scirillo@unisa.it (S. Cirillo)
ORCID(s):

DOI reference number: 10.18293/JVLC2020N2-007

ting in, and iii) to easily interact with specific portions of the search space, entailing the focus on a specific portion of results.

To this end, in this paper, we present DEVICE (DEpendencies VIsualizer on lattiCE), which permits to monitor the set of RFDs extracted from data streams through a lattice representation. Moreover, DEVICE enables the user to interact with the discovery results by zooming on the search space and/or filtering results according to specific attributes. Finally, results can be also filtered according to RFD threshold settings. From an architectural point-of-view, DEVICE is scalable towards all possible (incremental) RFD discovery algorithms, also thanks to an input driver connector module devoted to the parsing of input data, so enabling the standardization of discovery results.

The paper is organized as follows. Section 2 describes related visualization approaches and tools. Section 3 presents the theoretical foundations of RFDs and the representation used to model the search space of the discovery problem. Section 4 presents DEVICE, whereas Section 5 reports an experimental case study on the application of different discovery strategies over a real-world dataset. Finally, summary and future directions are included in Section 6.

2. Related Work

Big data visualization is an important research area in which the main goal is to provide effective visualization techniques capable to describe data into their "big" and challenging contexts [21, 13]. In fact, not only the dynamic nature of data increases the complexity of design choices, but also the necessity to provide insights to users in real-time and to enable effective interactions with graphical components.

More specifically, there are several contexts in which it is important to improve the understanding of algorithm/model results and characteristics, such as the data mining [12, 11], data privacy [10, 17], and the deep learning [8]. To this end, in the literature, many approaches and tools have been proposed. 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 i) the tool in [24], which provides multiple views to visually inspect the overall set of ARs, and ii) the hierarchical matrix-based visualization technique presented in [9].

Concerning the discovery of RFDs, the availability of efficient RFD discovery algorithms yields the necessity to manage big result sets of RFDs, most of which differ only for some values of relaxation parameters. However, 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 a possible huge number of discovered RFDs. Among these, one of the most effective platforms for data profiling is the Metanome project [18], 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. Another 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 [6]. 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.

Although all of the above approaches represent effective tools to visualize and explore properties and metadata after the execution of mining/discovery algorithms, a recent proposal goes beyond the only result visualization problem [1], since it allows users to explore how RFDs change over time, and to perform result comparisons among different time-slots. The latter approach shares similar goals to our proposal. Nevertheless, it is mainly focused on the analysis of temporal trends related to the number of discovered RFDs. Instead, DEVICE is able to represent how discovery results change into the search space. This characteristic makes it also useful for the analysis of how different algorithms browse the search space.

3. Background

Before describing the proposed tool we will review some background definitions on the concept of RFD and the graph lattice representation.

A relational database schema $\mathcal{R}$ is defined as a collection of relation schemas ($R_1, \dots, R_n$), where each R_i is defined over a set $attr(R_i)$ of attributes ($A_1, \dots, A_m$). Each attribute A_k has associated a domain $dom(A_k)$, which can be finite or infinite. A relation instance (or simply a relation) r_i of R_i is a set of tuples such that for each attribute $A_k \in attr(R_i)$, $t[A_k] \in dom(A_k)$, $\forall\, t \in r_i$, where $t[A_k]$ denotes the projection of t onto A_k. A database instance r of $\mathcal{R}$ is a collection of relations ($r_1, \dots, r_n$), where r_i is a relation instance of R_i, for $i \in [1, n]$.

Aiming to improve the quality of database schemas and to reduce manipulation anomalies, in the context of relational databases, functional dependencies (FDs) have been used as means to guide data normalization processes. In particular, an FD over database schema $\mathcal{R}$ is a statement $X \rightarrow Y$ (X implies Y) defined between two sets of attributes $X, Y \subseteq attr(\mathcal{R})$, 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) [7]. In particular, RFDs enable the consideration of some relaxation criteria, which can lead to i) the consideration of similarity/difference constraints as attribute comparison method, and/or ii) the possibility that the dependency might hold for a subset rather than all the tuples.

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 applied on $\mathbb{D}_c \subseteq dom(R)$ is denoted by

$$X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2} \tag{1}$$

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 $dom(A_i)$;
- $X = B_1, \dots, B_h$ 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 $r \vDash \varphi$, 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 $\Psi \leq \varepsilon$.

As an example, in a database of publications, it is likely to have a similar Affiliation for authors with the same Email address and similar Name. In particular, the functional determination should tolerate possible exceptions since authors might change affiliation over the years. This can be modeled by means of the following RFD:

$$\text{Name}_{\leq 4}, \text{Email}_{\leq 0} \xrightarrow{\psi(\text{Name, Email, Affiliation}) \leq 0.03} \text{Affiliation}_{\leq 5}$$

where the comparison constraints (ϕ_1, ϕ_2, and ϕ_3) use the edit distance as function, the $\leq$ as comparison operator, and 4, 0, 5 as thresholds for Name, Email, and Affiliation, respectively. In general, the search space of the dependency discovery strategy can be modeled as a graph representation of a lattice, which is partitioned into levels where level L_i contains all attribute combinations of size i. Each node in the lattice represents a unique set of attributes, and it is linked to nodes that contain a direct superset or subset of attributes. In other words, each edge refers to the inclusion relation between two attribute sets. Thus, a lattice permits to consider candidate RFDs at each level in terms of lattice's edges, allowing to represent the LHS and the RHS of an RFD [19]. It is worth to notice that this representation is complete for FDs and RFDs relaxing on the extent. In fact, to discover RFDs relaxing on the attribute comparison it is necessary to also consider all possible dispositions of similarity/distance constraints among attributes included in a combination. Nevertheless, for this kind of RFDs we simplified the representation by visualizing the presence of an RFD as an edge in the lattice when there exists at least one valid RFD involving the same attribute combinations.

