La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI 2014IT16M2OP005),
Fondo Sociale Europeo, Azione I.1 "Dottorati Innovativi con caratterizzazione Industriale"

UNIONE EUROPEA
Fondo Sociale Europeo

Università degli Studi di Salerno

Dipartimento di Informatica

Dottorato di Ricerca in Informatica
XXXIV Ciclo

TESI DI DOTTORATO / PH.D. THESIS

Data Stream Profiling: Evolutionary and Incremental Algorithms for Dependency Discovery

STEFANO CIRILLO

SUPERVISOR: **PROF. GIUSEPPE POLESE**

PHD PROGRAM DIRECTOR: **PROF. ANDREA DE LUCIA**

A.A 2020/2021

Stefano Cirillo
Data Stream Profiling: Evolutionary and Incremental algorithms for Dependency Discovery
Ph.D. Thesis
Reviewers: Prof. Felix Naumann and Prof. Domenico Beneventano
Supervisor: Prof. Giuseppe Polese
Co-Advisors: Prof. Vincenzo Deufemia and Prof.ssa Loredana Caruccio
Ph.D. Program Director: Prof. Andrea De Lucia

Università degli Studi di Salerno
Data Science and Technologies Laboratory
Department of Computer Science
Via Giovanni Paolo II, 132
84084 Fisciano (SA), Italy
© October 2021

// ABSTRACT

Data Profiling represents one of the most crucial processes in data quality assessment. It includes a set of activities to efficiently analyze datasets and provide insights from them. Such activities rely on the identification of metadata to capture semantic relationships within data, and can be exploited for several purposes, such as optimizing queries, cleaning data, evaluating feasibility of machine learning models, and so forth. The types of metadata range from simple counters of attribute values or null values, to complex integrity constraints, such as functional dependencies (FDs), relaxed functional dependencies (RFDs), and inclusion dependencies (INDs). However, the discovery of these metadata represents an important challenge for data profiling tasks, since the number of possible metadata can be exponential with respect to the number of attributes, and requires analyzing a huge number of attribute combinations. To this end, several discovery algorithms have been proposed in the literature, with the aim of providing solutions in which the complexity of the search space is reduced by exploiting some theoretical properties of the different types of metadata. Although some of the discovery algorithms described in the literature achieve good performances, most of them are not suitable in dynamic scenarios, in which new data are frequently added and updated into the datasets. This need is widely growning with the proliferation of the Internet of Things (IoT) technologies, since it is necessary to define new algorithms capable of dynamically analyzing the streams of data they produce.

In this scenario, after reviewing basic data profiling tasks and applications, as well as basic notations for representing profiling metadata, this thesis starts presenting an innovative tool that extracts metadata from unstructured web data sources, aiming to derive a focused crawler. Then, the thesis focuses on the discovery problem of FDs and RFDs in static and dynamic scenarios, by analyzing their complexities and by introducing several new incremental methodologies and algorithms for discovering

FDs and RFDs, aiming to avoid the re-execution of the discovery process from scratch upon update operations on datasets. In particular, a first proposal is an evolutionary discovery algorithm for hybrid RFDs named REDEVO (RElaxed fD EVOlutionary discovery algorithm), which uses naturally inspired operations to iteratively browse candidates over the search space, few of which survive the evolution process. It identifies a broad class of RFDs, by evaluating each candidate by means of support and confidence quality measures as a fitness function.

Then, this thesis presents four new discovery algorithms for FDs and RFDs in dynamic scenarios. The first algorithm is named INCREMENTAL-FD, which is able to update the set of holding FDs upon insertions of new tuples to the data instance, without having to restart the discovery process from scratch. It exploits a bit-vector representation of FDs, and an upward/downward search strategy, aiming to reduce the overall search space. The algorithm represents the baseline for the definition of a second incremental algorithm for discovering FDs, named REXY (RegEX-based incremental discoverY). The latter adopts a new validation method that exploits Regular Expressions (RegExs) to improve the validation process for each FD candidate, by restricting the search to a subset of data. The third algorithm is named COD3, an efficient and incremental algorithm for discovering FDs holding on data streams. To the best of our knowledge, COD3 represents the first proposal to use a non-blocking architectural model to face the problem of FD discovery from data streams. It relies on a novel data structure, named Validation Graph, which enables a fast exploration of the search space according to the discovered FDs, leading to a new fast FD validation process. The last algorithm presented in this thesis is BIRD (Bit-vector based Incremental RFD_e Discoverer), which tackles the problem of the incremental discovery of RFDs. BIRD analyzes how new inserted tuples impact on candidate RFDs, checking whether they invalidate some previously holding ones, and possibly generating new candidates. Moreover, BIRD is able to split the discovery process into level-wise parallel executions.

Finally, the thesis describes three new tools designed and developed for monitoring incremental discovery algorithms during their executions. These tools enable users to properly visualize the trend of FDs and

RFDs discovered over time, provide an overview at each time instant of the correlation between attributes included in the discovery results, compare FDs and RFDs resulting from different executions of the discovery algorithms, and directly manipulate discovery results through visual metaphors.

ABSTRACT

I processi di Data Profiling rappresentano uno strumento chiave per supportare la valutazione della qualità dei dati. Questi si articolano in attività rivolte all'analisi efficiente di insiemi di dati al fine di estrarre informazioni utili dagli stessi. Tali attività permettono di catturare relazioni semantiche all'interno dei dati attraverso l'estrazione di metadati, i quali possono essere sfruttati per diversi scopi, come l'ottimizzazione delle query, la pulizia dei dati e la valutazione dei modelli di machine learning. I tipi di metadati possono variare da semplici contatori sul numero di valori nulli o di valori distinti in un dataset, a vincoli di integrità, come dipendenze funzionali (Functional Dependency - FD), dipendenze funzionali rilassate (Relaxed Functional Dependency - RFD) e dipendenze di inclusione (Inclusion Dependency - IND). Tuttavia, la complessità del problema di discovery automatico di questi metadati rappresenta una delle principali sfide per le attività di data profiling, poiché il numero di possibili metadati potrebbe essere esponenziale rispetto al numero di attributi dei dataset, considerando la quantità di combinazioni di attributi da analizzare. A tal fine, sono stati proposti in letteratura diversi algoritmi per il loro discovery automatico dai dati, con l'obiettivo di fornire soluzioni in cui la complessità dello spazio di ricerca viene ridotta sfruttando alcune proprietà teoriche dei diversi tipi di metadati. Sebbene alcuni di questi algoritmi di discovery raggiungano buone prestazioni, la maggior parte di essi non è in grado di effettuare processi di discovery automatico in scenari dinamici, in cui si considera la possibilità che i dati vengano frequentemente aggiornati. Questa esigenza è ampiamente cresciuta con la diffusione dell'Internet of Things (IoT), poiché gli algoritmi dovrebbero essere in grado di analizzare dinamicamente i flussi di dati che questi strumenti intelligenti producono.

In questo scenario, dopo aver esaminato le attività e le applicazioni del data profiling, e dopo aver introdotto le notazioni di base per la rappresentazione dei metadati, questa tesi presenta uno strumento innovativo

che estrae i metadati da sorgenti di dati Web non strutturati, con l'obiettivo di derivare un crawler mirato. Successivamente, la tesi si concentra sul problema del discovery di FD e RFD in scenari statici e dinamici, analizzando la complessità del problema di discovery e introducendo diverse nuove metodologie e algoritmi incrementali per l'estrazione automatica di FD e RFD, con l'obiettivo di evitare la riesecuzione del processo di discovery dall'inizio dopo le operazioni di aggiornamento dei dati. In particolare, una prima proposta presentata è un algoritmo di discovery evoluzionistico per RFD ibride chiamato REDEVO (RElaxed fD EVOlutionary discovery algorithm), che utilizza operazioni ispirate alla selezione naturale delle specie per analizzare iterativamente insiemi di dipendenze candidate che evolvono, poche delle quali sopravviveranno al processo di evoluzione. REDEVO permette di identificare un'ampia classe di RFD, effettuando la validazione di ciascuna candidata mediante le misure di support e confidence, le quali definiscono la funzione di fitness.

Successivamente, questa tesi presenta quattro nuovi algoritmi di discovery di FD e RFD in scenari dinamici. Il primo algoritmo è denominato INCREMENTAL-FD, il quale è in grado di aggiornare l'insieme di FD valide dopo l'inserimento di nuove tuple nei dati, senza dover rieseguire completamente il processo di discovey. L'algoritmo sfrutta una rappresentazione vettoriale binaria delle FD e una strategia di ricerca in discesa/salita, con l'obiettivo di ridurre lo spazio di ricerca da analizzare. L'algoritmo INCREMENTAL-FD rappresenta la baseline per la definizione di un secondo algoritmo incrementale per il discovery di FD, chiamato REXY (RegEX-based incremental discoverY). Quest'ultimo adotta un nuovo metodo di validazione che sfrutta le espressioni regolari (RegEx) per migliorare il processo di validazione di ogni FD candidata, limitando la validazione al sottoinsieme di dati interessato dalle modifiche. Il terzo algoritmo incrementale, denominato COD3, permette di effettuare il discovery di FD su data stream. Al meglio della nostra conoscenza, COD3 rappresenta la prima proposta in letteratura che utilizza un modello architetturale non bloccante per affrontare il problema di discovery di FD dai flussi di dati. Esso pone le sua fondamenta su una nuova struttura dati, denominata Validation Graph, che consente una rapida esplorazione dello spazio di ricerca in base alle FD precedentemente inferite dai dati, e attaver-

so la quale è stato possibile definire un nuovo ed efficente processo di validazione di FD. L'ultimo algoritmo presentato in questa tesi è BIRD (Bit-vector based Incremental RFD_e Discoverer), che affronta il problema del discovery incrementale di RFD. BIRD analizza come le nuove tuple inserite impattano sulle RFD, verificando se tali tuple comportano l'invalidazione di alcune RFD già presenti ed eventualmente generando nuove RFD candidate. Inoltre, BIRD è in grado di suddividere il processo di discovery in esecuzioni parallele per ogni livello dello spazio di ricerca.

Infine, la tesi presenta tre nuovi tool progettati e sviluppati per monitorare i processi di discovery incrementali durante l'esecuzione degli algoritmi. Questi strumenti consentono agli utenti di visualizzare continuamente l'andamento e l'evoluzione delle FD e RFD estratte dai dati nel corso del tempo, di analizzare in ogni istante temporale la correlazione tra gli attributi inclusi nei risultati di discovery, e di confrontare FD e RFD estratte in diverse esecuzioni degli algoritmi e di manipolarle mediante nuove metafore visuali.

CONTENTS

1 INTRODUCTION 1

I CONTEXT & BACKGROUND

2 DATA PROFILING 9
 2.1 Profiling Tasks and Applications 9
 2.2 Profiling Metadata 13
  2.2.1 Unique Column Combinations (UCC) 14
  2.2.2 Functional Dependencies (FD) 15
  2.2.3 Relaxed Functional Dependencies (RFD) 17
  2.2.4 Inclusion Dependencies (IND) 22
 2.3 Open Challenges 23
  2.3.1 Profiling Heterogeneous Data 25
  2.3.2 Incremental Data Profiling 27
  2.3.3 Interactive Data Profiling 28

3 PROFILING UNSTRUCTURED WEB DATA SOURCES 31
 3.1 Problem Description 31
 3.2 Literature Review 33
 3.3 CAIMANS: Crawling Artifacts of Interest and Matching them Against Enterprise Sources 34
  3.3.1 Extraction and Profiling of artifacts from the web 35
  3.3.2 Experimental Results 47

II DISCOVERY ALGORITHMS FOR DATA PROFILING

4 DISCOVERY ALGORITHMS IN STATIC SCENARIOS 61
 4.1 Problem Description 61
 4.2 Literature Review 64
 4.3 A Genetic Approach for Discovering Hybrid RFDs 67
  4.3.1 Methodology 68
  4.3.2 Generation of the Initial Population 71
  4.3.3 Fitness Function 73
  4.3.4 Crossover 75

4.3.5 Mutation 77
4.3.6 The REDEVO Algorithm 79
4.3.7 Experimental Evaluation 87
4.3.8 Evaluation on configuration settings 92
5 DISCOVERY ALGORITHMS IN DYNAMIC SCENARIOS 97
5.1 Problem Description 97
5.1.1 Incremental Discovery of FDS 98
5.1.2 Continuous Discovery of FDS from dynamic sources 101
5.1.3 Incremental Discovery of RFD_cs 104
5.2 Literature Review 107
5.3 INCREMENTAL-FD: Incremental discovery algorithm of FDS 110
5.3.1 Methodology 110
5.3.2 Experimental Evaluation 116
5.4 REXY: An Incremental discovery algorithm of FDS 122
5.4.1 Methodology 122
5.4.2 The REXY Algorithm 126
5.4.3 Experimental Evaluation 129
5.5 COD3: Continuous Discovery of FD from Data Streams 133
5.5.1 Methodology 134
5.5.2 Graph-based FD Validation 140
5.5.3 The COD3 Algorithm 147
5.5.4 Experimental Evaluation 152
5.6 BIRD: An Incremental discovery algorithm of RFD_cs 164
5.6.1 Methodology 164
5.6.2 The BIRD Algorithm 170
5.6.3 Theoretical Evaluation 178
5.6.4 Experimental Evaluation 180
III TOOLS FOR VISUALIZING PROFILING METADATA
6 VISUALIZATION AND MONITORING TOOLS FOR INCREMENTAL DISCOVERY ALGORITHMS 191
6.1 Problem Description 191
6.2 Literature Review 193
6.3 DEVICE: A tool for monitoring the evolution of results of RFD discovery algorithms 194

6.3.1 System Overview 194
6.3.2 RFD Visualization 197
6.3.3 Interaction in depth 198
6.3.4 Case Studies 201
6.4 STRADYVAR: Dependency Visualization in Data Stream Profiling 209
6.4.1 System Overview 210
6.4.2 RFD visualization 212
6.4.3 Interaction in depth 214
6.4.4 User Study 220
6.5 INDITIO: Real-time validation of profiling metadata in a data management system 224
6.5.1 System Overview 225
6.5.2 Interaction in depth 226
6.5.3 User Study 230

IV CONCLUSION

7 CONCLUSION AND FUTURE WORK 241

BIBLIOGRAPHY 245

LIST OF FIGURES

Figure 2.1 Classification of traditional data profiling tasks [1]. 11
Figure 2.2 Relaxation criteria for RFDs [32]. 19
Figure 2.3 New directions of data profiling [118]. 24
Figure 3.1 The architecture of CAIMANS. 37
Figure 3.2 Flowchart of the Crawler module. 38
Figure 3.3 Flowchart of the Semantic Module. 44
Figure 3.4 Elbow diagram. 46
Figure 3.5 Silhouette diagram. 47
Figure 3.6 Results produced by CAIMANS for the e-procurement case study. 48
Figure 3.7 Displaying the result of the clustering phase. 49
Figure 3.8 Semantic Search form - results. 49
Figure 3.9 Precision evaluation for Google and CAIMANS. 54
Figure 3.10 Recall evaluation for Google and CAIMANS. 55
Figure 4.1 Flowchart of REDEVO. 70
Figure 4.2 Creating a pattern map from a snippet of the *Breast-Cancer* dataset. 71
Figure 4.3 Encoding of attributes yielding the representation of candidate RFDs. 72
Figure 4.4 An example of crossover for candidate RFDs. 76
Figure 4.5 An example of mutation for two candidate RFDs. 78
Figure 4.6 Time performances by varying tuple comparison and extent thresholds. 90
Figure 4.7 Number of RFDs by varying tuple comparison and extent thresholds. 91
Figure 4.8 Variation of genetic configuration parameters. 93
Figure 4.9 Comparative evaluation on Fd-Reduced dataset by varying the number of attributes. 94

Figure 4.10 Comparative evaluation on Fd-Reduced dataset by varying the number of tuples. 95
Figure 5.1 Binary vector representation of a functional dependency. 111
Figure 5.2 The lattice search space representation for the attribute set {A,B,C,D,E}. 111
Figure 5.3 Discovery steps on the lattice search space representation. 112
Figure 5.4 The *linked map* related to Example 1. 115
Figure 5.5 A comparison between execution times of TANE with respect to our proposal by varying the percentage of inserted tuples at time $\tau + 1$. 119
Figure 5.6 A comparison between execution times of TANE with respect to our proposal by varying the number of tuples. 120
Figure 5.7 An example of *RegexHashMap* data structure. 125
Figure 5.8 An example of RegEx creation for *Cars* dataset. 126
Figure 5.9 Validation times for each dataset. 132
Figure 5.10 Validation times of REXY and Incremental-FD. 132
Figure 5.11 Memory consumption of REXY and Incremental-FD. 133
Figure 5.12 Binary vector representation of the FD $ADE \rightarrow BC$. 133
Figure 5.13 Overview of the incremental discovery strategy of COD3. 135
Figure 5.14 An example of lattice for *Iris* dataset. 137
Figure 5.15 The COD3 pipeline. 138
Figure 5.16 An example of validation graph. 140
Figure 5.17 Performances of COD3 over real-world dataset. 155
Figure 5.18 Number of validations for each case defined in Algorithm 12. 156
Figure 5.19 Time performances and memory load of COD3 by considering the variation of FDs at any time. 158
Figure 5.20 Number of FDs discovered by COD3 considering five time intervals. 160

Figure 5.21 An overview of BIRD. 165
Figure 5.22 An overview of the proposed discovery process by considering $\varepsilon = 0.1$. 166
Figure 5.23 Binary representation of candidate RFD$_e$s. 169
Figure 5.24 The compressed *linked map* related to the Example in Figure 5.22. 170
Figure 5.25 Time performances of TANE, BIRD, and BIRD^p. 183
Figure 5.26 Time performances with $P_{\tau+1} = 60\%$ and $g3$-error threshold in the range $[0.1, 1]$. 184
Figure 5.27 Time performances of BIRD with $g3$-error threshold $\varepsilon = 0.2$ and $P_{\tau+1}$ in the range $[10\%, 90\%]$. 186
Figure 5.28 Scale-up performances of BIRD^p. 187
Figure 6.1 The system architecture of DEVICE. 195
Figure 6.2 The visual interface of DEVICE. 197
Figure 6.3 DEVICE gadgets to interact with the lattice graph. 199
Figure 6.4 The visual interface of DEVICE after filtering out the attribute A. 200
Figure 6.4 Monitoring COD3 and REDEVO algorithm during its executions. 205
Figure 6.5 Resulting FDs from the executions of COD3 on real streams. 207
Figure 6.6 Monitoring COD3 executions on real data streams. 208
Figure 6.7 STRADYVAR architecture. 211
Figure 6.8 Real-time monitoring interface. 213
Figure 6.9 Interacting with the dependency table. 214
Figure 6.10 Comparing discovery results between two different executions. 216
Figure 6.11 Analyzing the correlation among attributes according to holding RFDs. 218
Figure 6.12 Overview of the Playground visual editor. 219
Figure 6.13 An example of the Statistics block usage. 221
Figure 6.14 Distribution of user answers to quantitative questions. 224
Figure 6.15 The MySQL Workbench SQL Editor. 225

Figure 6.16 The INDITIO visual interface. 226
Figure 6.17 Visualization of UCCs. 227
Figure 6.18 Validation statistics provided by INDITIO. 228
Figure 6.19 Violation details of INDITIO. 231
Figure 6.20 Statistics concerning involved participants. 233
Figure 6.21 Distributing scores achieved by participants for each analyzed scenario (with and without INDITIO). 234
Figure 6.22 Comparative boxplots showing distribution of user answers to the quantitative questionnaire. 235
Figure 6.23 Distributing participant answers to the quantitative questions in the final questionnaire. 236

LIST OF TABLES

Table 2.1 A snippet of Michelin-starred restaurants dataset. 16
Table 3.1 Results obtained from the queries used in the experimental evaluation. 52
Table 3.2 Details of the topics and seed URLs used for the comparative evaluation on e-procurement domain. 56
Table 3.3 Results of the comparative evaluation on general topics. 58
Table 4.1 Fitness values of the candidate RFDs considered in Example 2. 77
Table 4.2 Characteristics of the considered real-world datasets. 88
Table 5.1 An example of a relation instance updated at time $\tau + 1$. 114
Table 5.2 Characteristics of the used datasets and discovery results. 118

Table 5.3 A comparison between execution times of TANE with respect to our proposal by varying the number of attributes. 120
Table 5.4 Snippet of the *Cars* dataset to illustrate validation and discovery strategies of REXY algorithm. 124
Table 5.5 Characteristics of the considered real-world datasets and REXY performances on them. 131
Table 5.6 Snippet of *iris* to illustrate the discovery strategy. 136
Table 5.7 The path matrix after the insertion of the tuple reported in *Example* 2. 144
Table 5.8 Snippet of the dataset *iris* for some candidate FDS. 145
Table 5.9 Characteristics of the considered real-world datasets. 154
Table 5.10 Summarized results obtained by COD3 across different execution sessions on real streams. 163
Table 5.11 Snippet of the *echocardiogram* dataset. 167
Table 5.12 Statistics of the considered public datasets [12]. 181
Table 6.1 Questions proposed to participants. 222
Table 6.2 Questions proposed to participants. 232

1 INTRODUCTION

If we just have a bunch of datasets in a repository,
it is unlikely anyone will ever be able to find,
let alone reuse, any of this data. With adequate metadata,
there is some hope, but even so, challenges will remain . . .