More formally, let $R = A_1, \dots, A_m$ be a relation schema with m attributes. The corresponding graph representation of lattice will contain a collection of attribute sets, where *Level 0* contains the empty set, *Level 1* singleton sets, one for each attribute, *Level 2* the pair sets, one for each possible combination of two attributes, and so forth. Finally, the last level, namely *Level M*, will contain a single set of all the attributes from R. A lattice edge links two attribute sets on two adjacent lattice levels. For instance, let AB and ABC be attribute sets on *Level 2* and *Level 3*, respectively, then the edge $e(AB, ABC)$ represents the candidate $AB \rightarrow C$.

The number of RFDs holding on a given set of data can be very huge, especially when relaxation criteria settings increase. Thus, aiming to provide compact representation on where the holding RFDs converge in the search space, we propose DEVICE, which is described in the next section.

4. A tool for analyzing RFDs during discovery processes

In this section, we describe DEVICE. It enables monitoring of RFDs validated during the execution of a discovery algorithm. In particular, we first present the system architecture (Section 4.1), and then provide details on the visual interface (Section 4.2), and the interactions that a user can perform (Section 4.3).

4.1. System architecture

Monitoring the RFD discovery results during the execution of discovery algorithms is a complex problem. In fact, it is necessary to deal with several issues that affect the choices of the system architecture: i) the existing discovery algorithms are based on different technologies and frameworks, so requiring the integration of at least one module to adapt the system to the different algorithms; ii) the presence of multiple visualization components expects frequent updates in a short time, and iii) the number of dependencies processed can be large at any instant of time. For these reasons, we proposed a modular client-server architecture for enabling users to monitor discovery algorithms during their executions, and interact with the results through a responsive visual interface. The architecture of the DEVICE is shown in Figure 1. In detail, the architecture is composed of several standalone modules, which share information with other ones by exploiting the JSON standard. This type of solution allows DEVICE to ensure high component modularity and

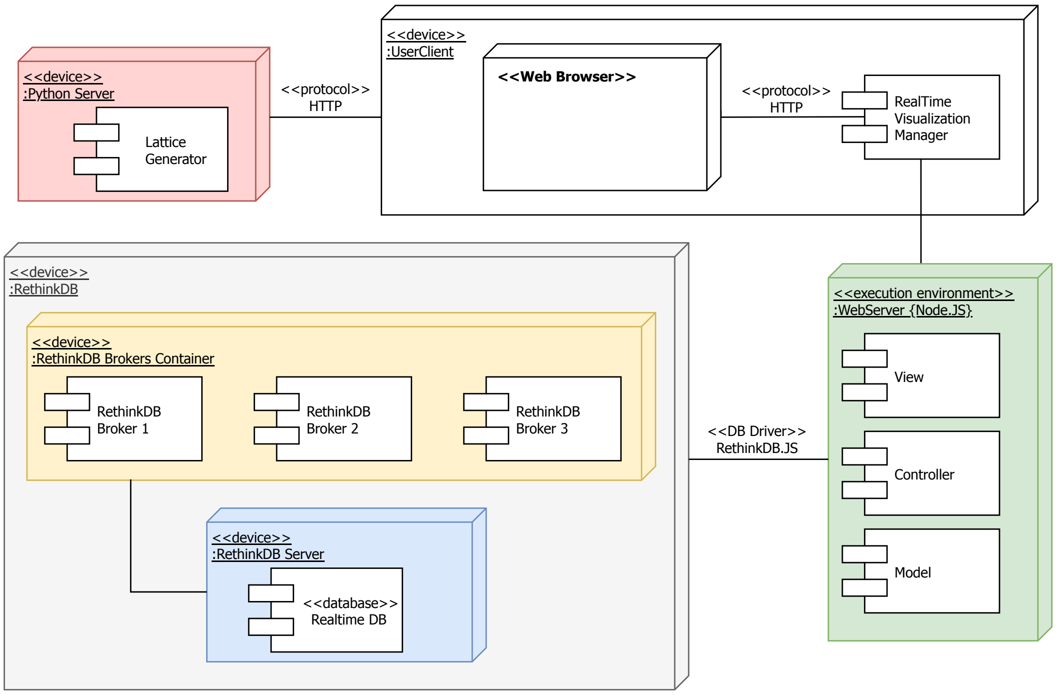


Figure 1: The system architecture of DEVICE.

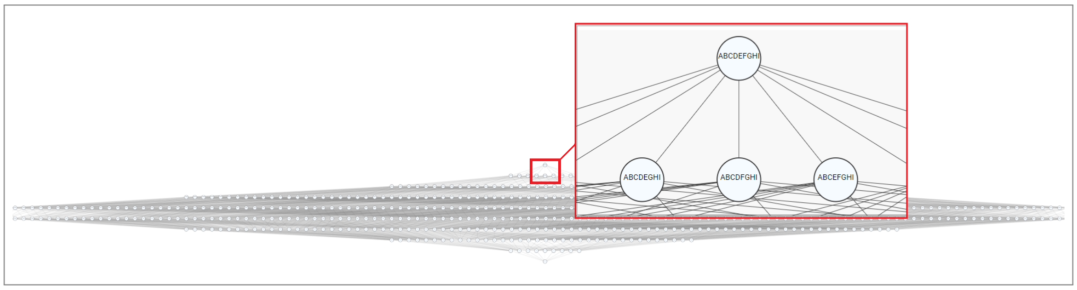


Figure 2: An example of lattice visualization.

maintainability, with the aim of updating or replacing any back-end module by simply adapting its output according to the JSON standard defined for the interaction.

The interface module on the client-side communicates with two different back-end subsystems. The first one allows DEVICE to automatically generate a lattice representation in JSON format by only considering the number of attributes. The set of nodes contains all the possible combinations of attributes on the lattice, while the set of edges contains all the existing links between two nodes of successive levels. The *Lattice Generator Server* receives a request containing the number of attributes for the lattice, creates the JSON, and returns its representation to DEVICE.

The second subsystem allows DEVICE to communicate with the discovery algorithms by exploiting a set of distributed message brokers. In particular, DEVICE is a web application distributed on multiple Node.JS server instances, which exploits the scalability of this technology combined with the speed of the RethinkDB[1] real-time database, to create a low latency and high-performance application. Although the architecture ensures flexibility, to make DEVICE compatible with most FD and RFD discovery algorithms, it has been necessary to integrate several communication modules to adapt the syntax of the dependencies of each algorithm, and to continuously monitor the results of each execution. To this end, the *Input Driver Connector* receives the dependencies from the algorithm, manipulates their syntax, and extracts a JSON version so as to store it in RethinkDB. The latter provides an internal set of message brokers that continuously store and send messages to the instance of Node.JS servers. The *Real-Time Visualization Manager* listens for messages from brokers, and decides which visual component manipulates in the interface.

As said before, the proposed tool is also able to handle continuous discovery algorithms [16] and therefore it requires to maximize fluidity and to minimize processing times

[1]https://rethinkdb.com

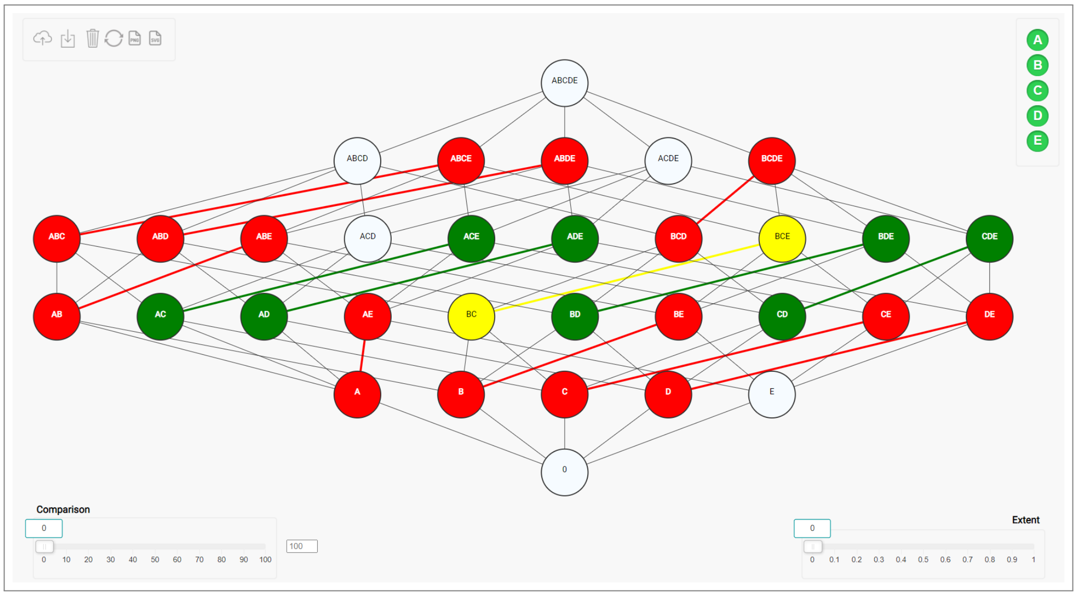


Figure 3: The visual interface of DEVICE during the execution of a discovery process.

within the visual interface. Thus, all selected technologies for both client- and server-side support real-time updating of data.

4.2. RFD Visualization

Due to the possible huge number of holding RFDs on a given set of processed data, systems for RFD visualization should enable users to analyze results in a compact way, by also giving the possibility to interact with them. For this reason, a static representation of discovery results after the execution of algorithms limits the analysis of how dependencies evolve over time, which is particularly interesting in dynamic contexts, like in the case of data streams.

The dynamic representation of a large portion of data requires the application of interactive graphs, capable of highlighting how information change over time. Hence, a dynamic visual representation of the search space has been implemented through a lattice graph representation. It enables a compact visualization on how holding RFDs converge into the search space (see Figure 8). As said before, the lattice permits to show candidate RFDs through the lattice edges, which connect attribute combinations differing by an attribute, so to represent in a compact way the LHS (common attributes) and the RHS (different attribute) of a candidate RFD. The lattice graph is responsible for displaying information about the candidate RFDs that have been validated during a discovery process. As shown in Figure 3, the lattice graph can show different colors during the execution of a discovery process. In particular, an edge is green when the corresponding candidate RFD has been evaluated and validated by the discovery algorithm. Instead, it assumes a red color when the RFD has been evaluated, but it is not valid. Finally, yellow edges represent candidate RFDs that are being analyzed. Aiming to emphasize the current validation results, DEVICE also uses colors for lattice nodes. In fact, a node assumes the same color as the last analyzed candidate RFD involving it.

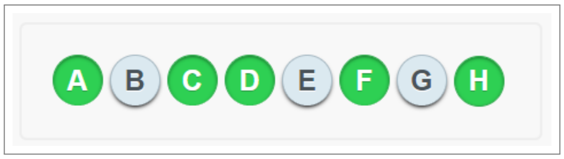
It is worth to notice that, although we expect that different algorithms produce the same resulting set of discovered RFDs, when they analyze the same data, it is not obvious how they move in the search space. Thus, DEVICE enables the comparison among different discovery algorithms and the analysis of possible bottlenecks during their execution on a given set of data.

The visual interface of DEVICE also provides different gadgets enabling users to interact with the lattice graph. Moreover, the vectorial representation of the graph also permits to zoom on or move each lattice component without losing the quality of the representation. Details on how users can interact with the lattice graph are provided in the following.