— **Agrawal, Divyakant, et al.** [7]

In the current era, there is the possibility to exploit a huge quantity of data to enhance data analysis processes. Both industry and research communities have manifested a tremendous interest in methodologies capable of treating raw data and extracting information and correlations from them. This need has been further manifested in the last few years with the spread of the Internet of Things (IoT), in which every object has taken on its own identity in the digital world. For this reason, the amount of available data has significantly increased, yielding data analysis processes involving data coming from objects ever closer to people. This has triggered several new research activities, among which data profiling tasks acquired a fundamental role.

Data profiling represents an important set of activities that have been conducted at least once by any IT professional and researcher. Every data scientist has implicitly performed data profiling tasks for his/her research activities, such as when using spreadsheets, database tables, and XML files for sorting, writing structured queries, or searching keywords in collections of data. Data profiling activities have found room in different contexts, such as sentiment analysis, anomaly detection, and Machine Learning (ML for short). In fact, several studies have recently highlighted the importance of these activities to support machine learning processes for improving the effectiveness of ML models [4, 99, 106], performing advanced feature selection activities [152, 159], and estimating the feasibility of a ML task on a given dataset [103].

Data profiling activities base their effectiveness on the adoption of metadata extracted from the data, which represent properties that should be valid on data. There are several types of metadata. Simpler metadata are per-column statistics, such as the number of null and distinct values in a column, its data type, or the most frequent patterns of its data values [2]. However, there exist several more complex metadata that might involve multiple columns of a dataset or multiple datasets. Among them, Functional Dependencies (FDs) play a fundamental role, since they capture important semantic relationships among data, which can be exploited for several purposes, such as data cleaning [45, 60], query rewriting [34, 102, 130, 131], query relaxation [116], record matching [61], data preparation [44, 137, 151], feature engineering [68], and so forth. However, sometimes the definition of FD is too rigid to be used in several practical settings. For this reason, the canonical definition of FD has been extended in order to introduce new approximations, yielding the definition of Relaxed Functional Dependency (RFD) [32]. The latter introduces two main relaxation criteria into the canonical FD definition, such as the possibility to have approximate methods (similarity operators, order relations, and so forth) to compare attribute values (relaxation on the *attribute comparison*), or the possibility to have FDs holding on a subset of tuples (relaxation on the *extent*). RFDs are particularly useful in contexts in which the presence of outliers in the data should be tolerated, or in cases in which there are few valid FDs. In fact, RFDs could intercept correlations among data that cannot be caught by canonical FDs, allowing to consider similarity thresholds to compare data and admitting some exceptions in their validation. FDs were originally specified at database design time, as properties of a schema that should hold on every instance of it. However, with the advent of Big Data it has arisen the necessity to develop techniques to automatically detect FDs and RFDs from data instances, aiming to reduce the design effort and monitor their evolution in several application domains. However, the number of FDs and RFDs holding on a given database instance could be exponential with respect to the number of its columns. This makes the discovery problem an extremely complex one. For this reason, most of the discovery algorithms described in the literature provide solutions to reduce the search space

complexity by exploiting theoretical properties of FDS and RFDS. Although several existing discovery algorithms achieved good performances [126], most of them are not suitable in dynamic scenarios, in which data are frequently updated (i.e., inserted and/or removed). This entailed the definition of new incremental FD and RFD discovery algorithms, capable of updating the set of metadata holding on a dataset upon data update operations, without having to re-execute the discovery process from scratch on the entire dataset. Recently, the advent of Internet of Things (IoT) and the consequent spreading of sensors capable of continuously producing data, has triggered new challenges for data profiling tasks and discovery algorithms. While for incremental data profiling algorithms we consider periodic updates, in continuous data profiling a further challenge concerns the definition of algorithms and methodologies for continuously handling profiling metadata whenever data are created or updated. To this end, this thesis presents several new data profiling algorithms and tools for discovering and analyzing metadata in both static and dynamic scenarios.

The whole thesis is composed of four main parts. The first part provides an overview of data profiling tasks and applications, introducing preliminaries and basic notations concerning some of the most relevant data profiling metadata, such as Unique Column Combinations (UCCS), Inclusion Dependencies (INDS), Functional Dependencies (FDS) and their extension Relaxed Functional Dependencies (RFDS). Then, it focuses the discussion on the problem of profiling unstructured data in real-world applications, by introducing CAIMANS [17], an intelligent tool to support organizations in the focused analysis of artifacts of interest from the web (e.g., calls for tender, BIMs, equipment, policies, market trends, and so on). CAIMANS extracts metadata from unstructured web data sources, aiming to derive a focused crawler enabling company analysts to make better strategic decisions for improving the productivity and competitiveness of their company.

The second part of the thesis describes a new algorithm for discovering hybrid RFDS (i.e., RFDS relaxing on both the *extent* and *attribute comparison*) from data, based on a heuristic search strategy. In particular, a new genetic algorithm named REDEVO is presented, which is inspired by the

process of natural selection belonging to the larger class of Evolutionary Algorithms (EAs). The algorithm exploits natural evolution operations of species, such as natural selection, crossover, and mutation, to perform the discovery step and evaluate candidate RFDs.

The third part of the thesis shows four new incremental discovery algorithms, conceived for discovering FDs and RFDs in dynamic scenarios. In particular, the first incremental algorithm for discovering FDs is named INCREMENTAL-FD [27]. It relies on a lattice to perform a level-by-level generation of candidate FDs to be successively validated through the refinement property [85]. Such a strategy permits the adoption of several pruning rules, in order to reduce the search space of the discovery process. The second incremental algorithm for discovering FDs is named REXY (RegeX-based incremental discoverY) [28, 29]. It introduces a new validation method exploiting regular expressions (RegExs) to extract the subset of data affecting discovery results, and it employs a compressed data representation limiting the memory load and optimizing the discovery process. The third algorithm is named COD3, and is the first algorithm optimized to continuously discover FDs from data streams. It adopts a new data structure, named Validation Graph, to efficiently handle the validation process and to provide a light representation to store data. The last proposal is an incremental discovery algorithm for RFDs relaxing on the extent named BIRD (Bit-vector based Incremental RFDs Discoverer) [25]. It performs a thorough analysis of initial candidate RFDs to reduce the number of candidates to be analyzed, and it adopts a new discovery process to explore and validate multiple RFDs at the same time.

Finally, the last part of this thesis is devoted to the description of tools for visualizing profiling metadata. In particular, we present three new visualization tools to monitor the results of incremental discovery algorithms and analyze the resulting metadata after running them on different datasets. The first tool is DEVICE [26], and is used for continuously monitoring resulting RFDs during the execution of discovery processes. In particular, it permits to analyze the evolution of results during the execution of discovery algorithms through a lattice representation of the search space. The second tool is STRADYVAR [22], a tool for visualizing RFDs discovered from a data stream. It permits to explore the results

of different types of RFDs and it uses quantitative measures to monitor the evolution of discovery results. Moreover, STRADYVAR enables the comparison among RFDs discovered across several execution sessions, also proving visual manipulation operators to dynamically compose and filter results. The third tool is named INDITIO [30]. It has been implemented within the MySQL Workbench client, and it is able to intercept queries and validate metadata before the execution of insertion operations. Its integration as MySQL Workbench plugin permits to verify in real-time whether the data to be inserted into a database instance will produce some violations on specific metadata, such as unique column combinations (UCCs) and/or functional dependencies (FDs).

Thesis Outline.

This thesis is structured in the following four main Parts:

- **Part I** *Context & Background*: concerning the state of the art and the open challenges of the data profiling research area. In particular, Chapter 2 first provides an overview of the research area by discussing its application contexts and open challenges. Then, it focuses on the definition of Unique Column Combination (UCC), Functional Dependency (FD), their extension Relaxed Functional Dependency (RFD), and Inclusion Dependency (IND). Chapter 3 describes a real-world scenario of metadata extraction, focusing the discussion on a new tool named CAIMANS, for extracting artifacts from unstructured web sources, based on an article published in [17].
- **Part II** *Discovery Algorithms for Data Profiling*: concerning the problem of discovering FDs and RFDs from data in static and dynamic scenarios. In particular, Chapter 4 motivates the necessity to infer FDs and RFDs from data and formulates the discovery problem for them. Moreover, it describes the REDEVO discovery algorithm, based on an article under review on the Information Sciences Journal, Elsevier. Chapter 5 first discusses the necessity to define incremental FD and RFD discovery algorithms, and then it describes four

discovery algorithms: INCREMENTAL-FD, REXY, and BIRD, based on the articles published in [27], [28, 29], and [25], respectively, and COD3, currently under review on IEEE Transactions on Knowledge and Data Engineering (TKDE).

- **Part III** *Tools for Visualizing Profiling Metadata*: presents three visual tools for monitoring and analyzing the results of FD and RFD discovery algorithms. More specifically, Chapter 6 describes INDITIO, DEVICE, and STRADYVAR tools, based on the articles published in [30], [26], and [22], respectively.
- **Part IV** *Conclusion*: discusses the final remarks and future directions of this research.

Part I

CONTEXT & BACKGROUND

2 DATA PROFILING

Data profiling is a set of activities and processes to determine the metadata about a given set of data [118]. It includes a large amount of metadata that can be valid on a single column/attribute, such as statistics on the number of null values and distinct values in a column, the type of data, and so forth; or on multiple columns/attributes, such as unique column combinations (UCCs), functional dependencies (FDs), relaxed functional dependencies (RFDs), and inclusion dependencies (INDs). These metadata allow data scientists to capture important semantic relationships within data, which can be exploited for several purposes, such as query optimization, data cleaning, and so forth.

In this chapter we first describe data profiling tasks and applications, then we introduce basic notations concerning several multiple columns metadata (e.g., UCC, FD, RFD, and IND). Finally, we provide an overview of the open challenges in the data profiling research area.

2.1 PROFILING TASKS AND APPLICATIONS

The amount of data and sources nowadays available has motivated IT professionals and companies to adopt techniques and tools capable of managing large sets of data, by combining innovative technologies and methodologies previously proposed in the state-of-the-art across several research areas. This has led to the consolidation of the connection between the worlds of research and industry, aiming to exploit theoretical knowledge in real-world application scenarios. Nevertheless, managing such large sets of data is an extremely complex task for both researchers and companies, which have to deal not only with the development of processes for the efficient management of data, but also with processes for solving data preparation and data quality issues. In fact, data is often

riddled with errors, inconsistencies, or duplicates, slowing down business processes. In this scenario, data profiling includes a set of tasks that can be applied to different application domains, and represents a fundamental means for identifying and solving these above mentioned problems. In particular, data profiling tasks aim to examine datasets and produce metadata of several dimensional complexity, and can be classified into single-column and multi-column [1]. Figure 2.1 shows a classification of profiling tasks. In particular, *single-column profiling* refers to the analysis of values in a single column and ranges from simple counts of values and the application of aggregation functions, to the analysis of value distributions and the discovery of patterns and data types. Among the possible metadata, there are descriptive statistics (e.g., min, max, count, and number of null values), value distributions (e.g., distinct values), domain classifications, and syntactic structures (i.e., patterns). Such metadata can be applied to many application domains, such as suggesting key candidates by considering columns with only unique values, or supporting query optimizers in database management systems (DBMS) to estimate the cost of an execution plan.

On the other hand, *multi-column profiling* refers to the set of tasks that can be applied to multiple columns for the analysis of inter-value dependencies across columns. These types of tasks generalize the profiling tasks on a single column and are mainly adopted to identify correlations between values. Among the metadata, there are correlations and association rules, clusters and outliers, and summaries and sketches. Such metadata can be applied to many application domains, such as data exploration, data analytics, machine learning, and data cleaning.

Other than the previous groups of tasks, Figure 2.1 shows a third group of tasks, namely those aiming to discover dependencies, which represent metadata describing relationships among multiple columns. As we would expect, this task might be classified as *multi-column profiling*. However, it has been assigned to a separate profiling category, since it contains a large and complex set of tasks for analyzing data, detecting dependencies, and applying them in advanced operations [118]. Among the metadata in this category there are unique column combinations (UCCs), inclusion dependencies (INDs), and functional dependencies (FDs)

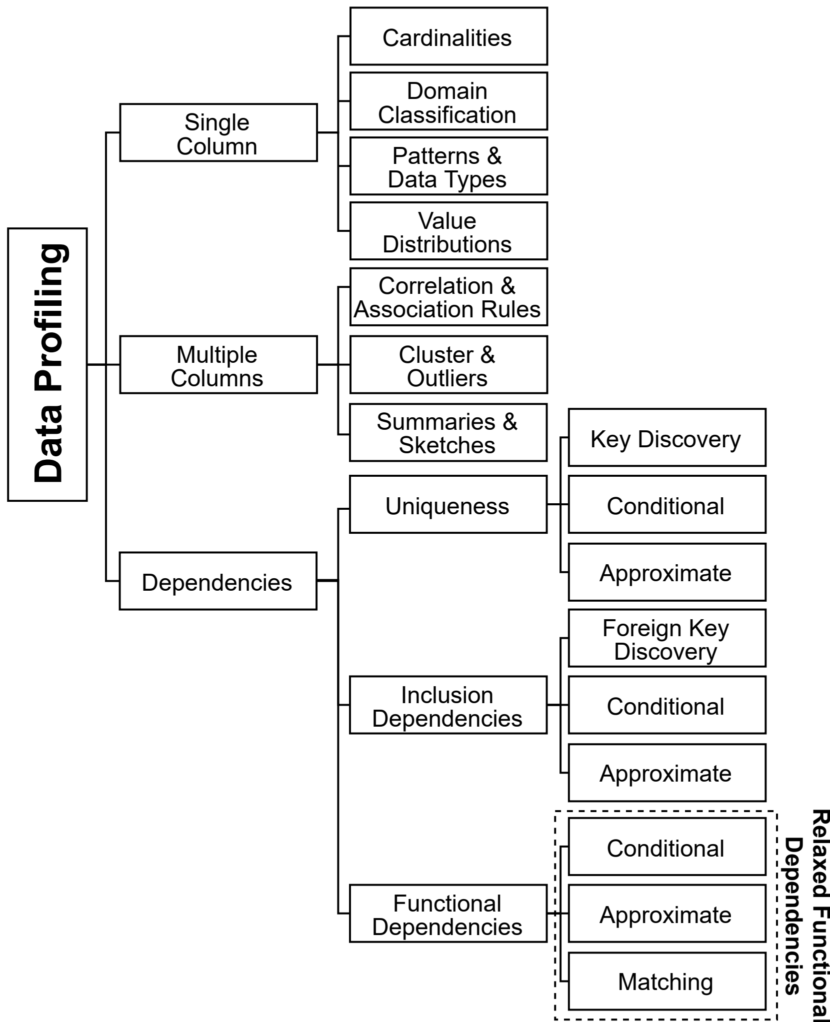


Figure 2.1: Classification of traditional data profiling tasks [1].

that will be further discussed in the next section. Such metadata can be applied to many application domains, such as the identification of suitable keys for a given table (e.g., UCC), the definition of foreign keys (e.g., IND), or the support to schema normalization tasks (e.g, FD).

In what follows, we report an overview of the main applications of data profiling metadata and tasks, in order to support advanced operations on data [1].

- *Database management*: The typical application of profiling tasks in this context is to analyze individual columns in a dataset, such as the number of unique and non-null values. In this scenario, profiling metadata can be used to extract such statistics and support optimizers for improving query performances [1, 133, 145].
- *Data cleaning*: The typical application of the profiling tasks in this context is to reveal errors, such as inconsistent formatting of values within a column, and detection of outliers and/or missing values. In this scenario, profiling metadata can be used to evaluate the general quality of a dataset and to solve quality issues [1].
- *Data exploration*: The typical application of the profiling tasks in this context is to determine the structure of datasets when they do not have a known schema or have an obsolete documentation (e.g., data files downloaded from the Web, old database dumps, etc.). In other cases, the schema might be incomplete, requiring the adoption of profiling metadata for identifying primary keys, foreign keys, or other metadata to reconstruct the structure and the history behind these data [1, 113].
- *Data integration*: The typical application of the profiling tasks in this context is to explore characteristics of the data, such as the semantics of columns and tables, and the types of data. A practical example in which data profiling tasks and metadata can support experts is schema matching, which aims to find semantically consistent correspondences between constructs of schemas under analysis [58]. More specifically, profiling metadata permits to create attribute features, such as the data type, average value length, and patterns, in order to compare schema constructs and align those attributes with the best matching ones [1, 110].
- *Database reverse engineering*: The typical application of the profiling tasks in this context is to identify relationships, attributes, and domain semantics of "bare" databases, aiming to reconstruct their

characteristics [132]. In this scenario, profiling metadata, namely "implicit constructs" [76], represent information of the dataset that are not explicitly specified by DDL statements [1].

- *Data analytics*: The typical application of the profiling tasks in this context is to prepare datasets for statistical analysis and data mining processes. In this scenario, profiling metadata can help analysts to understand the data with the aim of appropriately configuring tools for advance data analysis processes [1].

All of the above applications represent a general overview of use-cases where data profiling tasks provide significant support for advanced operations over data. Nevertheless, in the last few years, data profiling tasks have been adopted to support other application domains, such as machine learning [103] and data governance systems [51].

2.2 PROFILING METADATA

As introduced above, other than the single- and multi-column tasks, a third group of tasks has been identified, namely *dependencies*, which contains all the tasks for examining the datasets and discovering different types of dependencies. The latter represent metadata that describe relationships among attributes of a dataset, and their detection is an extremely complex problem. In fact, the difficulty of automatically extracting these dependencies from a given dataset is twofold: the need to develop efficient techniques to analyze large sets of data, and to evaluate them for identifying the most significant ones. Nevertheless, many researchers are focusing only on the first challenge, leaving out the problem of semantically interpreting the profiling results [1].

In this section, we describe preliminaries and basic notations concerning the definition of unique column combinations (UCCs), functional dependencies (FDs), a new generalized class of FDs, namely relaxed functional dependencies (RFDs) [32], and inclusion dependencies (INDs) that represent the most relevant classes of dependencies. The whole discussion is supported by several real-world examples.

2.2.1 *Unique Column Combinations (UCC)*

One of the main properties in the context of relational databases, is represented by candidate keys. They permit the definition of possible tuple identifiers of a relation instance, since no repetition in value combinations is allowed. They represent sets of columns whose projections have no duplicates, which can be identified by exploiting unique column combinations (UCCs) metadata. Before introducing the general definition for UCCs, let us recall the definition of a relational database schema.

Definition 2.2.1 (Relational database schema). A relational database schema $\mathcal{R}$ is defined as a collection of relation schemas ($R_1, \ldots, R_p$), where each R_i is defined over a set $attr(R_i)$ of attributes ($A_1, \ldots, A_M$). 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 $(t_1, \ldots, t_n)$ such that $\forall\ A_k \in attr(R_i)\ t_j[A_k] \in dom(A_k)$, where $t_j[A_k]$ denotes the projection of t_j onto A_k. A database instance r of $\mathcal{R}$ is a collection of relations ($r_1, \ldots, r_p$), where r_i is a relation instance of R_i, for $i \in [1, p]$.

Starting from this definition, we can now introduce the general definition of UCC.

Definition 2.2.2 (Unique Column Combination (UCC)). A UCC over a relation schema R is a sets of attributes $K \subseteq attr(R)$ such that given an instance r of R, for every pair of tuples (t_1, t_2) in r then $t_1[K] \neq t_2[K]$.

An important property of unique column combinations concerns their minimality.

Definition 2.2.3 (UCC Minimality). A UCC K over a relation schema R is minimal if and only if $\forall K' \subset K : (\exists t_1, t_2 \in r : (t_1[K'] = t_2[K']) \wedge (t_1 \neq t_2))$

Let us consider the relation instance r shown in Table 2.1. The unique column combination (UCC) {Latitude, Longitude} holding on r specifies that there are no restaurants located in the same geographical position for the tuples of the Michelin-starred restaurants dataset. Similarly, the UCC

{Name} holding on r, indicates that there are no two starred restaurants with the same name.

The UCCs are widely adopted in several areas of data management, such as anomaly detection, data integration, duplicate detection, and query optimization [80]. To this end, the discovery of UCCs from data represents one of the most important data profiling tasks. Although this operation seems to be a simple task even in large datasets, it is necessary to consider that the number of possible candidate UCCs is exponential in the number of attributes M of a dataset, i.e., $2^M - 1$. For instance, let us consider a dataset with 85 attributes, a naive approach must consider $2^{85} - 1$ column combinations to find all UCCs holding on the dataset.

This operation is infeasible in practice, especially when we consider datasets with hundreds of attributes. In fact, it has been demonstrated that the discovery of all unique and non-unique column combinations in a given dataset is an NP-hard problem [80]. Nonetheless, industries are making little investiments in this sector, since they rely on professional tools, which search for UCCs with few columns and/or limit the validation process with user-specified candidate UCCs. However, this can lead to a significant loss of information provided by the complete set of holding UCCs. For this reason, researchers are continuing to investigate effective solutions for discovering the entire set of UCCs on both small and large datasets.