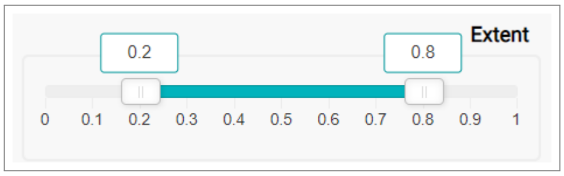
4.3. Interaction in depth

As mentioned above, aiming to emphasize the discovery results to a specific part of the search space, users can interact with the lattice graph by simply zooming on a specific part of the search space, or by moving its components into the visual interface. To this end, the user places the mouse pointer in correspondence with a lattice node and drags it in another place. Consequently, also edges linked to it are deformed by following the movement. Moreover, it is possible to filter out some nodes, so reducing the representation of the search space, by using the button list on the top-right of the visual interface (also shown in Figure 4(a)). In particular, each button represents an attribute of the analyzed data, and the user can select/deselect each of them to be included/excluded during the monitoring process. By default, all attributes appear in the search space. As an example, Figure 5 shows how the lattice is changed after the exclusion of the node A.

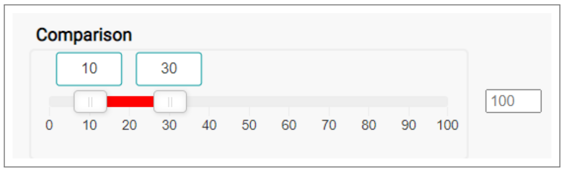
Concerning the RFD settings, DEVICE permits to visualize discovered RFDs by filtering results according to spe-



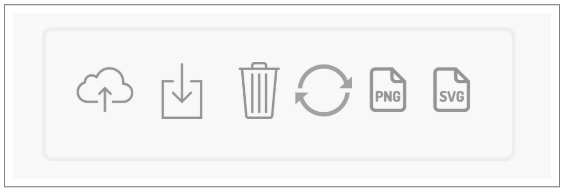
(a) Button list enabling the filtering of nodes in the lattice.



(b) Slider to define the range for the extent threshold.



(c) Slider to define the range the the comparison thresholds.



(d) Storing and execution icons.

Figure 4: DEVICE gadgets to interact with the lattice graph.

cific relaxation parameters. Figure 4(b) highlights a slider that enables the user in the definition of a specific range for the coverage measure threshold. In this way, lattice components' colors appear in accordance with the validation of RFDs having a satisfiability degree that meets the specified range bounds. Similarly, it is possible to filter out validation results in accordance with a range of thresholds composing difference constraints for the relaxation of the attribute comparison method (see Figure 4(c)). In particular, the bounds defined through the slider represent the range of possible thresholds that must appear on each attribute involved in candidate RFDs. However, as mentioned above, for sake of simplicity, a lattice edge is colored when at least one candidate RFD satisfies difference thresholds bounded by the range. Sliders are particularly useful for the analysis of RFD discovery results. In fact, with simple interactions, the user can evaluate how the set of holding RFDs can change as the relaxation settings are modified. Moreover, it is worth to notice that the interaction with sliders is enabled on the basis of the monitored algorithm. For instance, when a FD discovery algorithm is monitored, then sliders are set to $[0, 0]$ and cannot be modified. In this way, no errors and an exact comparison method (difference equal to 0) are admitted as RFD relaxation settings to represent FDs. In general, the same ranges are also used by default on both sliders, and it is possible to interact with them in accordance with the RFD category a discovery algorithm is devoted.

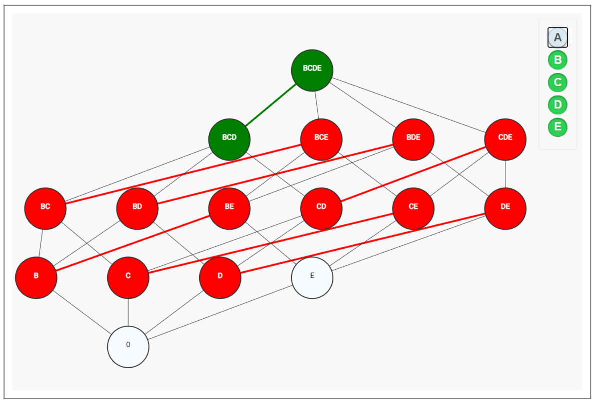


Figure 5: The visual interface of DEVICE after filtering out the attribute A.

Finally, the icons in Figure 4(d) enable the interaction with the monitored execution and the downloading of the lattice graph in several formats. In particular, the first two icons permit to upload or download the discovery results in a JSON file, respectively. The third and fourth icons enable the user to interact with the monitoring process. More specifically, the third icon permits to refresh the monitoring, by cleaning the lattice representation, and the fourth one gives the possibility to reload the lattice representation of the last execution of a discovery algorithm. Moreover, the colored lattice representation can be downloaded as an image in the *.png* or *.svg* format by using the second-last and the last icon, respectively.

5. Monitoring discovery algorithms

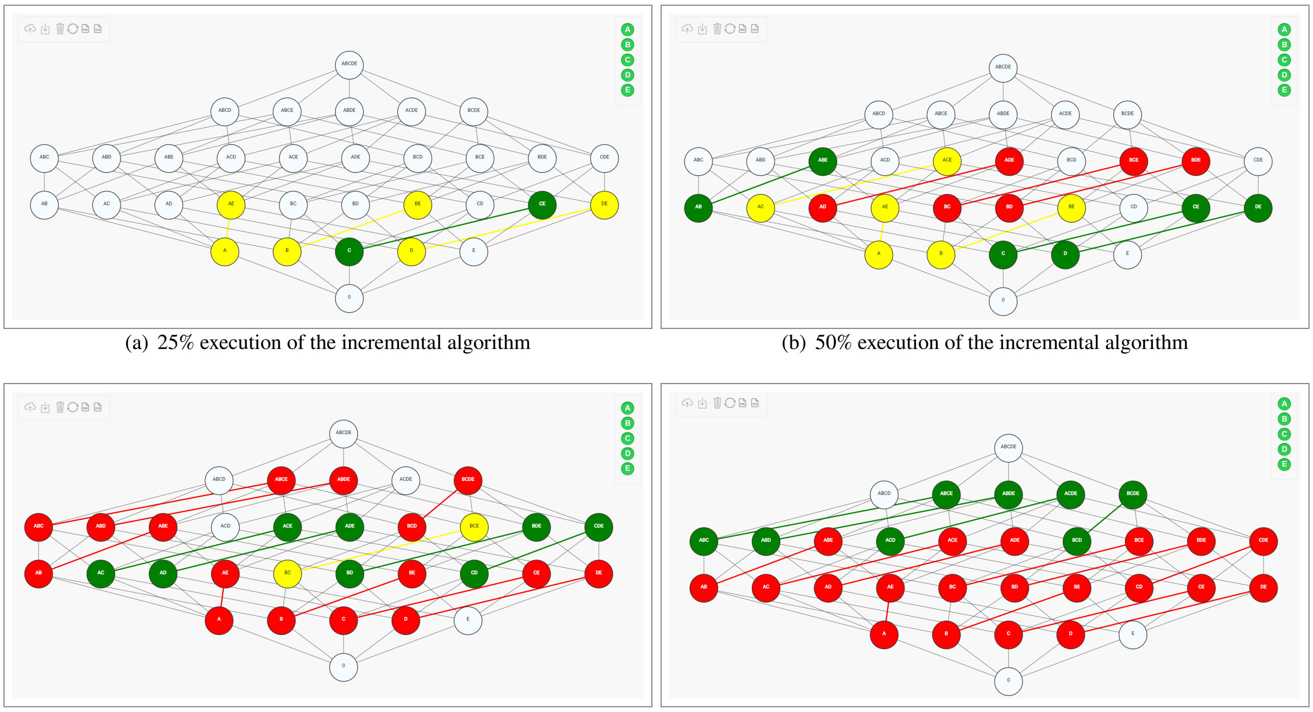
In this section, we show the effectiveness of DEVICE on different algorithms, by considering two different case studies on real-world datasets and on real sensor-based streams, with the aim of analyzing how metadata evolves over time. Moreover, a demonstration video of DEVICE[2] allows us to show how the users can interact with the tool during monitoring processes.

5.1. Case study on a real-world dataset

In order to verify the effectiveness of DEVICE on a real scenario, we analyzed discovery results on a real-world dataset. We selected two different discovery algorithms to analyze how their discovery strategy browses the search space, aiming to extract the holding RFDs.

The first algorithm involved in our evaluation is the genetic algorithm proposed in [5]. This type of algorithm has the potential to effectively tackle the problems arising in RFDs discovery, since they are particularly efficient for global searches in large search spaces by exploiting operations inspired to natural species evolutions, such as natural selection, crossover, and mutation. The discovery process starts with a population of RFD candidates which is randomly generated during the initialization phase and evolves by stochastically selecting

[2]https://youtu.be/QC2FjF50A60



(a) 25% execution of the incremental algorithm

(b) 50% execution of the incremental algorithm

(c) 75% execution of the incremental algorithm

(d) 100% execution of the incremental algorithm

Figure 6: Monitoring the incremental discovery algorithm during its executions.

multiple candidates from the current population. The algorithm exploits a fitness function to quickly validate each RFD and select the ones to involve during the evolution phases.

The second algorithm exploits incremental discovery strategies to extract functional dependencies from static and/or dynamic datasets [3]. Unlike the genetic algorithm, the incremental approach takes in input a set of candidates valid at a given instant of time, and returns the dependencies valid after updating at least one tuple. However, during the initial execution, the algorithm starts by considering the set of FDs candidates at the lower level of lattice, i.e. all the FDs with a single attribute on the LHS, and performs an upward search strategy of the lattice.

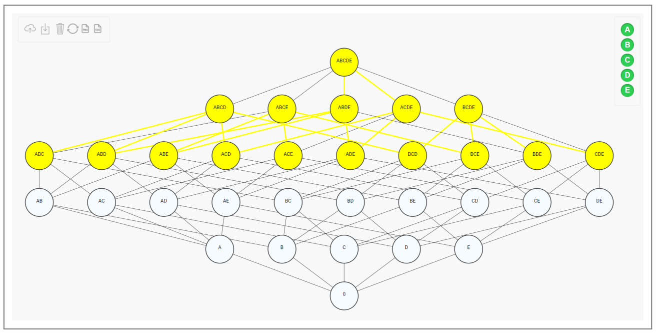
According to the characteristics of the considered algorithms, and to make processes comparable, we set parameters of genetic algorithm to discover canonical FDs. In particular, we choose the above-mentioned types of algorithms since they both use several iterations to get results. Nevertheless, their nature is quite different since the genetic algorithm analyzes new RFD candidates at each iteration by always considering the complete set of tuples; instead, the incremental algorithm analyzes new tuples at each iteration by considering the RFDs holding at the previous iteration (time-instant). Our aim is to show the usefulness of DEVICE in helping users to get insights on how algorithms can explore the search space.

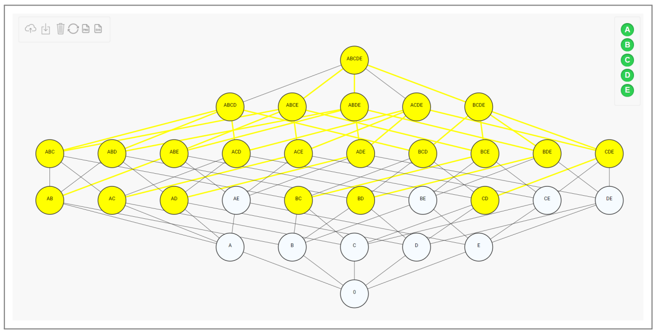
Although these algorithms have been created with two different technologies, the *Input Driver Connector* allowed us to quickly adapt their output modules to DEVICE. In fact, this enabled us to monitor their executions on the same dataset, and compare how they browse the search space. To perform our evaluation, we ran each algorithm on the *Iris* dataset by automatically storing the screen of the lattice approximately every 1 second. Each screen represents the status of the lattice at any instant of execution time. For the sake of clarity, we only report the screens at 25%, 50%, 75%, and 100% of their executions (Figures 6 and 7).