2.2.2 *Functional Dependencies (FD)*

The concept of functional dependency has been first introduced in 1970 with the theory of normalization of the relational databases [46]. Among several other scopes, FDs were used to evaluate whether a relation is normalized according to specific normal forms [47]. However, although FDs are traditionally viewed as properties of the database schema to be specified at design time, in several modern application domains there is the need to discover them from data, especially in the Big Data context [32]. Before discussing the problem of discovering FDs, let us first recall the definition of the canonical FD.

Restaurants									
	Name	**Latitude**	**Longitude**	**City**	**Region**	**ZipCode**	**Stars**	**Cuisine**	**IDChef**
t_1	Acadia	41.859	-87.625	Chicago	Chicago	60616	2	Contemporary	Chef2
t_2	Acquerello	37.791	-122.421	San Francisco	California	94109	2	Italian	Chef1
t_3	Alinea	41.913	-87.647	Chicago	Chicago	60614	3	Contemporary	Chef2
t_4	Amador	48.254	16.359	Wien	Austria	1190	3	Creative	Chef5
t_5	Aquavit	40.760	-73.972	New York	New York C.	10022	2	Scandinavian	Chef6
t_6	Aska	40.760	-73.966	New York	New York C.	10049	2	Scandinavian	Chef3
t_7	Atelier C.	37.798	-122.435	San Francisco	California	94123	3	Contemporary	Chef4
t_8	Atera	40.716	-74.005	New York	New York C.	10013	2	Contemporary	Chef5
t_9	Benu	37.785	-122.398	San Francisco	California	94105	3	Asian	Chef7
t_{10}	Blanca	40.704	-73.933	New York	New York C.	10013	2	Contemporary	Chef9
t_{11}	Californios	37.755	-122.417	San Francisco	California	94110	2	Mexican	Chef11
t_{12}	Campton P.	37.789	-122.406	San Francisco	California	94108	2	Indian	Chef10
t_{13}	Coi	37.798	-122.403	San Francisco	California	94133	2	Contemporary	Chef8
t_{14}	Commis	37.824	-122.255	San Francisco	California	94601	2	Contemporary	Chef1
t_{15}	D.O.M.	-23.566	-46.667	Sao Paulo	Sao Paulo	01411	2	Creative	Chef3
t_{16}	Daniel	40.766	-73.967	New York	New York C.	10065	2	French	Chef9

Chef					
	IDChef	**Firstname**	**LastName**	**Place of Birth**	**State**
t_1	Chef1	Suzette	Gresham	San Carlos	California
t_2	Chef2	Ryan	McCaskey	Saigon	Vietnam
t_3	Chef3	Fredrik	Berselius	Stockholm	Sweden
t_4	Chef4	Dominique	Crenn	Brittany	France
t_5	Chef5	Juan	Amador	Waiblingen	Spain
t_6	Chef6	Emma	Bengtsson	Falkenberg	Sweden
t_7	Chef7	Corey	Lee	Seoul	South Korea
t_8	Chef8	Daniel	Patterson	Lynn	Massachusetts
t_9	Chef9	Carlo	Mirarchi	Jamaica	Queens
t_{10}	Chef10	Srijith	Gopinathan	Kerala	South India
t_{11}	Chef11	David	Yoshimura	Houston	Texas

Table 2.1: A snippet of Michelin-starred restaurants dataset.

Definition 2.2.4 (Functional Dependency (FD)**).** Given a relational database schema $\mathcal{R}$, defined over a set of attributes $attr(\mathcal{R})$, derived as the union

of attributes from relation schemas R composing $\mathcal{R}$, assuming that they all have unique names. For each attribute $A \in attr(R)$, its domain is denoted by $dom(A)$. Moreover, given an instance r of R and a tuple $t \in r$, we use $t[A]$ to denote the projection of t onto A; similarly, for a set X of attributes in $attr(R)$, $t[X]$ denotes the projection of t onto X. An FD over R is a statement $X \rightarrow Y$ (X implies Y) with $X, Y \subseteq attr(R)$, such that, given an instance r over 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 are also named Left-Hand-Side (LHS) and Right-Hand-Side (RHS) of the FD, respectively. In particular, given $Z = \{A_1, \ldots, A_k\}$, $t[Z] \in dom(A_1) \times dom(A_2) \times \cdots \times dom(A_k)$ denotes the projection of t_i onto Z, i.e., the combination of values defined by t over the attributes $\{A_1, \ldots, A_k\}$, also denoted with $\Pi_{A_1,\ldots,A_k}(t)$ or $\Pi_Z(t)$.

Definition 2.2.5 (FD **Minimality).** Given two sets of attributes X and Y with $X, Y \subseteq attr(R)$, an FD $X \rightarrow Y$ over a relation schema R is minimal if and only if $\forall X' \subset X, \exists\ (t_1, t_2)$ on an instance r of $R : (t_1[X'] = t_2[X']) \wedge (t_1[Y] \neq t_2[Y])$ with $(t_1 \neq t_2)$

Let us consider the example relation instance r shown in Table 2.1. A functional dependency φ : City $\rightarrow$ Region holding on r specifies that for any two tuples of the Michelin-starred restaurants dataset, if they are located in the same city, then their region values must be equal. For instance, tuples t_2, t_7, t_{11}, t_{12}, t_{13}, and t_{14} in Table 2.1, having the same value for city, i.e., "*San Francisco*", also have equal values for region, i.e., "*California*". Similarly, tuples t_5, t_6, t_8, t_{10}, and t_{16} having equal values for city, i.e., "*New York C.*", also having equal values for region, i.e., "*New York*". We can notice that this property is also valid for all other tuple pairs sharing the same value for the attribute city.

2.2.3 *Relaxed Functional Dependencies (RFD)*

As the relational data model has evolved in different directions, also the theory behind functional dependencies underwent several extensions to enable the definition of new approximate relationships to be adopted in new application scenarios. As described in [150], the strict equality

criteria imposed by the canonical definition of FDs limits the detection of semantic relationships in the data.

For example, for the Michelin-starred restaurants dataset in Table 2.1, we can intuitively affirm there should be a certain relationship between *Region*, and geographic coordinates, such as *Latitude* and *Longitude*. However, as we expect, a region could cover a very large portion of territory that identifies different areas with different geographic coordinates.

Another example could concern large systems that daily use geographic information for the prediction of atmospheric or natural phenomena by studying historical data. In this case, RFDs could extract anomalous patterns in the data for the immediate identification of unexpected atmospheric events in various countries or cities.

For this reason, new dependencies have been introduced to cope with approximate comparisons and have been named Relaxed Functional Dependencies (RFDs), i.e., dependencies that relax one or more constraints of the canonical FD. In particular, there are over 30 different types of FDs under the definition of RFDs, and each of them represents a means for overcoming the limits imposed by the FDs [32]. Figure 2.2 shows an overview of the relaxation criteria used to categorize the different types of RFD. The first criterion is the tuple comparison method used on the Left-Hand-Side (LHS) and Right-Hand-Side (RHS) of the RFD, and we will refer to it by using the term *attribute comparison*. Thus, RFDs applying only this relaxation criterion are named RFD relaxing on the attribute comparison (RFD_c). The second criterion is the *extent*, which groups all the RFDs that permit a dependency to hold on a subset of tuples. Thus, RFDs applying only this relaxation criterion are named RFD relaxing on the extent (RFD_e). Both criteria have been further detailed in order to provide parameters and functions to classify the different types of RFDs. The attribute comparison criterion has been divided in two categories, approximate match and ordering criteria, respectively. In particular, the approximate match category is used to measure the similarity and/or the diversity of attribute values, whereas the ordering criteria compares the attributes based on a given order relation. Similarly to the attribute comparison, also the extent criterion has been divided into two categories, namely coverage measure and condition, respectively. The

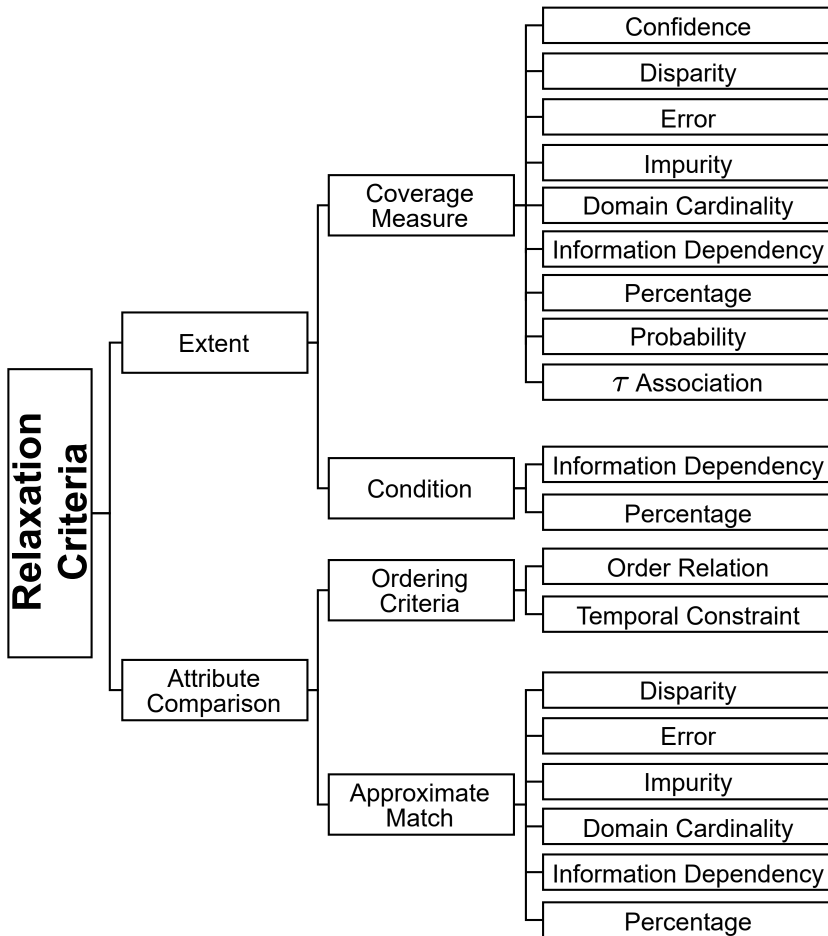


Figure 2.2: Relaxation criteria for RFDs [32].

former includes all categories of RFDs in which a coverage measure should be applied to quantify the degree of satisfiability of the RFD, whereas in the second category a condition should be specified to identify the subset of tuples on which a dependency holds.

Before introducing the general formalization for the RFDs, we recall some definitions that will allow us to describe the general semantics of RFDs and establish relationships among them [32].

Definition 2.2.6 (Constraint). A constraint ϕ is a predicate evaluating whether the similarity/distance, or the order relation, between two values of an attribute A falls within a predefined interval.

Thus, a constraint depends on a similarity/distance function, or an order relation, defined on an attribute domain, plus one or more comparison operators with associated threshold values defining the feasible intervals of values.

Definition 2.2.7 (Set of constraints). Given a set of attributes $X = \{A_1, \ldots, A_k\}$, a set of constraints $\Phi = \{\phi_1, \ldots, \phi_k\}$ on them represents a collection of constraints that are applied to $\{A_1, \ldots, A_k\}$, respectively.

Definition 2.2.8 (Coverage measure). Given two sets of attributes X and Y, a coverage measure Ψ, $\Psi : dom(X) \times dom(Y) \rightarrow \mathbb{R}^+$, quantifies the amount of tuple pairs in r satisfying the dependency.

Several coverage measures can be used to define the satisfiability degree of an RFD, but usually they return a value normalized on the total number of tuples n, so producing a value $v \in [0, 1]$. Among the most commonly used coverage measures, there are the confidence, the *g3-error*, and the probability.

For instance, the *g3-error* calculates the minimum fraction of tuples that must be removed from a relation instance r in order to make a dependency valid (see Formula 2.1).

$$g_3(X \rightarrow A, r) = 1 - \frac{max\{|s| \mid s \subseteq r \models X \rightarrow A\}}{|r|} \tag{2.1}$$

As discussed above, an RFD is a functional dependency that involves sets of constraints to evaluate the distance or similarity between attribute values (i.e., RFDs relaxing on the tuple comparison), and/or uses a coverage measure to indicate the minimum number or percentage of tuples on

which the RFD must hold (i.e., RFDs relaxing on the extent). Most of the RFDs defined in the literature relax on one of these two dimensions and only some of them relax on both dimensions (i.e., hybrid RFDs). In what follows, we recall a formal definition of RFD extracted from [37] which covers all these cases.

Definition 2.2.9 (Relaxed functional dependency (RFD)). Given a relation schema $R = (A_1, \dots, A_m)$, an RFD φ on R is denoted by

$$X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2} \tag{2.2}$$

where

- $X = X_1, \dots, X_h$ and $Y = Y_1, \dots, Y_k$, with $X, Y \subseteq attr(R)$ and $X \cap Y = \emptyset$;
- $\Phi_1 = \bigwedge_{X_i \in X} \phi_i[X_i]$ ($\Phi_2 = \bigwedge_{Y_j \in Y} \phi_j[Y_j]$, resp.), where ϕ_i (ϕ_j, resp.) is a conjunction of predicates on X_i (Y_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 $t_1[X_i]$ and $t_2[X_i]$ ($t_1[Y_j]$ and $t_2[Y_j]$, resp.) satisfy the set of 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.
- 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 $r \subseteq dom(R)$ 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 $\Psi \leq \varepsilon$.

For RFDs relaxing on the tuple comparison only, when no pair of tuples yields an RFD violation, the expression $\Psi(X, Y) = 0$ is omitted from the RFD expression. Moreover, when the equality constraint is used as a tuple comparison method, the set of constraints Φ is also omitted from the expression. Thus, the canonical FD can also be written in terms of (2.2) as: $X \rightarrow Y$.

Let us consider the example relation instance r shown in Table 2.1, it is likely to have the same region and zip code for restaurants in the

same city. Thus, an FD $\mathsf{City} \rightarrow \mathsf{ZipCode}, \mathsf{Region}$ might hold. However, these attributes might have been stored using different abbreviations and/or there may be several zip codes in a city that identify different areas or districts, hence the following RFD might hold:

$$\mathsf{City}_{\phi_1} \rightarrow \mathsf{ZipCode}_{\phi_2}, \mathsf{Region}_{\phi_3}$$

where ϕ_1, ϕ_2, and ϕ_3 are constraints using a string similarity function. Moreover, since cities might change zip code after the updating of the territorial topologies, or there might be multiple cities with the same name but located in different regions, the previous RFD should tolerate possible exceptions. This can be modeled by introducing a coverage measure into the RFD:

$$\mathsf{City}_{\phi_1} \xrightarrow{\psi(City,ZipCode,Region)\leq 0.2} \mathsf{ZipCode}_{\phi_2}, \mathsf{Region}_{\phi_3}$$

2.2.4 *Inclusion Dependencies (IND)*

In the context of relational databases, other than the candidate keys, one of the main properties is represented by foreign keys. The latter permit to define the relationships between tables of a relational database. In particular, they allow to maintain referential integrity across tables of a database, supporting researchers and IT professionals in preventing errors, and improving the performance of any operation that extracts data from tables that are linked by foreign keys. This relationship can be identified by exploiting the concept of inclusion dependency (IND) [158].

Definition 2.2.10 (Inclusion Dependency (IND)). Let $\mathcal{R} = R_1, R_2, \ldots, R_p$ be a database schema, and $r = r_1, r_2, \ldots, r_p$ be the database instance of $\mathcal{R}$, where each r_j corresponds to the relation instance of R_j, with $j \in [1, p]$. Let $\Pi_X(r_j)$ be the projection of r_j on attribute X from R_j, and $t[X]$ the restriction of tuple t to X so that $\Pi_X(r_j) = \{t[X] \text{ s.t. } t \in r_j\}$. The IND $R_i[X] \subseteq R_j[Y]$ between the two attribute sets X and Y is satisfied by an instance r over $dom(\mathcal{R})$ if and only if $\Pi_X(r_i) \subseteq \Pi_Y(r_j)$.

The attributes on X represent the Left-Hand-Side (LHS) of an IND and are defined as *dependent* attributes. On the other hand, the attributes on Y represent the right-hand side (RHS) of an IND and are defined as the *referenced* attributes.

Definition 2.2.11 (IND **Maximality).** Given two relations R_i and R_j the IND $R_i[X] \subseteq R_j[Y]$ between the two attribute sets X and Y is maximal if $R_i[XA] \subseteq R_j[YB]$ is invalid for any $A \in attr(R_i)$ and $B \in attr(R_j)$.

The set of all maximal INDs is a complete set of INDs, because all non-maximal INDs can be derived from it.

Let us consider the example instances r shown in Table 2.1. An inclusion dependency (IND) Restaurants.IDChef $\rightarrow$ Chef.IDChef over r specifies that the set of values appearing in the attribute IDChef of the relation Restaurants must be is a subset of the values appearing in the attribute IDChef of the relation Chef.

INDs are widely adopted in several areas of data management, such as anomaly detection, schema (re-)design, query optimization, or data integration. When such dependencies are not defined at design time, scalable and efficient algorithms allow to discover and validate them from a given database instance [143]. In particular, the validation process of INDs requires checking if each record of the projection on X is contained in the projection on Y. Nevertheless, the discovery of all non-trivial INDs (i.e., candidates in the form $R_i[X] \subseteq R_j[X]$) on a database schema with M attributes is an extremely complex problem, since the candidate space grows exponentially when searching INDs by considering many relations with high cardinality [127]. To this end, several algorithms for discovering INDs have been defined [13, 53, 94, 158], but industries continue to invest in this area with the aim to develop effective solutions for discovering INDs and applying them in commercial solutions.

2.3 OPEN CHALLENGES

In the last few years, research in data profiling has proliferated, due to the necessity to exploit profiling metadata to support several activities related to data science. Figure 2.3 shows an overview of the different

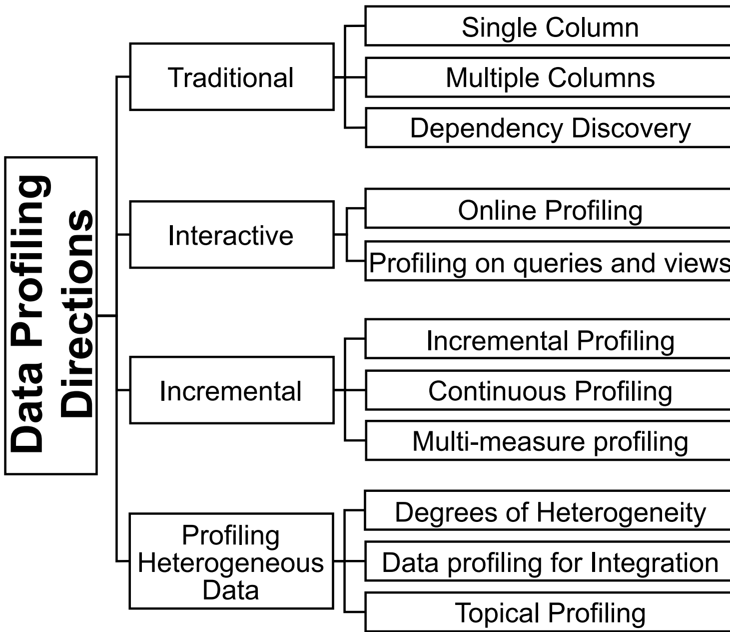


Figure 2.3: New directions of data profiling [118].

types of data profiling directions, ranging from traditional profiling to the most recent profiling scenarios.

Traditional data profiling represents the classical form of data profiling, which includes single- and multiple- column metadata and the tasks for discovering them. As discussed in the previous section, these tasks are generally applied to single relations or complete databases. However, the definition of new technologies that enable users to interact with the data and update its values, and the spreading of data sources that continuously produce data, have required the adaptation of traditional data profiling tasks to new directions. Among these, incremental data profiling represents all the tasks capable of efficiently handling the incremental update of metadata, without having to re-execute profiling tasks upon updated versions of the datasets. Another important challenge concerns the spreading of new data sources, since data profiling tasks

generally consider homogeneous datasets or single relations, but there are many scenarios in which data are collected and analyzed even though they originate from different and heterogeneous sources. To this end, data profiling tasks are evolving towards the profiling of heterogeneous datasets.

All the new directions introduced above only concern activities and tasks directly performed on the datasets. For this reason, research has hardly recognized that data profiling is an inherently user-oriented activity. However, it is important to notice that metadata is consumed and analyzed directly from users before adopting them for advanced operations. Thus, the necessity to further involve users in data profiling activities has led to the definition of visual metaphors and tools to enable users to interact with data profiling tasks. This new direction of data profiling is named interactive data profiling.