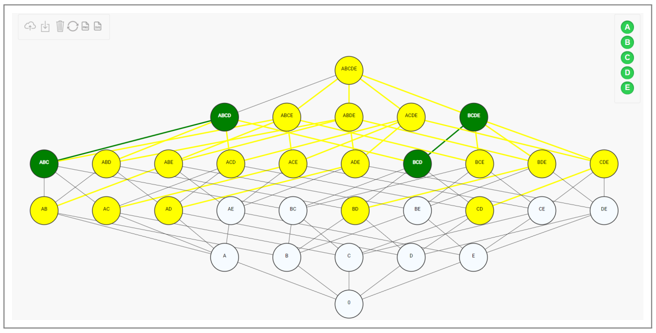
More specifically, figures 6(a) and 7(a) show the evolution of the discovery process for the incremental and genetic algorithms, respectively, at the 25% of their executions. We can notice the difference between the two discovery strategies. In fact, the genetic algorithm starts to consider candidates in the middle of the lattice, and next perform a random upward search. However, the incremental algorithm first selects candidates from the lowest level, then goes up by performing a targeted search.

Another relevant difference between the two search strategies concerns the validation strategy of the candidates. In fact, the genetic algorithm exploits an a posteriori validation strategy of the RFD candidates, which allows to define the first valid and invalid dependencies only after 50% of the execution (Figures 7(b), 7(c), and 7(d)). On the contrary, the incremental algorithm updates the RFDs validated at the earliest executions according to the dynamic change of the dataset, and exploits this information to move on the search space (Figures 6(b), 6(c), and 6(d)). However, as expected when algorithms end the executions (Figures 6(d) and 7(d)) they obtain the same set of resulting RFDs.

DEVICE provides a concrete representation of the discovery algorithms, allowing users and domain experts to easily


(a) 25% execution of the genetic algorithm


(b) 50% execution of the genetic algorithm


(c) 75% execution of the genetic algorithm

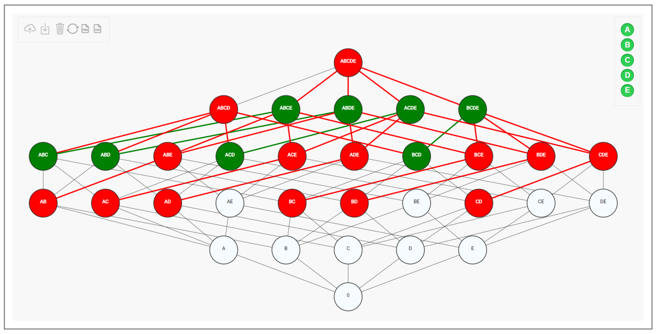

(d) 100% execution of the genetic algorithm

Figure 7: Monitoring the genetic discovery algorithm during its executions.

monitor each execution phase, and to concretely compare the different search strategies.

5.2. Case study on a real-world data stream

In our last experiment, we show the usefulness of DEVICE on a real-world data stream. In particular, we executed the algorithm in [3] on data from 1, 000 real sensors spread throughout Italy, made available by the *Openweathermap* portal[3]. These types of sensors share information about the weather forecast during the day. The data are frequently updated based on global and local weather models, satellites, radars, and a vast network of weather stations. In particular, we selected the following 8 attributes from the data stream:

- Temperature represents the temperature value in the Kelvin scale (K);
- Feels_like represents the human perception of weather in Kelvin scale (K);
- Sea_level represents the atmospheric pressure on the sea level (hPa);
- Grnd_level represents the atmospheric pressure on the ground level (hPa);
- Humidity represents the rate of humidity;
- Date represents the date of the weather forecast;
- Weather represents the weather condition (e.g. Rain, Snow, Extreme, etc.);
- Clouds_percentage represents the rate of cloud cover.

We considered a single execution of the algorithm on weather data streams lasting 4 days. The execution involved over 40, 000 tuples shared by over 1, 000 sensors. During the test, DEVICE continuously monitored the progress of the discovery algorithm, also storing the results and its status for different time intervals. Figure 8 shows the resulting FDs for each time interval. We can notice that the number of resulting FDs has a negative trend since the continuous insertion of new tuples has lead to many invalidations. Moreover, the algorithm in [3] incrementally discovers FDs, which require to be validated on the entire stream. This means that the initial number of FDs, i.e. dependencies involving few attributes, will probably evolve as the algorithm considers new tuples. To get some insights on the FD validation trend, DEVICE allows us to interact with its interface and explore the search space to concretely analyze how FDs evolved in this process.

Figure 9 shows the details of the discovery process by considering three different time intervals, 3, 48, and 96 hours, respectively. As expected, DEVICE shows that the algorithm has a large variation in the number of FDs after 3 hours and a small number of invalid FDs (Figure 9(a)). Moreover, as we can see, a relevant part of the search space has not been analyzed. This is due to the fact that the discovery strategy has already validated some minimal FDs, avoiding the analysis of candidates that can be directly inferred. Figure 9(b) and 9(c) show that many of the FDs validated after 3 hours, have been invalidated. Moreover, Figure 9(c) shows that the algorithm also analyzed many of the candidate FDs in the search space not analyzed before. This is due to the invalidation of

[3]https://openweathermap.org/

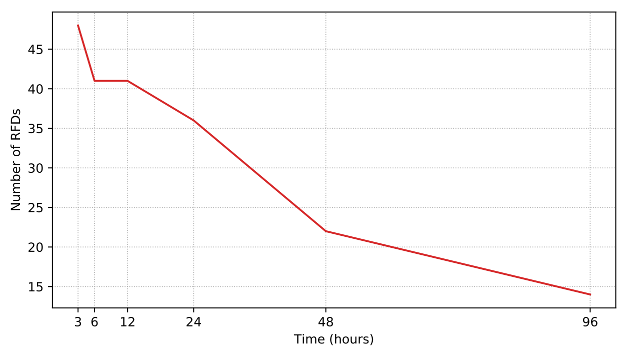


Figure 8: Resulting RFDs from the executions on real streams.

many dependencies on the right side of the search space. In fact, after 96 hours, only 14 FDs have been validated.