In general, data profiling tasks should manage the above-described challenges by also considering the following issues: *i*) the problem of discovering profiling metadata is computationally complex, especially when considering large datasets; *ii*) the validation of profiling metadata requires the verification of complex constraints on all columns and/or combinations of columns in a dataset; *iii*) the complexity of the problem and the size of datasets could lead to memory load issues during the execution of data profiling tasks [118].

In the following sections, we discuss the directions for data profiling and give a brief overview of well-known approaches.

2.3.1 *Profiling Heterogeneous Data*

While researchers are deeply investigating data profiling techniques, also business leaders are taking in consideration investments in technologies to effectively profile data. As we know, data can be produced from several huge data sources, and generally these are stored in homogeneous databases, in which data with the same format are considered. However, there might be cases in which data are produced from heterogeneous sources, and this still necessary to collect and analyze them together. For

these reasons, data profiling tasks should be adapted to work with data regardless of their format. In this scenario, data profiling tasks focus their activities on the definition of approaches for integrating these data, and for computing their degrees of heterogeneity [118]. In particular, the heterogeneity of data can be defined at many levels and degrees of severity, divided into syntactic, structural, and semantic heterogeneity [123]. Traditional data profiling tasks enable the evaluation of the syntactic heterogeneity of the data, by means of single- and multiple- column metadata. In fact, through metadata we can identify inconsistencies in data formatting and define approaches to solve them. Instead, structural heterogeneity can be identified in the form of unmatched patterns, i.e., they are represented and/or stored with differently structured information. However, this type of heterogeneity can be partially addressed by traditional data profiling through metadata such as keys and foreign keys, and it requires more advanced approaches exploiting them. Concerning semantic heterogeneity, it can be interpreted as a set of data in which there is a low correlation of meanings. In data profiling it can be addressed as the problem of discovering semantic overlaps of the data and their domains [118].

Although the identification and the resolution of the previous heterogeneity issues is an important challenge for data profiling tasks, another challenge addressed by data profiling is the integration of heterogeneous data. To this end, it is possible to exploit structural profiling for extracting information about the schema of the data, and semantic profiling for interpreting and extracting information about them. Moreover, metadata can also provide additional information useful for integrating the data. For instance, inclusion dependencies can suggest ways to join two tables that are not yet related, whereas relaxed functional dependencies can support the identification of similar data among schemas. The combination of these tasks and of metadata permits to support data integration activities for identifying schematic and data overlaps, assessing the possibility of integrating data, and indicating the effort required for it.

Data profiling supports another fundamental activity for heterogeneous data: the identification of topics or domains covered by different data sources. In fact, working with a large amount of heterogeneous data

also requires checking if they all belong to the same or similar domains in order to perform proper activities on them. To this end, topical data profiling should be able to extract semantic information, topics, and/or domains from given sets of data, aiming to determine topical overlap between them.

2.3.2 *Incremental Data Profiling*

The advent of the Internet of Things (IoT) and the consequent spreading of hardware and software sources, has led to the necessity to adapt data profiling tasks towards new scenarios. In fact, traditional data profiling tasks have been originally designed to work with fixed (i.e., static) datasets, requiring to be re-executed when datasets were updated. This is not always an efficient solution, especially for data profiling tasks that are extremely complex. To this end, researchers and IT professionals paid close attention to this problem and started defining new incremental methodologies capable of extending traditional data profiling tasks, and to adapt them to incremental scenarios. In particular, incremental data profiling tasks should be able to start from the results obtained from the execution of a task on a dataset before it receives updates, and dynamically profiling metadata upon updates, without having to re-profiling the entire dataset. Starting from this strategy, two new types of tasks have been defined, incremental and continuous data profiling tasks [118].

Incremental profiling activities rely on the idea of reusing both the profiling results and the historical information of the datasets. In fact, by keeping track of changes in a dataset it is possible to increase the performances of incremental data profiling tasks, by focusing the computation only on the updated part of the dataset. For instance, in the discovery of metadata like functional dependencies or unique column combinations, by keeping track of the updated values it is possible to re-validate only a portion of the metadata on the updated dataset. With simpler metadata, such as sum, count, and equi-width histograms, the process to update the metadata is implicitly incremental and it does not require the definition of particular methodologies, since they can be associatively calculated.

However, although incremental profiling methodologies aim to improve the effectiveness of data profiling tasks, it is necessary to explore how to perform such tasks when considering more complex scenarios, such as the continuous updating of the data. In fact, the spread of the IoT has led to the necessity to adopt data profiling tasks on continuous data streams generated from data providers and sensors, such as traffic sensors, health sensors, transaction logs, activity logs, etc. Typically, these data providers send many data in extremely short time intervals, creating a continuous data stream that must be rapidly and correctly managed, without having the possibility to store all the data to analyze in memory. Thus, incremental data profiling tasks need to be extended to efficiently manage these data.

Existing data stream management systems (DSMS) tend to perform operations only on a temporal window by means of queries, or by storing an aggregate representation of the data [118]. This could be useful for simple metadata, such as count, sum, and avg, but not for more complex ones, such as functional dependencies and inclusion dependencies, in which it is necessary to store part of the data already processed to correctly validate them. Similarly, also for more advanced tasks, such as data integration and data imputation, the execution of tasks on data streams is extremely more complex, since the tasks must be performed in short time, by continuously ensuring the correctness of results. For this reason, continuous data profiling tasks aim to find a good tradeoff between accuracy and resource consumption, without affecting the correctness of the performed operations.

2.3.3 *Interactive Data Profiling*

Research on data profiling seems to consider tasks that do not directly involve users. In fact, most tasks seem to require no interaction with users, since they work in batches on the data. However, the role that users play is fundamental in data profiling tasks. In most cases, metadata is analyzed directly by the user before being used in some applications, such as schema design or data cleaning. To this end, several data profiling tasks

have focused on the definition of tools and visual metaphors that involve users in the profiling processes. In particular, online profiling aims to provide intermediate results of profiling tasks that allow users to perform their analysis even when the profiling processes are working. However, the simple connection of a graphical interface to existing algorithms is not enough. In fact, data and metadata generated by data profiling tasks could be misleading for users, even due to confusing representations. For these reasons, visual tools should provide approximate or sampling-based methods, whose results gracefully improve when more computations are performed [118], and include visual languages and metaphors ensuring the user interaction with both processes and results. For instance, for metadata discovery, one of the main problems is the possibly high number of metadata that might hold on a given dataset, which might make it difficult for a user to get insights from them. Whereas for data cleaning and data integration tasks one of the main problems is to allow users to perform operations on a large number of tables and to virtually interact with them for re-computing profiling results. Addressing all of these problems requires a considerable effort, which is even greater when considering incremental data profiling tasks. In this case, the difficulty to design visual tools for analyzing and interacting with algorithms further increases, since it is necessary to devise tools that continuously interact with processes, without affecting their performances.

3 PROFILING UNSTRUCTURED WEB DATA SOURCES

Profiling heterogeneous data still represents an open challenge for researchers and IT professionals, since it is necessary to define methodologies capable of analyzing data regardless of their format. To this end, companies and public administrations are investing in developing tools to automatically profile data and extract useful information from them, aiming to support their decision-making processes.

In this chapter, we first provide an overview of the issues of data profiling on heterogeneous data in real-world applications, and then we show an innovative tool that combines AI and data profiling techniques for crawling and analyzing unstructured artifacts from the web.

3.1 PROBLEM DESCRIPTION

Data profiling comprises a broad range of activities and tasks to efficiently analyze sets of data, and extract useful information from them. Among them, the profiling of heterogeneous data represents an open challenge for researchers and IT professionals, since it requires the definition of methodologies and tools for reviewing the quality of the data and extracting metadata from different types of data sources. In general, data can be represented in different ways, yielding three main categories of data representations:

- **Structured data** is a standardized format for providing information and classifying the content of data sources [160]. This type of data depends on the existence of a schema that determines how the data can be stored, processed, and accessed. Each field is discrete and it can be accessed separately or jointly together with other fields, making structured data extremely powerful for querying and aggregating data in the whole database. The fixed schema typically

organizes data in tables with rows and columns, by also enabling the representation of relationships among data collected in different tables. Some examples of structured data include spreadsheets and/or relational databases.

- **Semi-structured data** is a form of structured data that does not follow a formal structure of data models associated with relational databases or other forms of data models [138]. Nevertheless, it contains tags or other markers to separate semantic elements, in order to enforce hierarchies of records and fields within the data. Some examples of semi-structured data include JSON and XML.
- **Unstructured data** is a form of data that does not follow a predefined data model [114]. This type of data is typically text-heavy but it may contain dates, numbers, and facts as well. This results in irregularities and ambiguities that make it difficult to understand the representation and the meaning of data by using traditional processes. Some examples of unstructured data include documents, No-SQL databases, and web pages.

The proper management of unstructured data, and the consequent definition of tools that exploit data profiling techniques, is becoming a key business goal for companies and public administrations. In fact, the profiling of unstructured data represents a set of fundamental processes that combines business rules and analytical algorithms to discover, understand, and extract information from the data. Thus, the need for efficient data profiling processes and platforms is growing, both in companies and public administrations that produce huge volumes of data during their ordinary activities. In fact, companies manage massive amounts of data extracted from different sources that are stored within their servers. For example these data can include previously offered services, proposals, bids, and so on. However, companies rely on expert managers to manually analyse them in order to make strategic decisions. On the other hand, also public administrations suffer from similar problems, mainly due to the digitization of thousands of documents that were previously collected in paper format, and the daily publication of hundreds of new documents on the web. Similar to companies, they rely on public employees for man-

ually crawling documents of different nature. Thus, both companies and public administrations need to adopt data profiling processes with the aim to develop technologies capable of automatically analyzing such data, in order to infer new knowledge and use it to improve their efficiency, productivity, and competitiveness. To this end, in the following sections, we propose an innovative tool, named CAIMANS (Crawling Artifacts of Interest and Matching them Against eNterprise Sources), which exploits AI and data profiling techniques to support organizations in crawling and analyzing artifacts of interest from the web.

3.2 LITERATURE REVIEW

Data profiling consists of several tasks and activities to determine metadata about a given set of data (see Section 2.1). Among them, several recent proposals have focused on the definition of methodologies and tools for profiling unstructured data [51]. In [115] authors propose a tool, namely jHound, for profiling collections of JSON documents. It tackles problems regarding data quality for both own and open hosted data. Moreover, jHound gives insights on how data looks like, and points out typical pitfalls which must be solved before processing data in subsequent data cleaning steps.

Another important issue addressed by researchers and IT professionals concerns the necessity of defining methodologies for profiling Linked Open Data (LOD), also due to the availability of many RDF data sources [18]. To this end, several research communities are fostering the application of data profiling techniques on open RDF datasets, such as SwetoDBLP [8] and LinkedMDB [78]. Recently, a tool has been proposed that can infer the actual schema, gather corresponding statistics, and present a UML-based visualization for the RDF data sources like SPARQL endpoints and RDF dumps [105]. Other researches are limited to treating RDF data statistics and summaries, such as Semantic sitemaps [50], RDFStats [101], and SCOVO vocabularies [79].

Other studies focus on the development of algorithms to profile specific topics from web pages and identify relationships between them, such as

commercial trends, people, and environmental situations. For instance, in [6] authors propose a methodology to represent students using data extracted from their home page, whereas other works use data extracted from multiple pages to profile entities [134].

The advent of IoT technologies and the spread of sensors that continuously produce data has further made data profiling a key activity for data analysis. In [154] authors adopt data profiling techniques for determining records useful for the identification of unknown gas samples. In [155] authors adopt data profiling methodologies for detecting sprint intervals of runners by using data from high-resolution sensors, and for computing the ground contact times in order to evaluate sprint performances.

Other theoretical studies address the problem of extracting relationships between unstructured data [9, 32]. A recent study proposes an algorithm for discovering Temporal Graph Functional Dependencies (TGFDs) [120]. These are a class of data quality rules imposing topological, attribute dependency constraints that hold for a period of time. In [48, 49] authors define approaches for discovering Temporal Functional Dependencies (ATFDs) from clinical databases. In [77] authors propose a new approach for extracting functional dependencies in XML documents based on homomorphisms between XML data trees and schema graphs.

As discussed above, data profiling tasks can support public or business processes in different application scenarios. To this end, companies and public administrations make increasing use of these methodologies and tools, with the aim of maximizing the extraction of knowledge from data.

In the next section, we propose a new tool for extracting and profiling artifacts from the web, capable of analyzing their content and classifying those most relevant to specific search criteria.

3.3 CAIMANS: CRAWLING ARTIFACTS OF INTEREST AND MATCHING THEM AGAINST ENTERPRISE SOURCES

This section presents the tool CAIMANS, developed in cooperation with industry in the context of a project funded by the Italian Minister of Economic Development, aiming to crawl web sources. In particular,

CAIMANS aims to *i*) crawl artifacts from the web whose informative content matches specific topics of interest, trying to overcome possible linguistic ambiguities of contents written in natural language, and *ii*) match the characteristics of crawled artifacts against data and knowledge stored within enterprise local sources. To this end, we have combined several data profiling, machine learning, and natural language processing techniques by also extending and adapting some of them in order to solve some of the previously mentioned problems. In particular, we face the K-means clustering algorithm problem of converging to local minimum by relying on multiple random starting points [16], and have defined several new modules to effectively gather, manage, and process data.

The first module is a novel web crawler capable to extract and preprocess unstructured data from the web. The output of such component is formatted in a suitable way to enable further analysis against a set of query terms, so as to find the artifacts that are more pertinent to the enterprise goals and capabilities. The second module relies on enterprise data and knowledge sources. A third CAIMANS' module aims to find metadata and semantic matches between the crawled artifacts and the knowledge stored within the enterprise sources. Finally, the last module is responsible for visualizing the crawled artifacts. All the approaches underlying CAIMANS have also been conceived to work off-line, by running the analysis in batch mode and visualizing the results at the most convenient time for the user. In the following sections, we describe the methodologies behind each module of CAIMANS and prove its effectiveness by comparing its crawling component against similar crawlers, by plugging them within our system.

3.3.1 *Extraction and Profiling of artifacts from the web*

Current Web search engines require users to search for artifacts of interest by mainly entering query strings. Generally, this limits the search and does not guarantee that correct results will be immediately obtained. Human experts must carry out many manual searches in order to obtain useful results. In fact, often within the Search Engine Results Page (SERP)

there are many pages outside the search scope. To this end, CAIMANS can reduce the overall search time, by increasing the number of correct results with respect to manual search methodologies. In particular, thanks to the advanced management of search parameters and URLs to avoid, CAIMANS can guarantee a faster convergence of the search engine towards a set of artifacts of interest. For instance, in the call for tenders domain, CAIMANS enables a company to extract call for tenders from the Web and classify them according to the company's needs and past experiences. At the same time, CAIMANS can search the company's data and knowledge sources to retrieve artifacts related to past calls for tender, focusing on the proposed solutions and achieved evaluations, so as to have a starting point for writing a proposal for the current calls for tender. This will potentially reduce the effort for preparing bids, yielding cost reduction, and increasing the chances of a company to gain contracts than relying only on the manual work of human experts.

CAIMANS has been conceived as a modular platform, in which each component can be singularly used and interact with other ones by defining a proper workflow. Each component was developed independently from the others, by formally specifying its RESTful interface. Figure 3.1 shows the system architecture of CAIMANS, in which the interactions between its modules are shown. The following sections describe the characteristics and techniques used for the development of the individual components.

3.3.1.1 *Crawler Module*

This module has been engineered based on the call for tender domain, and then it has been adapted to be used in similar domains. Such a module has the duty of traversing the web and returning the artifacts pertinent to a given query, by following exploration and priority rules, and abiding by search limits. In particular, this module represents a focused crawler, which is based on the Linkage Locality criteria [95]. In other words, starting from a set of user-defined web pages that are pertinent to the searched topic, the exploration begins by retrieving all the pages related to them. This is based on the assumption that the web pages

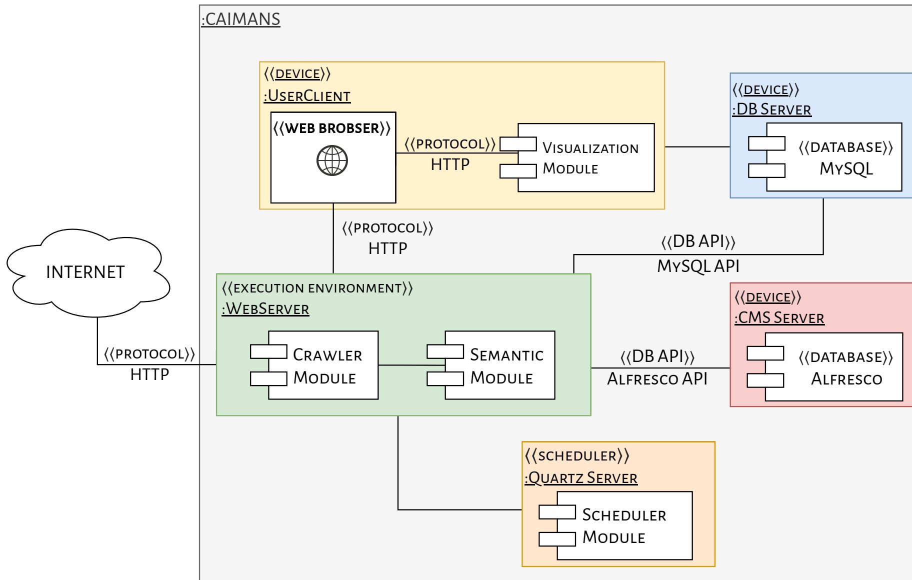


Figure 3.1: The architecture of CAIMANS.

on a given topic are more likely linked to those concerning the same topic. For example, if the crawler analyses the content of the Italian government gazette website, most linked pages contain information about calls for tender. These links are placed in the exploration queue one after the other, in order to be analyzed by different semantic levels. To specify the priority order in which URLs have to be visited during the crawling phase, a navigation queue has been defined. Furthermore, to define search limits, the crawler uses a customized URL blacklist that excludes out-of-context sub-domains from scanning. The crawler module requires the following further functional components:

- A component to download web pages from the absolute URLs, using the HTTP protocol;
- A component for extracting content and links from HTML documents;
- A component to validate the syntax and the existence of a URL;

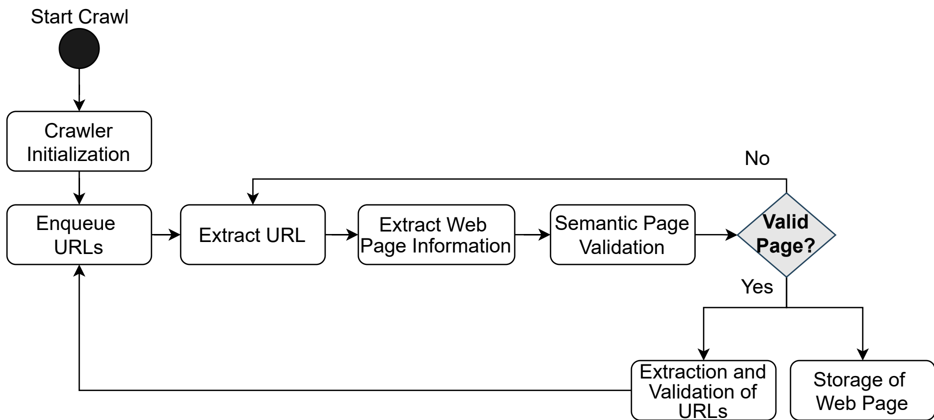


Figure 3.2: Flowchart of the Crawler module.

- A component for determining whether a URL has been encountered before;
- A component for extracting content and links from RSS Feed;
- A component to avoid the exploration of blacklisted domains/-pages.

Moreover, the whole crawler module has been realized by extending the web crawler Mercator architecture [81]. The latter enables a focused search, ensuring that all components are independent from each other and cooperate in a single system by taking input data from the previous component, processing it, and returning the output to the next component. All the components can be maintained and updated separately from each other, reducing the impact of changes in the other modules. Crawling is accomplished through a scheduled sequential process managed by the Quartz Scheduler [38], through a CRON expression[1]. The latter is a string composed of 7 space-separated fields, representing seconds, minutes, hours, day, month, weekday, and year, respectively.

Example 1. If we consider the CRON string "0 15 10 ? * MON-FRI", the schedule fires at 10:15 AM every Monday, Tuesday, Wednesday, Thursday,

1 http://www.quartz-scheduler.org

and Friday. The value "*" is used to select all values within a field, whereas the value "?" is useful when it is not necessary to specify the day.

The scheduled search job is automatically started at a given time, or based on a recurring schedule. These kinds of tasks can be easily performed on external servers or cloud machines. The scalable architecture of Mercator enables CAIMANS to parallelize the single components according to the search engine execution platform. The execution steps of the crawling module are shown in Figure 3.1.

Execution steps. The first step corresponds to the crawler initialization, consisting in the definition of the query string, the keywords, and the maximum number of pages that the crawler must explore in order to reduce the number of incorrect results and to increase the search accuracy. The next crawling phase starts when an absolute URL is extracted from the URL queue. In order to avoid that the crawler only analyses pages from the same domain, the elements in the queue are shuffled after n loops. The parameter n is an integer randomly generated in the range $K/2$ and K, where K is the maximum limit of pages to be explored. Next, the frontier component of Mercator extracts the URL, checking whether it is not contained in a blacklist. Each web page is analyzed by using an HTML analyzer, based on the XPath syntax. The latter enables fast access to the content of the HTML tags. Thus, it avoids processing a complex HTML parser. A stemming algorithm is used to process all words in the text, reducing those with the same root to a common format, by stripping the derivational and inflectional suffixes from each word. More specifically, we have employed a stemming algorithm for the Italian language [55], that allows the module to quickly analyze the text extracted from the given web page by means of Natural Language Processing (NLP) techniques. To this end, we have used basic NLP techniques, since the volume of artifacts to be processed is quite large, and using complex techniques would not allow having fast processing. Furthermore, we have extended this algorithm with a module capable of using a set of keywords to be involved in the analysis of web pages. Once the keywords

have been processed, each word and its synonyms are searched in the text of the web page to verify the correctness of the results.

Example 2. If we consider the keyword "fixtures", by using the stemming algorithm its root "fixtur-" is extracted. Starting from the singular form of the word, all of its related synonyms are searched, i.e., "doors", "windows", "shutters", and so on. Then, for each synonym, its root is extracted, i.e., "door-", "window-", "shutter-", and searched within the text.

The extraction module uses accurate regular expressions to extract the contents. For instance, in the calls for tender domain, these correspond to prices, opening date, closing date, SOA categories, and other information useful for classifying the results. Furthermore, in addition to the regular expressions already defined, it is possible to define new expressions to be involved in the search. This type of strategy ensures that each module is fully customizable during the initialization phase. Towards the end of the cycle of phases, a test is performed to verify whether the page is valid. If so, then the crawler stores the results in a database, extracts the links contained in it, by storing them in the exploration queue, and goes back to its second phase. During this process, dynamic URLs related to servers or main pages of generic portals are discarded, in order to avoid the insertion in the exploration queue of pages that are out of context. Moreover, in the validation phase the crawler checks whether a link is an absolute URL and if it does not refer to a website already visited before and inserted in the blacklist. The validation component has a single crawl method that takes in input a URL and returns a boolean value indicating whether or not the URL should be added to the queue. On the other hand, if the page is not valid, the crawler goes back to its third phase. The process stops when the crawler exceeds the time limit or the number of pages to be visited.

3.3.1.2 *CMS Module*

This module needs to process several types of artifacts. For instance, civil engineering projects include technical artifacts, describing the design

of a building or its maintenance plan, but also administrative ones, stating the financial planning and the project schedule. Such a volume of data is characterized by having multiple kinds of formats, ranging from PDF to DOCS, or a level of structure and formalization spanning from unstructured material (such as images from scanned documents) to more structured ones (such as XML rendering of architectural plans). Traditionally, such a set of materials is poorly managed by the companies, which usually maintain flat storage within the folders of their servers or employees' computers. Although such a strategy does not require a considerable computer science background (motivating its large usage) is not efficient due to its intrinsic difficulty in retrieving a specific artifact of interest within the set. The traditional approach of using relational databases can help having more effective retrievals, thanks to queries expressed in a SQL-like language. However, such a solution is viable for storing properties, but it is less effective when dealing with files. A Content Management System (CMS) [124] represents a trade-off between folder storage using the operating system and a relational database, so that the artifacts of interest are in the file system, and the CMS manages pointers to them within certain tables coupled with metadata supporting queries for their retrieval. Such pointers are transparent to the users, and the CMS guarantees their consistency concerning artifact mobility within the file system.

In our project, we have adopted the Alfresco CMS platform[2] for storing and retrieving the artifacts of interest. For instance, in the e-procurement domain, the file system can be structured by considering the different parts of any civil project, each of which can contain artifacts related to a call for tender or a bid for it. This type of strategy allows CAIMANS to quickly interact with company artifacts, aiming to characterize the search by involving information extracted from the company's knowledge base. We have built a set of RESTful web services on top of Alfresco, in order to enable the storage and retrieval of artifacts. The API used to interact with Alfresco has been the one provided within the Content Management

2 https://www.alfresco.com/

Interoperability Services (CMIS) [43] (and the Apache Chemistry[3] library for .NET), so that we can use any possible CMIS-compliant CMS and replace Alfresco based on the customer needs. Such services have been made secure by using the standard JSON Web Token (JWT) [86], for stateless authentication and authorization. The retrieval of artifacts is made possible by using the CMIS Query Language, which is based on the SQL-92 SELECT statement. Such a language has a syntax particularly troublesome for a user with a poor computer science background, and it requires knowledge on the right term to look for, because if the query contains a synonym of a term contained in the artifact the match is not detected. In order to simplify the query, it is possible to exploit a different approach based on faceted search, which involves augmenting traditional search techniques with means to enable users narrow down search results, thanks to the faceted classification of items.

The similarity between an artifact and the provided query can be detected by using Solr[4], a library based on the Apache Lucene, thanks to a term indexing process. Thus, we have used the API provided by the SearchService of Alfresco to let a user retrieve artifacts employing the term-matching provided by CMIS and the faceted search of Solr. The extraction and analysis phases are performed by the semantic module which will be discussed in the next section.

3.3.1.3 *Semantic Module*

The purpose of this module is to analyze the results extracted from the crawler module and the artifacts saved in the corporate CMS to define those that are closest to the search parameters and the query string. This can be described as an Information Retrieval problem [69], in which it is necessary to exploit Query Expansion techniques [24, 35] for minimizing the query-results mismatch, therefore improving retrieval performances. In order to realise an effective artifact search, techniques from the NLP literature are usually exploited [5, 65, 74, 153]. They permit to overcome the limits of basic text-matching realized by traditional query languages

3 https://chemistry.apache.org/
4 https://lucene.apache.org/solr/

(such as SQL), returning more pertinent results thanks to stemming and similarity operations. Thus, this module benefits not only from the user's query, but also from keywords and their synonyms. In fact, thanks to the interaction with the CMS module, the semantic module exploits the historical knowledge of a company to include several new related terms for analysis.

Figure 3.3 shows the workflow of the semantic module. As we can see, it accomplishes its analysis by considering several phases and techniques. Initially, to determine the artifacts containing the query keywords, the frequency of the keywords contained within each artifact (e.g., a call for tender) is calculated through the TF-IDF weighting scheme [98]. Moreover, other than calculating the keyword frequency, the TF-IDF algorithm [135] also calculates a value that is directly proportional to the frequency of the term in the document, but inversely proportional to the frequency of the term in the entire collection of documents. In this way, common keywords will have a lower value than those appearing less frequently within the artifacts. Although the use of TF-IDF allows the semantic module to efficiently evaluate the results obtained by the crawler module with respect to the performed search, there are some cases in which this technique is not suitable. Indeed, if there are no common terms between a web page and a given topic, we cannot achieve a proper similarity for the web page. As described in [57], this problem can lead to ignoring some links to pages that are pertinent to the search query, since there must also be common terms among the topics of the hyperlinks in order to achieve a fair similarity value of the artifacts. For this reason, the semantic module combines TF-IDF and cosine similarity in order to evaluate results, by also using the anchor text of hyperlinks as its artifacts.

Experimental results demonstrate that the combination of these techniques improves the performance of focused crawlers, outperforming other focused crawlers relying on different metrics and techniques [57]. Nevertheless, in the experimental evaluation proposed in this thesis, we have demonstrated the effectiveness of TF-IDF and cosine similarity with respect to other well-known techniques used in semantic search engines, such as Dice and Jaccard similarities.

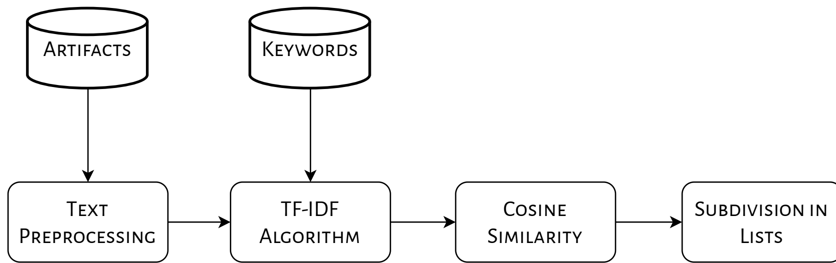


Figure 3.3: Flowchart of the Semantic Module.

Thus, we used the cosine similarity to determine the artifacts that are more similar to the query parameters, based on the frequency calculated in the previous phase. This similarity metric has been chosen for its independence from the length of the artifact, whereby artifacts with the same composition but different word counts will be treated identically. Thus, based on the cosine similarity, the retrieved artifacts have been sorted and divided into the following three lists:

1. **White list**: the subset of artifacts of interest to the user;
2. **Black list**: the subset of artifacts that have been selected by the search engine but have no relevance to the context of the search;
3. **Gray list**: the subset of artifacts whose relevance to the user is uncertain.

To derive the threshold values and distribute the artifacts among the three lists mentioned above, an empirical study has been carried out, aiming to minimize the overlap between artifacts.

3.3.1.4 *Visualisation Module*

This module clusters similar artifacts contained in White and Gray lists, in order to let the user gain immediate insights from retrieved artifacts. The goal of cluster analysis is to group the results based on information found in the data describing artifacts and their relationships, so that artifacts belonging to the same cluster will be related to one another and unrelated to those in other clusters. The greater the similarity within

a group and the greater the difference between groups, the better the clustering.

This type of analysis plays an important role in several areas, such as social sciences [100], documents classification [10, 82, 107], statistics, pattern recognition, information retrieval, data mining, and so on. However, in some cases cluster analysis is only a useful starting point for other purposes, such as data summarization or data visualization. Among the available clustering algorithms, one of the most used and studied is K-means [88], which is an unsupervised learning algorithm able to easily adapt itself to several contexts and to quickly analyze large datasets. In particular, it is a popular cluster analysis method, which partitions a dataset D into K disjoint clusters, such that each element belongs to the cluster with the nearest mean.

Clustering Optimization. One of the drawbacks of the K-means algorithm is that it often convergences to local minima. To tackle this problem, in this thesis we propose an empirical solution to extend the search out of the local minimum, aiming to reach a minimum closer to the global one. In particular, the solution relies on multiple executions of the K-means algorithm with different random starting points. All the obtained solutions are saved and displayed to compare the achieved results. We have tested the proposed extension in the e-procurement domain by using a dataset of calls for tender consisting of 150 calls concerning all the Italian regions, each consisting of 17 features. In order to search the appropriate number of clusters, we used the Elbow method [73]. The latter looks at the percentage of variance expressed as a function of the number of clusters. The method relies on the idea that one should choose a number of clusters so that adding another cluster will not yield improved modeling of the data. The percentage of variance expressed by the clusters is plotted against the number of clusters as shown in Figure 3.4. Notice that, at the same point, the curve drops drastically forming an angle in the graph, which means that the most suitable number of clusters to choose is between 3 and 4. Thus, further tests are necessary to decide whether the optimal number of clusters is 3 or 4. To this end, we exploit the Silhouette method [104], which defines how similar a point is to its own cluster (cohesion) compared to other clusters (separation). Figure 3.5

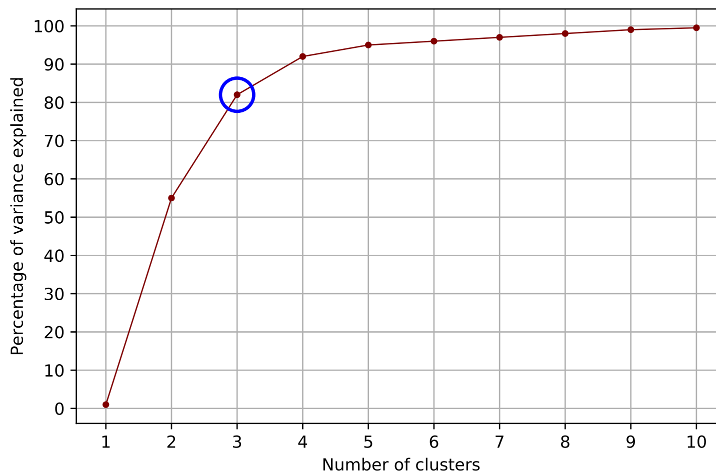


Figure 3.4: Elbow diagram.

shows the average silhouette scores achieved in our tests. Notice that the silhouette score reaches its global maximum at the optimal K. Thus, we have chosen $K = 3$ as the number of clusters for the K-means algorithm.

Each time the clustering algorithm runs, it picks K random seeds to determine the starting centroids of each cluster. In order to guarantee different random seed values, they were generated by using a timestamp with microsecond precision. The algorithm stops the iteration when the distances between consecutive points are less than the given tolerance value, which in our case has been experimentally set to 0.1. The outputs of each algorithm execution are the coordinates, the labels, and the inertia value of the K cluster centers. In order to visualize the obtained clusters and reduce the problem dimensions, the Multidimensional Scaling (MDS) algorithm has been used [20].

The visualization module groups the extracted data into a given number K of clusters, according to features describing the artifacts of interest. In particular, in the e-procurement domain we have selected the following features of calls for tender: *amounts*, *opening dates*, *closing dates*, and *SOA*

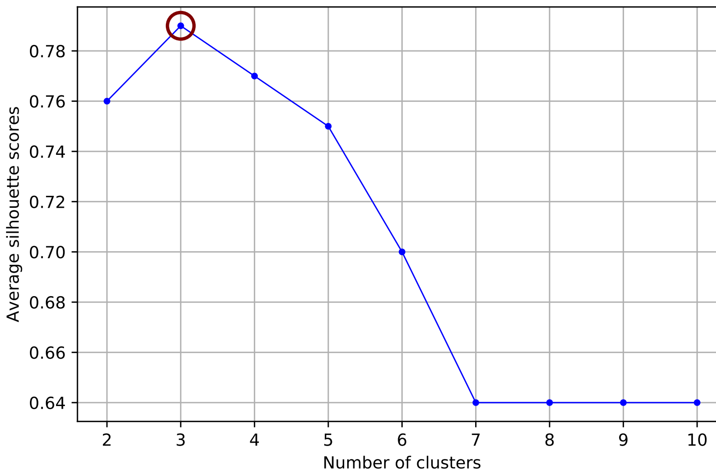


Figure 3.5: Silhouette diagram.

category[5]. As said above, according to the Elbow and Silhouette methods we have selected $K = 3$. Moreover, in order to improve the quality of the final classification and to remove possible false-positive results, the visualization module exploits a dynamic page removal module. For instance, the following are examples of pages considered as dynamic: homepages, pages with a clear IP in the URL, and pages with frequent updates. In this way, all the identified groups are disjointed and their intersection is empty.

3.3.2 *Experimental Results*

The prototype of the proposed system has been developed in *C#* .NET framework 4.7.1[6], based on the Model-View-Controller (MVC) architectural pattern. Each module has been developed standalone and combined into a single .NET solution connected by project references. A RESTful

5 https://www.attestazionesoa.it/
6 https://dotnet.microsoft.com/

Titolo	Descrizione	Link
Gazzetta Ufficiale	ministero infrastrutture e trasporti provveditorato interregionale per le oo.pp. per il lazio, labruzzo e la sardegna sede: via monzambano n. 10 - 00185 roma codice fiscale: 97350070583 (gu 5a serie speciale - contratti pubblici n.126 del 29-10-2018) esito di gara - procedura aperta per laffidamento della progettazione esecutiva e della realizzazione dei lavori di ristrutturazione, trasformazione ed ampliamento degli impianti tecnologici delle sedi della sogei - societa generale dinformatica s.p.a. in roma, via mario carucci n. 99 nonche allaffidamento del servizio di supporto tecnico specialistico agli impianti tecnologici interessati dallintervento - cup d84e14000830005 - cig 7387004c88 procedura aperta esperita presso questo provveditorato nelle sedute pubbliche dei giorni 23/04/18, 04/05/18, 15/05/18, 17/05/18, 04/06/18 e 09/08/18 per laffidamento della progettazione esecutiva e della realizzazione dei lavori di ristrutturazione, trasformazione ed ampliamento degli impianti tecnologici delle sedi della sogei -societa generale dinformatica s.p.a. - in roma, via mario carucci n.99 nonche allaffidamento del servizio di supporto tecnico specialistico agli impianti tecnologici interessati dallintervento. importo a base dasta di €.21.970.383,93. numero offerte ammesse: 11. aggiudicataria definitiva: gruppo ecf spa, con sede in roma, via curtatone n.4, cap.00185, con il miglior punteggio totale ottenuto di 96,900 (offerta tecnica 70 punti, offerta tempo 5 punti, offerta economica 21,90 punti), il ribasso offerto del 29,550% e la riduzione offerta di giorni 210 sul tempo complessivo di esecuzione, non anomala; il provveditore ing. vittorio rapisarda federico tx18bga23075 realizzazione istituto poligrafico e zecca dello stato s.p.a.	http://www.gazzettau...
Albo Pretorio | Amministrazione Aperta	Consorzio Intercomunale Gestione Rifiuti BN 1 in Liquidazione c/Provincia di ... Bando di Gara · PROCEDURA APERTA AFFIDAMENTO SERVIZI INGEGNERIA ...	http://app1.provinci...
Gazzetta Ufficiale	ministero delle infrastrutture e dei trasporti provveditorato interregionale per le opere pubbliche campania - molise - puglia - basilicata sede di napoli sede: via marchese campodisola n. 21 - 80133 napoli punti di contatto: pec: oopp.campaniamolise-uff1@pec.mit.gov.it (gu 5a serie speciale - contratti pubblici n.126 del 29-10-2018) esito di gara si rende noto, a norma degli artt. 72 del d.l.vo n. 50/2016 e s.m.i., che questo provveditorato ha concluso la procedura aperta per laffidamento dei servizi di architettura e ingegneria per la redazione della progettazione della fattibilita tecnica economica, definitiva ed esecutiva, e del coordinamento della sicurezza in fase di progettazione, afferenti gli interventi di conservazione, manutenzione, restauro e valorizzazione dellabbazia del goleto in santangelo dei lombardi (av). cup: d64i18000070005 - cig: 7466766a38 - cpv: 71310000- hanno presentato offerta nel termine n. 16 imprese. ditte escluse: n. 5. con decreto provveditoriale n.27614 del 12.10.2018, e stata dichiarata laggiudicazione dei servizi in oggetto in favore del rtp ing. toscano giovanni (capogruppo) - arch. giosue amoroso (mandante) - ing. francesco bocchino (mandante) - geol. gennaro dagostino (mandante), con sede in aversa alla via san paolo n.25 c.f e p.i. 01500490618 che ha conseguito un punteggio totale di punti 87 34 e per il corrispettivo di € 24.761.50 al netto del ribasso offerto del 52%. il provveditore vicario dott.ssa vania de cocco tx18bga23130 realizzazione istituto poligrafico e zecca dello stato s.p.a.	http://www.gazzettau...

Figure 3.6: Results produced by CAIMANS for the e-procurement case study.

API service has been integrated into the solution to simplify inquires to both the crawler and the CMS.

The screenshot in Figure 3.6 shows the results produced by CAIMANS for the e-procurement case study. It shows the extracted calls for tender, sorted by the similarity values of results, according to the search parameters. For each result, the table shows the title, a short preview of the artifact, and the referring URL. The background color is one of the lists to which the document has been assigned. Figure 3.7 shows the three clusters that the visualization module has identified for each extracted call for tender. Each cluster is represented by a different color, and the size of the individual circles indicates the similarity value of the retrieved artifact with respect to the query parameters.

3.3.2.1 *Evaluation Criteria*

The most important measure for evaluating a search engine is the relevance of the retrieved results with respect to the search parameters. Generally, the algorithms underlying web search engines analyze page semantics and the number of references to the page, aiming to reduce the number of retrieved results and increase their quality.

In this study, before comparing the performances of CAIMANS with those of similar systems, we have first compared it with Google by using precision and recall metrics computed on the set of the retrieved artifacts. To this end, in order to construct the confusion matrix, a domain expert

Figure 3.7: Displaying the result of the clustering phase.

Figure 3.8: Semantic Search form - results.

needs to be involved in the evaluation phase, in order to define true positives, false positives, and false negatives, whereas true negatives were automatically determined by CAIMANS through the blacklist. Since Google does not divide results like CAIMANS, rather it retrieves the list of one hundred artifacts more pertinent to the search query, we have derived two different formulations of the precision measure. In particular, based on the metrics used in [83], the precision measure for Google has been defined as follows:

$$P_{Google} = \frac{\sum_{i=1}^{|White|} r_i}{|White \cup Gray \cup Google|} \tag{3.1}$$

Where *White* and *Gray* are the lists retrieved by CAIMANS, whereas *Google* are the results retrieved by Google, and the rank r_i is a number between 0 and 1 representing the semantic similarity of the i-th artifact in *White* with respect to the search query.