The evaluation performed on these real-world streams permits to understand how this kind of tool is able to support users and domain experts in the analysis of correlations holding on data streams. In fact, at each instant, an expert can concretely visualize and evaluate discovery results, and s/he can also monitor the evolution of holding RFDs over time. Moreover, the different gadgets embedded in DEVICE support users to interact with results during the monitoring process. For instance, the zoom feature (see Figure 9(d)) permits to focus the monitoring only on a specific part of the search space; instead, the filter feature (see Figure 9(e)) permits to isolate a specific set of RFD candidates. These two features enable to perform detailed analysis in order to consider the possibility to re-execute discovery processes on the same stream configurations, but with a reduced set of attributes. In general, these kinds of interactions could allow users to reduce the complexity of the analysis especially when they have to monitor big datasets and/or data streams.

6. Conclusion and Future Directions

Information visualization techniques aim, among others, to facilitate analytical processes and to reduce their interpretation complexity by exploiting, possibly specific or novel, visual representations. Nevertheless, the current big data contexts entail several challenging scenarios, where data are dynamically produced. In particular, in the context of dependency discovery from data streams (i.e., continuous profiling), dynamic data might produce the evolution of many FDs and/or RFDs. Thus, it is necessary not only to adequate discovery algorithms to fast execution processes, but also to allow users to analyze holding RFDs, and how they change over time. To this end, we have proposed the tool DEVICE, which relies on a lattice graph representation of the search space to let users actively visualize holding RFDs in a compact way during the execution of (incremental) RFD discovery algorithms. DEVICE also represents a useful means to compare different discovery algorithms, and to analyze how they browse the search space.

In the future, we would like to test the usability of DEVICE, by involving domain experts and scientists in the interpretation of discovery results on both incremental scenarios and algorithm comparison tasks. Thus, based on these results, we would like to extend DEVICE to better support its usefulness in the analysis tasks. Moreover, we would like to lighten the representation of the search space, when it has to represent big datasets. To this end, we are working on different grouping functionalities, which would enable the lattice graph with the possibility to dynamically change its shape according to the RFDs validated over time.

References

[1] Breve, B., Caruccio, L., Cirillo, S., Deufemia, V., Polese, G., 2020. Visualizing dependencies during incremental discovery processes., in: Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference.

[2] Caruccio, L., Cirillo, S., 2020. Incremental discovery of imprecise functional dependencies. Journal of Data and Information Quality (JDIQ) 12, 19:1–19:25.

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

[4] Caruccio, L., Deufemia, V., Polese, G., 2016. On the discovery of relaxed functional dependencies, in: Proceedings of the 20th International Database Engineering & Applications Symposium, IDEAS 2016, Montreal, QC, Canada, July 11-13, 2016, ACM. pp. 53–61.

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

[6] Caruccio, L., Deufemia, V., Polese, G., 2019b. Visualization of (multimedia) dependencies from big data. Multimedia Tools and Applications 78, 33151–33167.

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

[8] Chakraborty, S., Tomsett, R., Raghavendra, R., Harborne, D., Alzantot, M., Cerutti, F., Srivastava, M., Preece, A., Julier, S., Rao, R.M., et al., 2017. Interpretability of deep learning models: a survey of results, in: 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), IEEE. pp. 1–6.

[9] Chen, W., Xie, C., Shang, P., Peng, Q., 2017. Visual analysis of user-driven association rule mining. Journal of Visual Languages and Computing 42, 76–85.

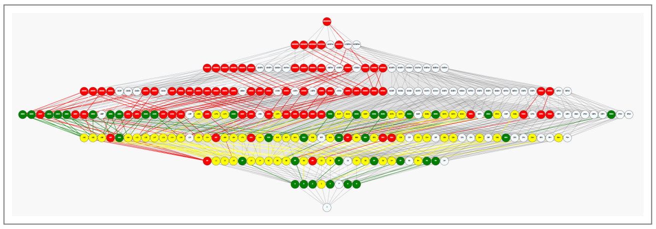
[10] Cirillo, S., Desiato, D., Breve, B., 2019. CHRAVAT – chronology awareness visual analytic tool, in: 2019 23rd International Conference Information Visualisation (IV), IEEE. pp. 255–260.

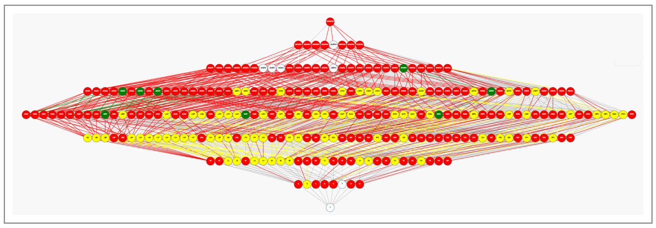
[11] Costagliola, G., Fuccella, V., Giordano, M., Polese, G., 2008. Monitoring online tests through data visualization. IEEE Transactions on Knowledge and Data Engineering 21, 773–784.