Similarly, the precision measure for CAIMANS has been defined has follows:

$$P_{\text{CAIMANS}} = \frac{\sum_{i=1}^{|White|+|Gray|} r_i}{|White \cup Gray \cup Google|} \tag{3.2}$$

In addition to 3.2, other than the ranks of the artifacts in *White*, 3.2 also considers the ranks of the artifacts in *Gray*. As said above, the ranks are calculated by the semantic module. The domain expert determined the true positives among the retrieved artifacts. Analogously, the recall measures for the two compared systems have been defined according to the following formulas:

$$R_{Google} = \frac{|TP_{Google}|}{|White| + |Google|} \tag{3.3}$$

$$R_{\text{CAIMANS}} = \frac{|TP_{White}|}{|White| + |Google|} \tag{3.4}$$

where TP_{Google} and TP_{White} are the true positive results extracted from Google and CAIMANS, respectively, according to the selections made by the domain expert.

In order to express the accuracy of the proposed search engine through a unique measure, the F-measure was used to combine the precision and recall metrics. Starting from the definitions of precision and recall given in 3.2-3.4, we have derived the following formula for the F_1-measure:

$$(F_1)_{Google} = 2 * \frac{P_{Google} * R_{\text{CAIMANS}}}{P_{Google} + R_{\text{CAIMANS}}} \tag{3.5}$$

$$(F_1)_{\text{CAIMANS}} = 2 * \frac{P_{\text{CAIMANS}} * R_{\text{CAIMANS}}}{P_{\text{CAIMANS}} + R_{\text{CAIMANS}}} \tag{3.6}$$

3.3.2.2 *Semantic Web Search evaluation*

In this section, we experimentally show how the cooperation between the semantic and the crawling modules improves search effectiveness with respect to traditional search engines. To this end, we compared the results achieved by CAIMANS with those achieved by Google in the e-procurement domain. In particular, in order to evaluate the correctness of the results of each search session, we involved a user with expertise in the domain of e-procurement, focusing on the selection of findings related to the search target.

Tests were performed on a virtual machine running on a Mac with an Intel Xeon 3.20 GHz processor and 64GB of RAM. More specifically, 8GB of RAM and 250GB of local disks were dedicated to the virtual machine. The search session was accomplished using a fast connection at 340 Mbps/sec.

During the tests, we defined a single search configuration, with a unique set of keywords, URL seeds, and stopwords, aiming to carry out a peer analysis of all search sessions. As known, among millions of analyzed pages, Google only shows the best 100 results, which contain all true and false positives, depending on the performed search. For this reason, and also because of the burden of the manual evaluation by the domain expert, the number of search sessions accomplished during the tests was limited to 175. Table 3.1 shows a part of the query strings

ID	Query String	#Words	Time of CAIMANS (s)
1	Bandi di gara Bologna	3	2047
2	Onlus e gare d'appalto	3	1922
3	Autostrade gare appalto	3	2279
4	Bandi di gara Bolzano	3	2106
5	Bandi di gara Salerno provincia	4	249
6	Bandi di gara regione Sardegna	4	2230
7	Bandi di gara regione Calabria	4	2238
8	Gare d'appalto Banca d'Italia	4	256
9	Bandi di gara distributori automatici 2019	5	225
10	Bandi di gara beni culturali Puglia 2019	5	243

ID	Google Results	TP_{Google}	CAIMANS Results	Common Results	|Black|	|Gray|	|White|	$TP_{CAIMANS}$
1	100	18	544	11	2	55	487	538
2	100	4	478	2	24	70	384	435
3	100	3	621	2	22	164	435	563
4	100	8	621	5	22	126	473	594
5	100	1	67	0	22	22	23	27
6	100	18	505	13	24	131	350	474
7	100	25	475	18	34	105	336	393
8	100	1	75	1	43	15	17	25
9	100	1	77	1	43	17	17	26
10	100	0	80	0	46	18	16	27

ID	Rate of Dynamic Pages	P_{Google}	$P_{CAIMANS}$	R_{Google}	$R_{CAIMANS}$	$(F_1)_{Google}$	$(F_1)_{CAIMANS}$
1	0,74	0,46	0,93	0,03	0,82	0,59	0,87
2	4,19	0,41	0,83	0,01	0,78	0,54	0,80
3	6,01	0,40	0,79	0,01	0,79	0,53	0,79
4	0,83	0,38	0,75	0,01	0,82	0,52	0,79
5	35,56	0,37	0,75	0,01	0,11	0,16	0,19
6	1,46	0,36	0,72	0,04	0,77	0,49	0,75
7	10,88	0,38	0,76	0,06	0,72	0,50	0,74
8	15,63	0,34	0,68	0,01	0,12	0,18	0,20
9	14,71	0,31	0,63	0,01	0,13	0,18	0,21
10	14,71	0,31	0,62	0,00	0,13	0,18	0,21

Table 3.1: Results obtained from the queries used in the experimental evaluation.

used in our experiments. In particular, we evaluated different generated queries, such as two, three, four, and five word queries.

The first type of query involved four to five words, such as "bandi di gara beni culturali Puglia 2019" (public procurements for cultural heritage Puglia region year 2019) or "bandi di gara Salerno provincia" (public procurements in the province of Salerno). However, based on the results extracted by Google for these types of queries, the domain expert considered only a few of them as pertinent, filtering 0 calls in many cases. Vice versa, CAIMANS always retrieved a high number of valid results, even when Google's true positive results were few. The second type of query involved two to three words, such as "gare d'appalto Basilicata" (public procurements in Basilicata region) or "gare d'appalto Bologna" (public procurements in the city of Bologna). Even though the domain expert filtered a conspicuous number of results among those returned by Google for these types of queries, with CAIMANS we could considerably increase the size of the result set. Successively, the domain expert was asked to perform a further filtering phase, in which only the results classified in the White and the Gray lists were re-examined. Table 3.1 shows some results of the evaluation phase, in which a set of query strings with different numbers of words were submitted. We can observe that the number of artifacts that Google extracted and classified as true positive (TP_{Google}) is always lower than CAIMANS (TP_{CAIMANS}). We can notice that even when Google did not find useful results, CAIMANS often extracted several pages related to the search target.

The semantic module plays a fundamental role within CAIMANS, since it discards all uninteresting results and is able to identify pages related to the search criteria, increasing the quality of extracted pages. Figure 3.9 shows the precision values computed for CAIMANS, according to the formulas (3.1) and (3.2). In particular, the average value is higher when considering all the true positives of the White and the Gray lists. Moreover, the second filtering activity accomplished by the domain expert has further refined search results. The achieved precision values show that the CAIMANS provides a high number of relevant results in the focus centered search.

By using formulas (3.3) and (3.4), it was possible to define the values of the recall, i.e. the probability that a pertinent artifact is retrieved in the focused centred search. Figure 3.10 compares the recall achieved by

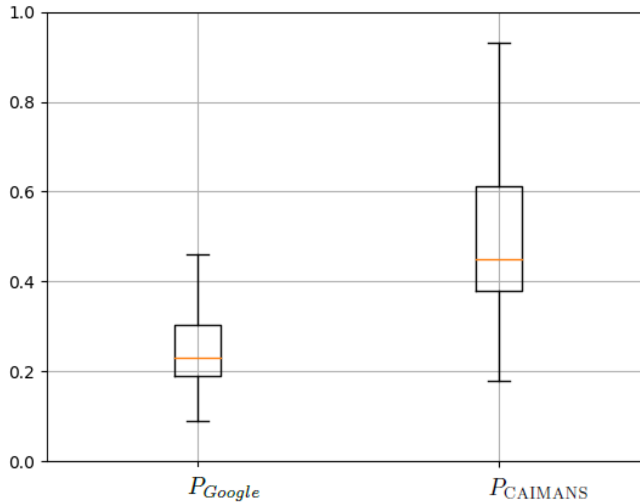


Figure 3.9: Precision evaluation for Google and CAIMANS.

CAIMANS to that of Google. The results show that the average recall of CAIMANS is higher than the one achieved by Google. However, when generic strings were used, such as "bandi di gara" (public procurements) or its synonym "gare d'appalto" (calls for tender), the values of the recall for the two compared systems turn out to be similar.

The main goal of this evaluation was to show the effectiveness of CAIMANS for focused search by considering different types of search operations. Although time performances of CAIMANS are quite good (see Table 3.1), we cannot compare them with Google, since the latter is able to browse a large part of the web in few seconds by exploiting efficient algorithms and scalable architectures. For this reason, we have chosen to design CAIMANS as a batch system, which is able to explore the web through multiple scheduled searches. Experimental results show that the performances are quite similar for queries with a different number of words.

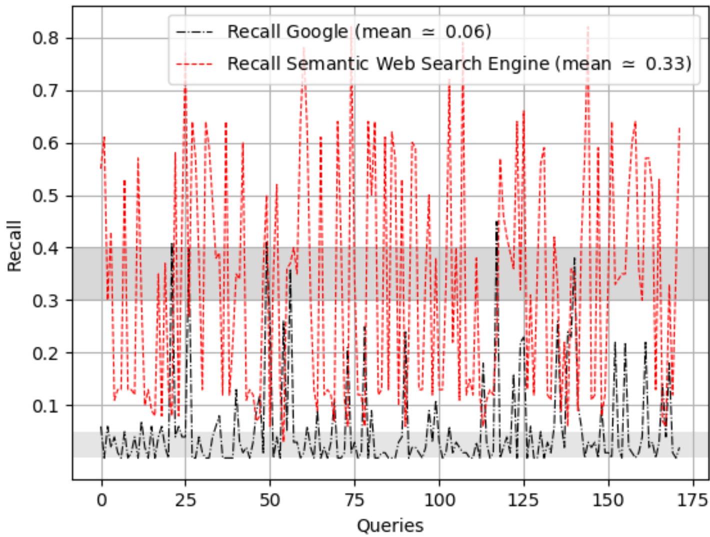


Figure 3.10: Recall evaluation for Google and CAIMANS.

3.3.2.3 *Comparative evaluation*

In this section, we show the results of a comparative evaluation among CAIMANS and similar systems, on multiple domains. Since to the best of our knowledge, in the literature there is no system similar to CAIMANS, we performed such comparison on its main module, i.e., the crawler. To this end, we selected two focused crawlers, one based on Breadth-First Search (BFS) and the other on Depth First Search (DFS)[7], and plugged them in turn within the CAIMANS's architecture.

A BFS crawler relies on the Breadth-First Search of a tree or chart [23]. The exploration starts with the seed URLs associated to the current domain, and for each of them, the crawler saves all the links whose depth is one more than the analysed URL. After exploring all the URLs in one level, the crawler scans the pages at the next level by exploiting the same strategy. On the contrary, a DFS crawler relies on the Depth-First Search of a tree or graph [156]. The exploration starts with the seed URLs, and

7 https://github.com/ethanZHY/Crawler_BFS_DFS_Python3

ID	Domain	Seed URLs
1	Gare d'appalto ("Calls for tender")	www.gazzettaufficiale.it www.ooppcampania-appalti.maggiolicloud.it www.ansa.it
2	Robot Programmabili ("Programmable Robot")	www.robot-advance.com www.wiki.ezvid.com www.softbankrobotics.com
3	Equipaggiamento per Hockey ("Hockey equipment supply")	www.decathlon.it www.skatepro.it it.hockeyoffice.com
4	Forniture per uffici ("Office supplies")	www.visualcapitalist.com www.oknoplast.it www.prontopro.it
5	Giochi da tavolo ("Table Games supplies")	www.boardgamesofferte.it www.boardgameitalia.it it.wikipedia.org
6	Cloud Provider ("Cloud providing")	www.zerounoweb.it www.meteo.it www.techcompany360.it

Table 3.2: Details of the topics and seed URLs used for the comparative evaluation on e-procurement domain.

for each of them, it performs a deep scan until all URLs on that path are retrieved. Then, it goes back to scan other branches of the tree.

All the three configurations of CAIMANS with the different compared crawlers were run on the same application domains, and for each of them, we configured the semantic module to evaluate the resulting pages by means of the Cosine Similarity, the Dice Similarity [146], and the Jaccard Similarity [119] metrics.

Table 3.2 shows the domains and the corresponding seed URLs used in our evaluation. In particular, 6 domains were divided into two categories based on their seed URLs: the first category containing domains in which the seed URLs are closely related to the target topic: "Gare d'appalto" ("Calls for tender"), "Robot programmabili" ("Programmable Robot"), and "Equipaggiamento per Hockey" ("Hockey equipment sup-

ply”), whereas the second one contains domains not directly related to the target topic: “Forniture per uffici” (“Office supplies”), “Giochi da tavolo” (“Table games supplies”), and “Cloud Provider”. Notice that, all the tests with the three system configurations have been accomplished by considering common keywords and stopwords for each target topic.

Table 3.3 shows the results of the comparative evaluation on selected domains, highlighting the number of web pages retrieved and the results of the semantic module for each execution. Each crawler was run assuming that the maximum number of pages to browse was 1000. However, if all URLs within the queue were crawled, i.e., if the URL queue was empty during execution, the crawler would stop its crawling process. Results show that the number of web pages extracted by CAIMANS is greater than those extracted by the configuration with the DFS and BFS crawlers. This is probably due to the fact that such crawlers mainly focus their exploration on the subdomains of the seed URLs (Table 3.2). Vice versa, CAIMANS is able to prioritize searches in web domains beyond the subdomains of the seed URLs, by exploiting a search strategy in which the order of the links within the URL queue changes continuously (Section 3.3.1.1).

Concerning the comparative evaluation of similarity metrics, results show that the number of pages of interest extracted by CAIMANS and BFS crawlers using an evaluation based on the Cosine Similarity is always greater than the one achieved with other similarity metrics. Moreover, it is important to notice that several results classified as not relevant (i.e., black list) from the semantic modules with Dice and Jaccard metrics have been evaluated as uncertain relevant results (i.e., grey list) with the Cosine Similarity metrics. For this reason, these results have been individually assessed, and all of them have been classified as artifacts of interest.

Although these results show the effectiveness of the Cosine Similarity, results in Table 3.3 show that there are some exceptions for the DFS crawler, in which the number of pages of interest evaluated with Dice and Jaccard similarity metrics is greater than the one achieved with the Cosine Similarity. In particular, the semantic module configured with the Jaccard Similirarity for the DFS crawler outperforms the results of

Topic	CAIMANS		Cosine Similarity			Dice Similarity			Jaccard Similarity		
	Results	Time (s)	|Black|	|Gray|	|White|	|Black|	|Gray|	|White|	|Black|	|Gray|	|White|
1	882	1541	377	250	255	494	265	123	597	198	87
2	241	1289	101	119	21	152	76	13	150	82	9
3	134	1276	59	47	28	76	37	21	85	45	4
4	859	1541	298	303	258	421	234	204	476	214	169
5	47	97	9	11	27	18	3	26	11	27	9
6	241	261	51	68	122	130	53	78	72	77	112

Topic	DFS		Cosine Similarity			Dice Similarity			Jaccard Similarity		
	Results	Time (s)	|Black|	|Gray|	|White|	|Black|	|Gray|	|White|	|Black|	|Gray|	|White|
1	31	10520	8	17	6	12	17	2	16	11	4
2	68	5546	25	38	5	37	25	3	42	7	19
3	22	8854	7	11	4	15	6	1	10	8	4
4	33	11201	22	8	3	17	13	3	25	6	2
5	37	9476	20	9	8	22	10	5	14	13	10
6	60	2272	34	18	8	31	21	8	17	37	6

Topic	BFS		Cosine Similarity			Dice Similarity			Jaccard Similarity		
	Results	Time (s)	|Black|	|Gray|	|White|	|Black|	|Gray|	|White|	|Black|	|Gray|	|White|
1	20	135	20	0	0	17	3	0	19	1	0
2	4	4437	3	0	1	4	0	0	3	0	1
3	18	1404	12	1	5	10	5	3	12	4	2
4	149	2625	39	98	12	72	69	8	71	73	5
5	29	240	10	13	6	16	7	6	15	9	5
6	9	186	6	3	0	8	1	0	9	0	0

Table 3.3: Results of the comparative evaluation on general topics.

other modules for the topics “Programmable Robot” and “Table Games supplies” (i.e., Topic 2 and 5). However, several results in the white list have been included in the grey lists of the other two semantic modules configured with Dice and Cosine Similarity, respectively.

Part II

DISCOVERY ALGORITHMS FOR DATA PROFILING

4 DISCOVERY ALGORITHMS IN STATIC SCENARIOS

The availability of massive quantities of data yields the possibility to enhance data-intensive processes such as data analytics, data fusion, predictive model training, and so forth. Nevertheless, such processes not only to be robust with respect to errors or dirty representations of data but also to collect metadata capable of characterizing statistics and relationships among data. As discussed in the previous sections, one such type of relationship is represented by RFDs, since due to the use of approximate matching paradigms, and the possibility of admit exceptions, they exhibit a suitable level of robustness. However, it is difficult to specify RFDs together with their thresholds, so it is vital to develop algorithms that can automatically discover them from large amounts of data, minimizing the amount the information to be provided by the user.

In this chapter, we first introduce the discovery problem for RFDs in a static scenario, and then we will discuss a discovery algorithm for hybrid RFDs relying on evolutionary approaches.

4.1 PROBLEM DESCRIPTION

The main goal of RFD discovery algorithms is to find RFDs holding to one or more collections of data. However, although the availability of big data collections stimulates the exploitation of RFDs, the large data volumes and the necessity to derive proper settings of RFD thresholds make the discovery process computationally expensive [37].

The definition of algorithms and methodologies capable of efficiently discovering RFDs has aroused interest since the early '80s [84, 85, 111, 144]. However, it is still an open challenge for research communities [108], since there are many RFDs for which efficient algorithms have not

been defined yet [32]. As described in [1], the discovery of RFDs is more complex than the discovery of FDs, even though the number of potential FDs can be exponential and their discovery might require to analyzing a huge number of attribute combinations. In fact, RFD discovery algorithms adopt some relaxation criteria to validate RFDs that might cause an increase in complexity. In particular, the RFDs relaxing on the extent based on a coverage measure prevent the exploitation of properties simplifying the validation phase, since they admit exceptions to their validity. This type of RFD might admit restrictions on the validity domain specified by means of conditions, requiring the identification of the set of attributes on which to specify such conditions [37]. Similar consideration apply to the RFDs relaxing on the attribute comparison method. In particular, for this type of RFDs it is not possible to exploit the disjointness property of equivalence classes induced by the equality comparison method, since they yield intersecting similarity sets, which do not guarantee the transitivity property useful to simplify the validation phase. Moreover, this complexity becomes even worse when no thresholds are defined as input, since it is also necessary to identify patterns of value similarity representing valid RFDs. This causes the size of the search space to be related to the distribution of value similarities [37].

Starting from these notions, we can now provide the general definition and the complexity for the FD and the RFD discovery problems.

Definition 4.1.1 (FD **Discovery Problem**)**.** Given a relation instance r of a relation R, an FD discovery algorithm aims to find the set of all minimal FDs holding in r, having the property that tuples equal on the LHS must be equal also on the RHS.

Definition 4.1.2 (RFD **Discovery Problem**)**.** Given a relation instance r of a relation R, an RFD discovery algorithm aims to find the set of all minimal RFDs holding in r, having the property that tuples similar on the LHS must be similar also on the RHS. This must be true for a subset of tuples, based on the threshold specified for the coverage measure, and/or on the conditions defining valid patterns of values.

The complexity for discovering RFDs relaxing on the extent (RFD_es) by using a coverage measure is equal to the complexity for discovering FDs.

In particular, discovering RFD$_e$s (FDs) over a relation instance r entails finding all the possible column combinations, and for each of them find all the possible partitions forming the LHS and the RHS of candidate RFD$_e$s (FDs). Without loss of generality, we can consider only candidates with a single attribute on the RHS. Given this, the RFD$_e$s (FDs) discovery problem has an extremely large search space. In fact, given a relation R with M attributes and a relation instance r with n tuples, we need to consider all the possible attribute combinations with size k from 2 to M, counting each of them as many times as the number of attributes in R, in order to account for the number of different candidates with a single RHS attribute. This complexity is synthesized by the following formula:

$$\sum_{k=2}^{M} \binom{M}{k} k \tag{4.1}$$

Since this complexity represents only the number of candidate FDs that could be potentially checked, discovery algorithms need to tackle several other issues, such as the validation of each candidate FD. More specifically, for the FD discovery problem, candidate FD validation is linear in the number of tuples. Such complexity is the same when discovering RFDs relaxing on the extent only through a coverage measure. However, the fact that these RFDs can also be valid for a subset of tuples prevents the possibility of exploiting several pruning strategies typical of FD discovery algorithms. This is not true when discovering RFDs relaxing on the extent only through a condition rather than a coverage measure. In this case, the necessity to generate candidate conditions considerably increases the size of the search space. This also depends on the types of conditions that could be generated. In general, the problem of generating an optimal pattern of conditions that could be associated to a candidate RFD is NP-Complete [71]. Vice versa, the number of candidate RFDs can considerably increase when considering RFDs relaxing on the attribute comparison (RFD$_c$s). To this end, it is necessary to consider two different cases, depending on whether or not a discovery algorithm considers pre-defined similarity thresholds associated with the attributes of the dataset. In the

first case, the complexity corresponds to that already defined in Formula 4.1, whereas the validation problem becomes quadratic in the number of tuples, since the transitivity property is not satisfied for value similarities [37]. In the second case, the search space is far more complex since for each candidate, it is necessary to consider d^k possible dispositions with repetitions of thresholds for the selected attributes, where d represents the number of possible distance values between tuple pairs. For this reasons, starting from the Formula 4.1, we can synthesize the overall complexity for discovering RFD_cs as:

$$\sum_{k=2}^{M} \binom{M}{k} \, k \, d^k \tag{4.2}$$

Moreover, also in this case the RFD validation is quadratic in the number of tuples.

4.2 LITERATURE REVIEW

The problem of discovering data dependencies was initially addressed with the automatic discovery of canonical FDs. In fact, the discovery algorithms for FDs have set the basic concepts and the main strategies to tackle an extremely complex problem (i.e., exponential in the number of attributes) with acceptable execution times [126]. However, the RFD discovery problem is even more complex, due to the necessity of considering a generalized data property, which permits to restrict the validity of a dependency over subsets of data (i.e., *relaxation on the extent*), and/or to compare data in terms of their similarity rather than equality (i.e., *relaxation on the attribute comparison*).

Although recent surveys listed many different types of RFDs [32, 149], few of them are equipped with algorithms for discovering them from data [108]. The RFDs relaxing on the extent (i.e., RFD_e) are also known in the literature with the name *Approximate Functional Dependencies* (AFDs) or *Partial Dependencies*. RFD_es measure the amount of data satisfying the

dependency, namely *satisfiability degree*, through a coverage measure [70], among which the g_3-error measure is the most frequently used [90].

Another strategy to determine the validity of a RFD for a subset of tuples is to filter input data by specifying conditions, in order to derive the domain on which the RFD holds [19], yielding the concept of Conditional Functional Dependency. Among the approaches for this class of RFDs, the algorithm proposed in [42] generates RFD candidates by exploiting the attribute lattice derived from the partitions of attribute values. Alternatively, the greedy algorithm proposed in [71] tries to derive valid RFDs by finding a close-to-optimal tableau, where the closeness is measured by using *support* and *confidence* measures. Indeed, the method proposed in [89] exploits the $g3$-error measure of super keys to determine the approximate satisfiability degree. In [63] authors adapted three algorithms introduced for FD discovery, namely TANE [85], FD_Mine [164], and FastFD [162], to tackle the problem of discovering RFD_es.

A novel and efficient algorithm for RFD_e discovery is PYRO [93]. It aims to find minimal RFD_es by exploiting both samples of agree sets, and a lattice traversal strategy, which uses efficient data structures to estimate and calculate the error of RFD_e candidates. PYRO also uses parallelism in order to speed up the runtime. Instead, the method proposed in [89] exploits the error measure of super keys to determine the extent of RFD_es. Finally, to cope with the complexity of the RFD_e discovery problem, there are some approaches limiting the discovery only to meaningful RFD_es, yielding the loss of some minimal RFD_es holding on the database instance [139].

The RFDs relaxing only on the attribute comparison (i.e. RFD_cs) differ in the way they manage similarity predicates to accommodate errors, and different representations in unreliable data sources [62]. Thus, discovery algorithms have to evaluate RFDs by means of similarity/difference constraints that are composed of similarity/difference functions, operators, and thresholds. The latter could be set as input parameters or automatically inferred from data [31, 37]. One of the most recent discovery algorithms for this class of RFDs evaluates the utility of a candidate RFD for a given database instance by determining the corresponding similarity threshold pattern [148]. The utility is measured by means of

support and confidence, whereas thresholds are determined by analyzing the statistical distribution of data. Another approach for discovering RFDs relaxing on the attribute comparison has been proposed in [147]. It uses differential functions to evaluate the similarity between tuple values, and it exploits reduction algorithms to detect valid RFDs by first fixing the Right-Hand-Side (RHS) differential function for each attribute in the database schema, and then finding the set of differential functions that reduce the Left-Hand-Side (LHS). The performances of the algorithm are improved through pruning strategies based on the subsumption order of differential functions, implication property, and instance exclusion. Domino is a recent proposal for the discovery of RFDs relaxing on the attribute comparison [31]. It is able to infer RFDs together with difference thresholds by exploiting the concept of multi-attribute dominance. After analyzing difference patterns between tuple pairs, the dominance permits to reduce the number of patterns to be considered, and to define threshold boundaries above which a candidate RFD does not hold. Instead, another discovery algorithm for the same class of RFDs is defined in [96], which tries to reduce the problem search space by assuming a user-specified distance threshold as an upper limit for the distance intervals of the LHS. The algorithm is based on a distance-based subspace clustering model, and exploits pruning strategies to efficiently discover RFDs when high threshold values are specified. As opposed to these discovery algorithms, which focus each on a specific class of RFDs, the discovery algorithm proposed in [33, 37] aims to identify RFDs relaxing on both the extent and the attribute comparison, namely *hybrid* RFDs [32]. In particular, the algorithm proposed in [37], namely DiMε, relies on a lattice-based algorithm conceived for FD discovery, which is fed with similar subsets of tuples derived from previously computed differential matrices. In this work, the authors also demonstrate the NP-hardness of the RFD discovery problem for hybrid RFDs, yielding the necessity of designing approximate solutions.

As we have seen, several research communities have tried to design and develop efficient techniques for extracting and using this type of metadata, but there are still many challenges to be faced. Nevertheless, FDs and RFDs have been widely used for different purposes, among

which data cleaning and machine learning tasks, such as for feature selection and for the evaluation of the correlation between the dependent attribute and the predictive ones. In particular, authors in [159] combine FDs and the K-Nearest Neighbourhood (KNN) to propose an innovative feature selection algorithm KNN-FD, claiming that it avoids the overfitting problem during the prediction phase. In [99] the authors use FDs to build decision trees, aiming to derive more a compact structure in order to improve accuracy. In [4] authors present a framework for training and evaluating a class of statistical learning models inside a relational database. In particular, they exploit FDs holding on a relation instance to reduce the dimensionality of the underlying optimization problem. In fact, the experimental evaluation demonstrates that the usage of FDs permits to obtain the best results by optimizing some parameters that functionally determine others. A recent study exploits RFDs to tackle the problem of evaluating the feasibility of classification in machine learning models [103]. In particular, the authors show that the usage of RFDs in this domain can provide fundamental contributions to data scientists. In fact, experimental results show evidence that RFDs provide a tight upper bound for the accuracy of classification tasks on real-world datasets. Furthermore, a careful experimental evaluation shows that RFDs provide excellent results in deriving classification models for synthetic datasets, which represent an extremely challenging task on this type of data.

4.3 A GENETIC APPROACH FOR DISCOVERING HYBRID RFDS

In this section, we present a RFD discovery algorithm, named REDEVO (RElaxed fD EVOlutionary discovery algorithm), which relies on a genetic algorithm. The latter is usually adopted for providing efficient global searches for problems with large search spaces, even though it does not always guarantee an optimal solution. More specifically, genetic algorithms exploit operations inspired by the evolution of natural species, such as natural selection, crossover, and mutation [136]. According to these principles, REDEVO adopts these evolutionary operations to the RFD discovery process.

4.3.1 *Methodology*

REDEVO is able to discover hybrid RFDs, i.e., RFDs which relax on both *extent* and *attribute comparison*. Before presenting it, we need to introduce some basic notions underlying it. As mentioned in the FD definition described in Section 2.2.2, the projections of two tuples over a subset of attributes are compared by means of the equality constraint. This is one of the two FD dimensions that have been modified in order to define RFDs, by enabling the use of tuple comparisons based on *constraints*. In these types of algorithms, we focus on the concept of *difference constraint*, which is a predicate evaluating whether the distance between two values is less or equal to a predefined threshold.

Definition 4.3.1 (Difference Constraint). Let r be a relation instance of a relation schema R, $\mathbb{D}$ a set of distance functions defined over each attribute domain. A difference constraint ϕ is a logic expression of the form ($\phi_1 \wedge \ldots \wedge \phi_m$), where ϕ_k is a predicate $\delta(t_i[A], t_j[A]) \leq c_k$, with t_i and t_j tuples of r, $t_i[A]$ and $t_j[A]$, respectively, their projections, on $A \in attr(R)$, $\delta \in \mathbb{D}$, and c_k a threshold.

In other words, a predicate of a difference constraint depends on a distance function defined on an attribute domain, plus the $\leq$ comparison operators with associated threshold values defining the feasible distance between attribute values. As an example, let us consider the snippet of the *Michelin-starred restaurants* dataset shown in Table 2.1. Let $latitude_1$ and $latitude_2$ be the values of attribute Latitude for two different tuples in the dataset. Then, the constraint $abs(latitude_1, latitude_2) \leq 1$ is satisfied if the absolute difference between the two values is below the threshold value 1. Thus, for instance, the values $latitude_1 = 41.85904$ and $latitude_2 = 41.91328$ of the attribute Latitude satisfy the above defined difference constraint.

In what follows, we will introduce the concept of *minimal* RFD based on the observation that once an RFD φ is found from it, many more RFDs can be derived from φ by varying its threshold values or adding attributes to its LHS.

Definition 4.3.2 (Minimal RFD). An RFD $X_{1(\leq\alpha_1)} \ldots X_{n(\leq\alpha_n)} \xrightarrow{\Psi\geq\varepsilon} A_{(\leq\beta)}$ holding on a relation instance r is said to be *minimal* iff

1. $X_{1(\leq\alpha_1+\varepsilon_1)} \ldots X_{n(\leq\alpha_n+\varepsilon_n)} \xrightarrow{\Psi\geq\varepsilon} A_{(\leq\beta-\varepsilon_{n+1})}$ does not hold on r, where $\varepsilon_i \geq 0$ and $\exists\, j$ such that $\varepsilon_j > 0$, with $1 \leq i, j \leq n+1$; and

2. $X_{1(\leq\alpha_1)} \ldots X_{i-1(\leq\alpha_{i-1})} X_{i+1(\leq\alpha_{i+1})} \ldots X_{n(\leq\alpha_n)} \xrightarrow{\Psi\geq\varepsilon} A_{(\leq\beta)}$ does not hold on r, for any $1 \leq i \leq n$.

In other words, a minimal RFD no longer holds if we increase its LHS thresholds or decrease its RHS threshold. The same happens when removing one of its LHS attributes.

Starting from these brief notions behind the RFDs discovery process, we can describe the REDEVO algorithm. Figure 4.1 provides an overview of its underlying discovery process, introducing the interactions between its main phases.

In following sections, we first introduce the preprocessing operations performed to effectively evaluate difference constraints, and then describe the encoding technique used for each individual of the population, and the strategies adopted for the selection, the crossover, and the mutation steps. Finally, we analyze the termination strategies adopted to make the algorithm stop its exploration when reaching a satisfactory solution.

4.3.1.1 *Difference dataset*

As said above, the RFD discovery process needs to evaluate the similarity of tuples on subsets of attributes. To this end, in this dissertation we consider difference constraints defined through difference functions and user-specified difference thresholds. Moreover, we use a coverage measure to determine the satisfiability degree for each candidate RFD. Also in this case, the evaluation of candidate RFDs is performed according to a user specified threshold, i.e., the extent threshold. Based on these parameters, REDEVO creates a set of *pattern tuples* from an input dataset by evaluating the difference constraints between the attribute values of each tuple pair.

Definition 4.3.3 (Pattern tuple)**.** Let $\mathcal{R}$ be a relational database schema over a set of attributes $attr(\mathcal{R})$, $R = \{A_1, \ldots, A_m\}$ a relation schema of

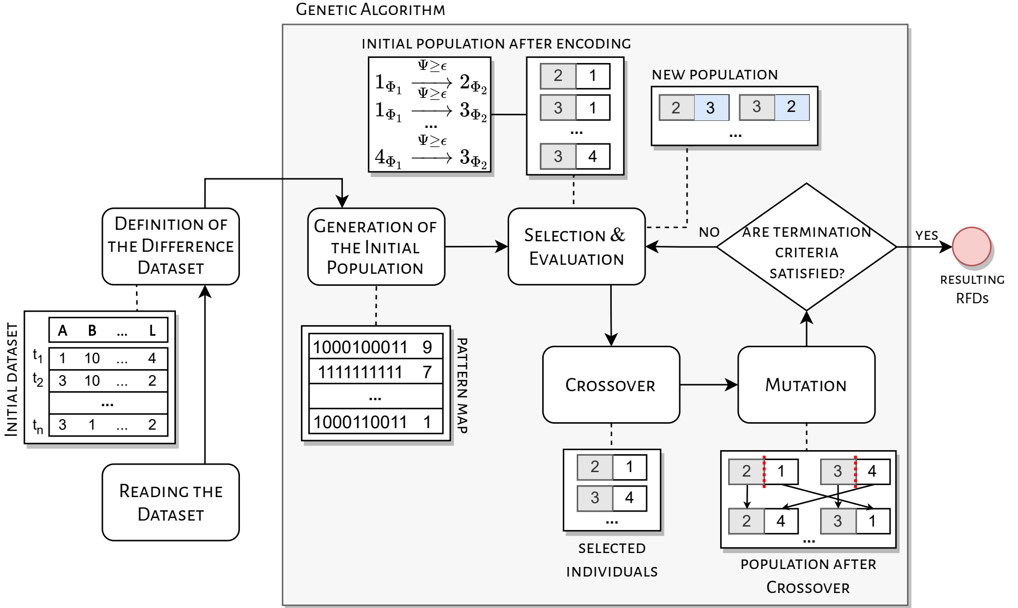
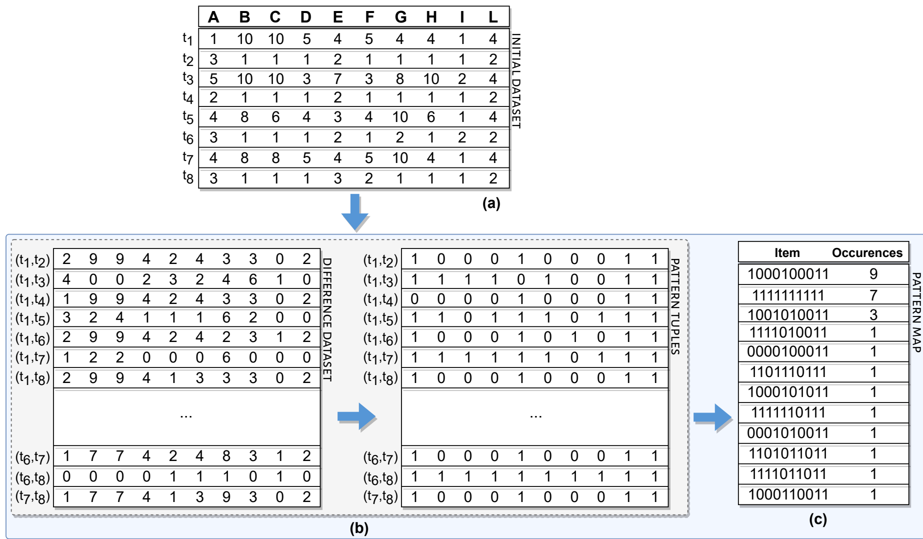


Figure 4.1: Flowchart of REDEVO.

$\mathcal{R}$, and r a relation instance of R. A pattern tuple $p_{i,j} = \{s_1, \ldots, s_m\}$ for a tuple pair (t_i, t_j) of r encodes the similarity between the attribute values of t_1 and t_2. In particular, s_k contains the value 1 if $t_i[A_k]$ is similar to $t_j[A_k]$ according to the difference constraint for the attribute A_k, considering the difference threshold provided in input, 0 otherwise.

Since the number of pattern tuples is extremely large with respect to the input dataset, we define a compression technique to remove redundancies. Thus, we defined a compressed map P, called *pattern map*, which contains each distinct pattern tuple p as *key*, and the frequency by which it occurs.
Figure 4.2 shows the process for creating a pattern map, starting from a snippet of the *Breast-Cancer* dataset (Figure 4.2a), and by considering a difference threshold equal to 2 for all attributes. In particular, we calculate the differences between the attribute values of each tuple pair, enabling the mapping into pattern tuples (Figure 4.2b). Then, we obtain the final compressed representation in terms of pattern map (Figure 4.2c).



	A	B	C	D	E	F	G	H	I	L
t_1	1	10	10	5	4	5	4	4	1	4
t_2	3	1	1	1	2	1	1	1	1	2
t_3	5	10	10	3	7	3	8	10	2	4
t_4	2	1	1	1	2	1	1	1	1	2
t_5	4	8	6	4	3	4	10	6	1	4
t_6	3	1	1	1	2	1	2	1	2	2
t_7	4	8	8	5	4	5	10	4	1	4
t_8	3	1	1	1	3	2	1	1	1	2

(t_1,t_2)	2	9	9	4	2	4	3	3	0	2
(t_1,t_3)	4	0	0	2	3	2	4	6	1	0
(t_1,t_4)	1	9	9	4	2	4	3	3	0	2
(t_1,t_5)	3	2	4	1	1	1	6	2	0	0
(t_1,t_6)	2	9	9	4	2	4	2	3	1	2
(t_1,t_7)	1	2	2	0	0	0	6	0	0	0
(t_1,t_8)	2	9	9	4	1	3	3	3	0	2
	...									
(t_6,t_7)	1	7	7	4	2	4	8	3	1	2
(t_6,t_8)	0	0	0	0	1	1	1	0	1	0
(t_7,t_8)	1	7	7	4	1	3	9	3	0	2

(t_1,t_2)	1	0	0	0	1	0	0	0	1	1
(t_1,t_3)	1	1	1	1	0	1	0	0	1	1
(t_1,t_4)	0	0	0	0	1	0	0	0	1	1
(t_1,t_5)	1	1	0	1	1	1	0	1	1	1
(t_1,t_6)	1	0	0	0	1	0	1	0	1	1
(t_1,t_7)	1	1	1	1	1	1	0	1	1	1
(t_1,t_8)	1	0	0	0	1	0	0	0	1	1
	...									
(t_6,t_7)	1	0	0	0	1	0	0	0	1	1
(t_6,t_8)	1	1	1	1	1	1	1	1	1	1
(t_7,t_8)	1	0	0	0	1	0	0	0	1	1

Item	Occurences
1000100011	9
1111111111	7
1001010011	3
1111010011	1
0000100011	1
1101110111	1
1000101011	1
1111110111	1
0001010011	1
1101011011	1
1111011011	1
1000110011	1

Figure 4.2: Creating a pattern map from a snippet of the *Breast-Cancer* dataset.

4.3.2 *Generation of the Initial Population*

In the context of Genetic Algorithms, each chromosome (a.k.a. individual) encodes a candidate solution of the optimization problem, in terms of a sequence of genes, each representing a characteristic of the solution itself. Generally, the encoding of a solution for a given problem is represented by an array of bits or a string. REDEVO uses an array V of integers to represent each individual of the population, such that each gene corresponds to the index of an attribute A_i involved into a candidate RFD. In particular, let $X_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ be a candidate RFD, then the corresponding individual contains the index of the attribute on the RHS as the first gene, followed by the indices of the attributes on the LHS. Figure 4.3 shows possible candidate RFDs defined as individuals in a population for the *Breast-Cancer* dataset, where each attribute of the dataset has been encoded by means of a unique integer value (ID). The chromosomes shown in Figure 4.3 represent the following RFDs:

$$\varphi_1 : \text{Epithelial Size}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Clump Thickness}_{\leq 2}$$

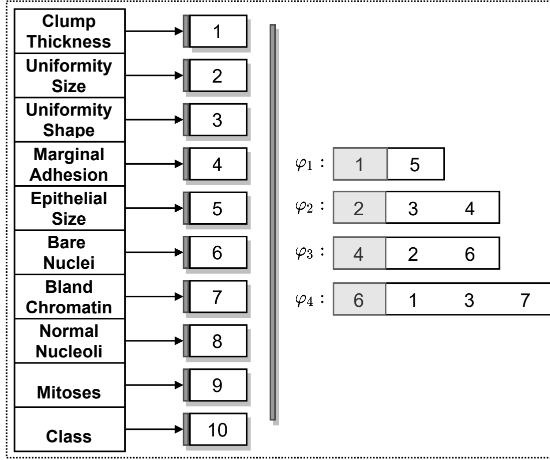


Figure 4.3: Encoding of attributes yielding the representation of candidate RFDs.

φ_2 : Uniformity Shape$_{\leq 2}$, Marginal Adhesion$_{\leq 2}$ $\xrightarrow{\Psi \geq \epsilon}$ Uniformity Size$_{\leq 2}$

φ_3 : Uniformity Size$_{\leq 2}$, Bare Nuclei$_{\leq 2}$ $\xrightarrow{\Psi \geq \epsilon}$ Marginal Adhesion$_{\leq 2}$

φ_4 : Clump Thickness$_{\leq 2}$, Uniformity Shape$_{\leq 2}$, Bland Chromatin$_{\leq 2}$ $\xrightarrow{\Psi \geq \epsilon}$ Bare Nuclei$_{\leq 2}$

According to the encoding strategy and the consequent representation of candidate RFDs as individuals, the generation of the initial population aims to define a set of individuals according to the dimensionality of the dataset under analysis. In particular, in the initial population, all individuals are defined over a specific lattice level of the search space, i.e., they contain the same number of genes. In particular, we have experimentally verified that a better convergence of the algorithm is ensured starting from the median lattice level when the number of attributes M of the pattern tuples in P is less than or equal to 15 (e.g., individuals with $\frac{M}{2}$ attributes on the LHS are generated), otherwise, the initial population is formed starting from the lattice level with three attributes for each node (e.g., individuals with 2 attributes on the LHS are generated). In both cases, since the number of possible candidate RFDs in a lattice level can be

huge, the dimension of the initial population might be reduced according to an input percentage determining the expected number of candidate RFDS (i.e., individuals) to be randomly selected.