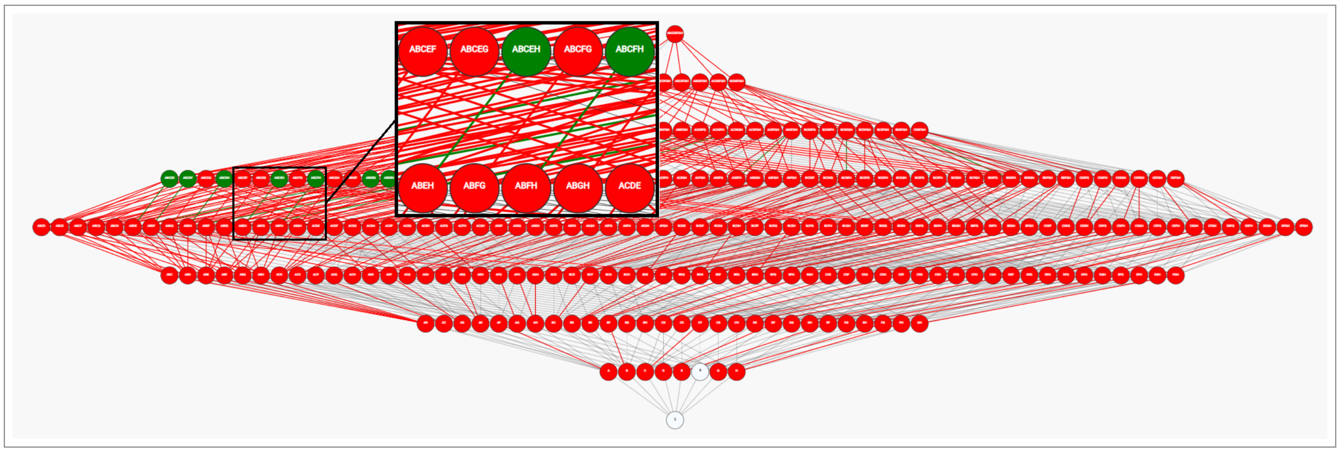
[12] De Oliveira, M.F., Levkowitz, H., 2003. From visual data exploration to visual data mining: a survey. IEEE Transactions on Visualization and Computer Graphics 9, 378–394.

[13] Di Rocco, L., Dassereto, F., Bertolotto, M., Buscaldi, D., Catania, B., Guerrini, G., 2020. Sherloc: a knowledge-driven algorithm for geolocating microblog messages at sub-city level. International Journal of Geographical Information Science , 1–32.

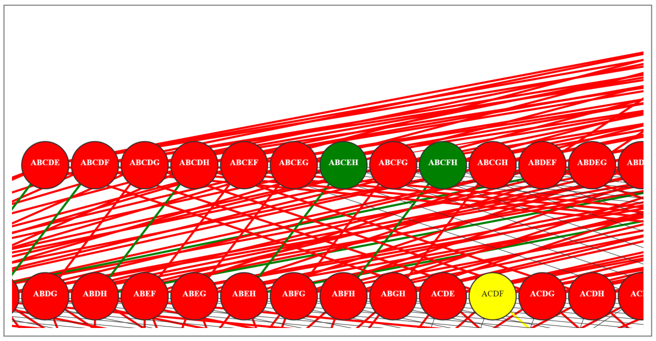
[14] Kruse, S., Hahn, D., Walter, M., Naumann, F., 2017. Metacrate: Organize and analyze millions of data profiles, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, ACM. pp. 2483–2486.


(a) Monitoring interface after 3 hours.


(b) Monitoring interface after 48 hours.



(c) Monitoring interface after 96 hours.



(d) Monitoring interface after zooming.

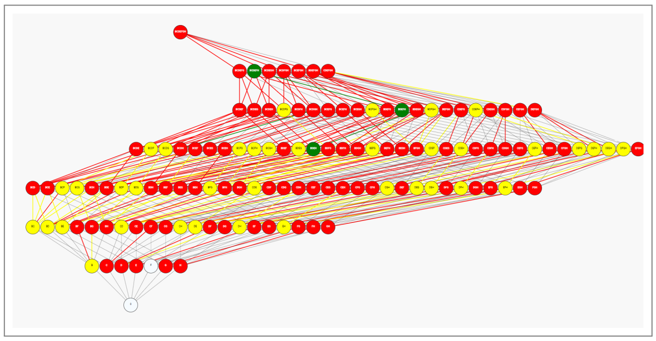

(e) Monitoring interface after filtering.

Figure 9: Monitoring the incremental discovery algorithm during its executions on real streams.

[15] Kruse, S., Naumann, F., 2018. Efficient discovery of approximate dependencies. Proceedings of the VLDB Endowment 11, 759–772.

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

[17] Nicolazzo, S., Nocera, A., Ursino, D., Virgili, L., 2020. A privacy-preserving approach to prevent feature disclosure in an IoT scenario. Future Generation Computer Systems 105, 502–519.

[18] Papenbrock, T., Bergmann, T., Finke, M., Zwiener, J., Naumann, F., 2015a. Data profiling with metanome. Proceedings of the VLDB Endowment 8, 1860–1863.

[19] Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J.P., Schönberg, M., Zwiener, J., Naumann, F., 2015b. Functional dependency discovery: An experimental evaluation of seven algorithms. Proceedings of the VLDB Endowment 8, 1082–1093.

[20] Papenbrock, T., Naumann, F., 2016. A hybrid approach to functional dependency discovery, in: Proceedings of the 2016 International Conference on Management of Data, ACM. pp. 821–833.

[21] Raghav, R., Pothula, S., Vengattaraman, T., Ponnurangam, D., 2016. A survey of data visualization tools for analyzing large volume of data in big data platform, in: Proceedings of the 2016 International Conference on Communication and Electronics Systems (ICCES), IEEE. pp. 1–6.

[22] Saxena, H., Golab, L., Ilyas, I.F., 2019. Distributed discovery of functional dependencies, in: IEEE 35th International Conference on Data Engineering, IEEE. pp. 1590–1593.

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

[24] Sekhavat, Y.A., Hoeber, O., 2013. Visualizing association rules using linked matrix, graph, and detail views. International Journal of Intelligence Science 3, 34–49.

[25] Song, S., Chen, L., 2013. Efficient discovery of similarity constraints for matching dependencies. Data & Knowledge Engineering 87, 146–166.

[26] Wei, Z., Link, S., 2019. Discovery and ranking of functional dependencies, in: IEEE 35th International Conference on Data Engineering, IEEE. pp. 1526–1537.