4.3.3 *Fitness Function*

The fitness function evaluates how a given individual is close to the optimal solution, enabling genetic algorithms to converge towards an optimal solution and determine the speed of convergence. In our proposal, we exploit well-known measures in the context of association rule mining, namely *support* and *confidence* [97], to define the fitness function of REDEVO. In fact, it has been proven that such measures also permit to efficiently evaluate candidate FDS and RFDS [85].

Formally, let X be an itemset in a transaction database r, the *support* of a set X, denoted with $sup(X)$, represents the ratio of the tuples in R containing X. An *association rule* is an implication $X \rightarrow Y$, in which $X \cap Y = \emptyset$. The *support* of the association rule is the support of $X \cup Y$, which is the union of X and Y. The *confidence* of the association rule is the ratio $sup(X \cup Y)/sup(X)$. In other words, the support represents the statistic relevance of occurring patterns, whereas confidence represents the strength of implication. In order to compute support and confidence measures over the pattern map, it is necessary to introduce the compliance property of pattern tuples.

Definition 4.3.4 (Pattern tuple compliance)**.** Given a relational database schema $\mathcal{R}$ defined over a set of attributes $attr(\mathcal{R})$, $R = \{A_1, \ldots, A_m\}$ a relation schema, r a relation instance of R, $p = \{s_1, \ldots, s_m\}$ a pattern tuple, X a set of attributes, and Φ the set of difference constraints associated to X. Then, *p complies with* Φ if and only if for each attribute $A_i \in X$ $p[A_i] = 1$.

Thus, given the pattern map P computed over a database instance r, $sup(X)$ represents the ratio of tuple pairs that are similar on all attributes in X. More formally, given a pattern map P containing a collection of key-values $(< p_1, o_1 >, \ldots, < p_l, o_l >)$, where each p represents a pattern tuple and o the number of tuple pairs yielding p, and let X be a set of

attributes, with Φ the set of difference constraints associated to it, then the support of X can be defined as:

$$sup(X) = \frac{\sum_{k=1}^{l} o_k \text{ s.t. } < p_k, o_k > \text{ in } P \ \wedge \ p_k \text{ complies with } \Phi}{\sum_{k=1}^{l} o_k \text{ s.t. } < p_k, o_k > \text{ in } P} \quad (4.3)$$

Consequently, it is possible to state that a candidate RFD $\varphi : X_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ is satisfied by all tuples of a relation instance r if and only if $sup(X) = sup(X \cup A)$. Moreover, to admit the possibility that an RFD $\varphi : X_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ holds on a subset of tuples (relaxation on the extent), then we can have $sup(X) > sup(X \cup A)$. Thus, it is possible to define a coverage measure by computing the ratio between tuple pairs in r that are similar on the sets $X \cup A$ and X, respectively. This corresponds to the confidence value of φ:

$$conf(\varphi) = \frac{sup(X \cup A)}{sup(X)} \quad (4.4)$$

This confidence measure is used by REDEVO as fitness function.

Example 1. Let us consider the dataset in Figure 4.2a, and the following RFD:

$$\varphi : \text{Epithelial Size}_{\leq 2}, \text{Mitoses}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Normal Nucleoli}_{\leq 2}$$

We can calculate the associated *support* and *confidence* values as follows:

$$sup(X) = sup(\text{Epithelial Size, Mitoses}) = \frac{21}{28} \simeq 0.75$$

$$sup(X \cup A) = sup(\text{Epithelial Size, Mitoses, Normal Nucleoli}) = \frac{9}{28} \simeq 0.32$$

$$conf(\varphi) = \frac{9}{28} \cdot \frac{28}{21} = \frac{9}{21} \simeq 0.43$$

This candidate RFD will not be considered as valid by REDEVO unless the confidence measure threshold specified in input is less than or equal to 0.43.

4.3.4 *Crossover*

The evolution of the population occurs through the application of crossover and mutation operations. These are applied with specific probabilities on individuals of the population, which represent input parameters of the algorithm.

The crossover operation permits REDEVO to define a set of new candidate RFDs to be considered in the evolution step. In particular, REDEVO uses a crossover strategy that considers two candidate RFDs with the same attribute on the RHS, i.e., $\varphi_1 : W_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ and $\varphi_2 : X_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$, and constructs a new candidate RFD φ_{final}, as follows:

1) randomly selects a cut point for the LHSs of φ_1 and φ_2, i.e., W and X, which permits to split them in four new subsets of attributes, i.e., W_1, W_2, X_1, and X_2;

2) evaluates the candidate RFDs obtained by using the fitness function defined above (i.e., $\varphi_{1_1} : W_{1\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$, $\varphi_{1_2} : W_{2\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ for φ_1, and $\varphi_{2_1} : X_{1\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$, $\varphi_{2_2} : X_{2\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ for φ_2);

3) compares φ_{1_1} and φ_{1_2} (φ_{2_1} and φ_{2_2}, respectively), selecting the one having the LHS with a higher confidence value;

4) among the candidate RFDs selected in 3), adds to the population those satisfying the fitness function. On the contrary, if none of the candidate RFDs selected in 3) satisfy the fitness function, then their LHSs are combined to derive the LHS of a new RFD φ_{final} with attribute Y as RHS. Consequently, the resulting RFD φ_{final} will be added to the population. Notice that, if even one of the RFDs from which φ_{final} is derived satisfied the fitness function, it would not

make sense to add φ_{final} to the population, since it would not be minimal.

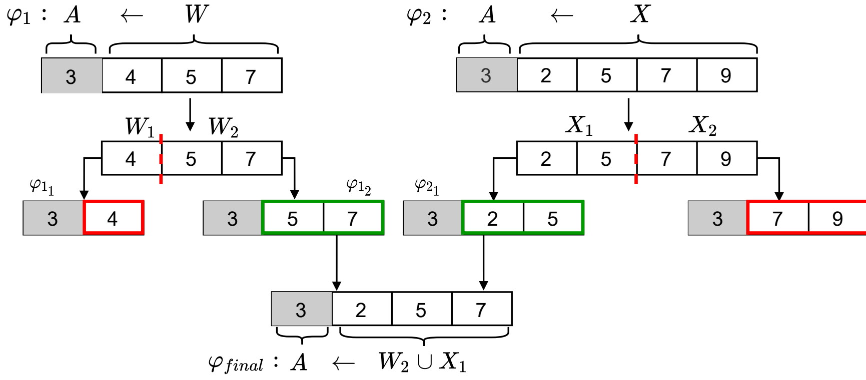


Figure 4.4: An example of crossover for candidate RFDs.

Example 2. Figure 4.4 shows an example of crossover operation from REDEVO. Let us consider the dataset in Figure 4.2a, a fitness value ε equals to 0.9, and the following two candidate RFDs:

$$\varphi_1 : \text{Marginal Adhesion}_{\leq 2}, \text{Epithelial Size}_{\leq 2}, \text{Bland Chromatin}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Uniformity Shape}_{\leq 2}$$

$$\varphi_2 : \text{Uniformity Size}_{\leq 2}, \text{Epithelial Size}_{\leq 2}, \text{Bland Chromatin}_{\leq 2}, \text{Mitoses}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Uniformity Shape}_{\leq 2}$$

The crossover function randomly selected two cut points (red lines) on the LHS attribute sets W and X, respectively, and for each subset $W_1 =$ Marginal Adhesion with ID 4, $W_2 =$ Epithelial Size$_{\leq 2}$, Bland Chromatin$_{\leq 2}$ with IDs 5 and 7, $X_1 =$ Bland Chromatin$_{\leq 2}$,Mitoses$_{\leq 2}$ with IDs 2 and 5, and $X_2 =$ Uniformity Size$_{\leq 2}$,Epithelial Size$_{\leq 2}$ with IDs 7 and 9, defines new candidate RFDs as follows:

$$\varphi_{1_1} : \text{Marginal Adhesion}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Uniformity Shape}_{\leq 2}$$

	$sup(X)$	$sup(X \cup A)$	**Fitness**
φ_{1_1}	~ 0.57	~ 0.35	0.625
φ_{1_2}	~ 0.28	~ 0.25	0.875
φ_{2_1}	~ 0.35	~ 0.28	0.88
φ_{2_2}	~ 0.32	~ 0.28	0.8

Table 4.1: Fitness values of the candidate RFDs considered in Example 2.

φ_{1_2} : Epithelial Size$_{\leq 2}$,Bland Chromatin$_{\leq 2}$ $\xrightarrow{\Psi \geq \epsilon}$ Uniformity Shape$_{\leq 2}$

φ_{2_1} : Bland Chromatin$_{\leq 2}$,Mitoses$_{\leq 2}$ $\xrightarrow{\Psi \geq \epsilon}$ Uniformity Shape$_{\leq 2}$

φ_{2_2} : Uniformity Size$_{\leq 2}$,Epithelial Size$_{\leq 2}$ $\xrightarrow{\Psi \geq \epsilon}$ Uniformity Shape$_{\leq 2}$

REDEVO selects one RFD between φ_{1_1} and φ_{1_2}, and one between φ_{2_1} and φ_{2_2}, each time selecting the one with a higher fitness value. Table 4.1 shows the fitness values of each RFD considered in this step. None of them satisfy the fitness function $\varepsilon = 0.9$. Consequently, φ_{1_2} and φ_{2_1} are combined to derive the following new candidate RFD:
φ_{final} : Uniformity Size$_{\leq 2}$,Epithelial Size$_{\leq 2}$,Bland Chromatin$_{\leq 2}$ $\xrightarrow{\Psi \geq \epsilon}$Uniformity Shape$_{\leq 2}$

4.3.5 *Mutation*

Similarly to the crossover operation, a random probability value is generated for each new candidate RFD, and only those with a probability value below the input threshold undergo the mutation step. The latter starts with a candidate RFD $\varphi : X_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ and returns a new candidate RFD, which might be mutated in some attributes. More specifically, the mutation step of REDEVO works in two different ways depending on whether the confidence value of φ is or is not greater than an input threshold:

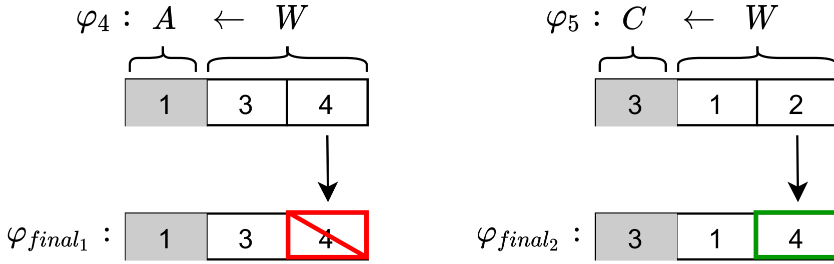


Figure 4.5: An example of mutation for two candidate RFDs.

- *Case 1*: randomly removes one attribute B from X, and returns the new RFD $\varphi_2 : X \backslash B_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ iff $|X \backslash B| \geq 1$;
- *Case 2*: randomly selects an attribute of X and replaces it with a randomly selected new attribute B, such that $B \notin X$ and $B \neq A$.

In Case 1, REDEVO generates a new candidate RFDs that is minimal with respect to the one from which is derived, whereas in Case 2 it explores a new part of the search space by considering new attributes on the LHS of the original RFD φ. In both cases, resulting RFDs will be added to the population and will be analyzed in the next iterations according to the search strategies behind each step.

Example 3. Figure 4.5 shows an example of application of the mutation operation on two candidate RFDs for the dataset in Figure 4.2a and a fitness value of 0.9:

$$\varphi_4 : \text{Uniformity Shape}_{\leq 2}, \text{Marginal Adhesion}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Clump Thickness}_{\leq 2}$$

$$\varphi_5 : \text{Clump Thickness}_{\leq 2}, \text{Uniformity Size}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Uniformity Shape}_{\leq 2}$$

The mutation function first calculates the confidence values of φ_4 and φ_5, and then it chooses the strategy to be adopted. In particular, since the confidence value of φ_4 is equal to 1, the mutation process tries to remove an attribute from the LHS of φ_4 (*Case 1*). On the other hand, since the confidence value of φ_5 is equal to ~ 0.83, the mutation function replaces an attribute from the LHS of φ_5 (*Case 2*). Thus, the resulting RFDs

φ_4 : $\text{Uniformity Shape}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Clump Thickness}_{\leq 2}$

φ_5 : $\text{Clump Thickness}_{\leq 2}, \text{Marginal Adhesion}_{\leq 2} \xrightarrow{\Psi \geq \epsilon} \text{Uniformity Shape}_{\leq 2}$

are added into the population as new individuals, in order to be analyzed in the next generation steps of REDEVO.

4.3.6 *The* REDEVO *Algorithm*

In order to explain how the discovery problem has been encoded in a genetic algorithm, in this section we describe the main procedure behind REDEVO, and the procedures of each step described in Section 4.3.1.

4.3.6.1 *Main procedure of* REDEVO

The main procedure of REDEVO is shown in Algorithm 1. It first defines the initial population (line 1) by considering the number of attributes of pattern tuples in P and the percentage of individuals to be considered in the initial population. Then, it starts the discovery process by setting the maximum number of generations to be performed to *MaxIter* (lines 3-9). For each generation, REDEVO selects the best candidate RFDs from the initial population, which are those with a fitness value greater than or equal to the object fitness value ε, together with a percentage of individuals whose fitness value is below ε (line 4). Then, among those having a randomly generated crossover probability below the input p_C value, it performs the crossover operation on a percentage of them given by the input percentage q_C of individuals to cross (line 5). The population resulting from the previous step is passed to the mutation procedure, which performs the mutation operation on the candidate RFDs having a mutation probability below p_m (line 6). At the end of each generation step, REDEVO checks whether the last populations obtained from the previous T generations returned a percentage of candidates satisfying the percentage limit defined by q_T (lines 7-8). If so, the algorithm stops the discovery process and removes from the resulting population the candidate RFDs that are not minimal with respect to others in the population

Algorithm 1 REDEVO

INPUT:
$P \rightarrow$ Similarity pattern map
$M \rightarrow$ Number of attributes of patterns tuples in P
$MaxIter \rightarrow$ Maximum number of iterations for REDEVO
$\varepsilon \rightarrow$ Fitness value objective
$p_C \rightarrow$ Crossover probability
$p_m \rightarrow$ Mutation probability
$q_I \rightarrow$ Percentage of individuals in the initial population to be considered
$q_C \rightarrow$ Percentage of individuals to cross
$q_S \rightarrow$ Percentage of individuals with lowest fitness values to extract
$q_T \rightarrow$ Percentage of stabilization of the results
$T \rightarrow$ Number of generations to be considered for the calculation of p_T

OUTPUT: `Pop` $\rightarrow$ Final Population

```
1:  Pop ← INITIALIZE(k, q_I)
2:  g ← 0
3:  while g < MaxIter do
4:      Selected_Pop ← SELECTION(Pop, ε, P, q_S)
5:      Pop ← CROSSOVER(Selected_Pop, ε, p_C, q_C, P, M)
6:      Pop ← MUTATION(Pop, ε, p_m, P, M)
7:      if TERMINATION(Pop, T, q_T) is True then
8:          break
9:      g ← g + 1
10: Final_Pop ← MINIMALITY(Pop)
11: return Final_Pop
```

(line 10). Finally, the procedure returns the minimal set of final RFDs (line 11). Further details of each step are described in following sections.

Algorithm 2 INITIALIZE

INPUT:
$M \rightarrow$ Number of attributes of pattern tuples in P
$q_I \rightarrow$ Percentage of individuals to be considered in the initial population
OUTPUT: Pop $\rightarrow$ Initial Population

1: $\text{LHS}_{card} \leftarrow 0$
2: **if** $k \leq 15$ **then**
3: $\quad \text{LHS}_{card} \leftarrow \lfloor \frac{M}{2} \rfloor$
4: **else**
5: $\quad \text{LHS}_{card} \leftarrow 2$
6: Init_Pop $\leftarrow$ COMBINE($[1, 2, \ldots, m]$, LHS_{card})
7: Pop $\leftarrow$ EXTRACT(Init_Pop, q_I)
8: **return** Pop

4.3.6.2 INITIALIZE *procedure*

The INITIALIZE procedure is shown in Algorithm 2. It starts by defining the LHS cardinalities of the individuals in the initial population. In particular, if the number of attributes M of pattern tuples in P is less than or equal to 15, the procedure starts candidate RFDs with $\frac{M}{2}$ attributes on the LHS; otherwise, it start with those having 2 attributes on the LHS (lines 1-5). This strategy allows REDEVO to consider initial candidate RFDs with an average number of attributes on the LHS for datasets with few attributes, and candidate RFDs with 2 attributes for greater datasets, preventing in both cases the possibility to have a huge initial population of candidates to evolve, which would tremendously lengthen the convergence of the algorithm. Successively, the procedure combines the attributes of pattern tuples in P in order to define a new population Init_Pop of candidate RFDs with LHS_{card} attributes on the LHS (line 6). Finally, the procedure randomly extracts some candidates from Init_Pop, according to the percentage q_I and returns the initial population (lines 7-8).

4.3.6.3 SELECTION *procedure*

The SELECTION procedure shown in Algorithm 3 creates a new population of individuals by selecting the best ones from the initial population, i.e., the candidate RFDs having a fitness value above the input fitness value (lines 3-4). Successively, for each individual φ in the initial population `Init_Pop`, if its fitness value is greater than or equal to the objective fitness value ε, then φ is added to the set `Pop_High` of best candidate RFDs (lines 5-6). Otherwise, φ is added to the set `Pop_Low` of individuals with confidence values lower than the fitness value objective (lines 7-8). At the end of the selection step, if the set of candidate RFDs with a confidence of at least ε contains few elements, then REDEVO randomly extracts a percentage q_S of candidates from `Pop_Low` (lines 9-11). This strategy allows REDEVO to consider more candidate RFDs in the search space, increasing the possibility that some candidates with higher confidence values may be generated by individuals with lower confidence values. It is important to notice that a high percentage of individuals with lower confidence values could lead to a slower convergence of REDEVO. Thus, a suitable percentage value q_S should be set in the configuration step. At the end of procedure SELECTION, a new population `Pop` of individuals is returned (line 12).

4.3.6.4 CROSSOVER *procedure*

The CROSSOVER procedure shown in Algorithm 4 creates a new population of individuals by crossing individual pairs. The procedure starts by considering, for each attribute, the set of individuals from the initial population `Init_Pop` with the same attribute on the RHS, and randomly extracting some of them, according to the percentage q_C of individuals to cross (line 4). Then, the procedure checks if the set of selected candidate RFDs to cross contains at least one pair of individuals (line 5). If so, it is possible to perform a crossover operation according to the strategy defined in Section 4.3.4 (lines 6-34). More in detail, the procedure first verifies the possibility to cross between pairs of individuals randomly selected among those sharing the same RHS, by verifying whether their

Algorithm 3 SELECTION

INPUT:
`Init_Pop` $\rightarrow$ Population of individuals to be selected
$\varepsilon \rightarrow$ Fitness value
$P \rightarrow$ Compressed version of difference dataset
$q_S \rightarrow$ Percentage of individuals with lowest fitness values to extract
OUTPUT: `Pop` $\rightarrow$ Population of selected individuals

1: `Pop_High` $\leftarrow \emptyset$
2: `Pop_Low` $\leftarrow \emptyset$
3: **for each** $\varphi \in$ `Init_Pop` **do**
4: $\quad \varepsilon_1 \leftarrow$ EVALUATE_FITNESS(φ, P)
5: $\quad$ **if** $\varepsilon_1 \geq \varepsilon$ **then**
6: $\quad\quad$ `Pop_High` $\leftarrow$ `Pop_High` $\cup \{\varphi\}$
7: $\quad$ **else**
8: $\quad\quad$ `Pop_Low` $\leftarrow$ `Pop_Low` $\cup \{\varphi\}$
9: `Pop` $\leftarrow$ `Pop_High`
10: **if** |`Pop`| ≤ 2 **then**
11: $\quad$ `Pop` $\leftarrow$ `Pop` $\cup$ EXTRACT(`Pop_Low`, q_S)
12: **return** `Pop`

randomly generated number is not greater than the crossover probability p_C (line 9-10). For each of them, the procedure defines a random cut point among the attributes on the LHS (lines 11-12). Then, the procedure defines four new individuals, i.e., φ_{1_1}, φ_{1_2}, φ_{2_1}, and φ_{2_2}, according to the approach defined in Section 4.3.4, comparing φ_{1_1} with φ_{1_2}, and φ_{2_1} with φ_{2_2}, selecting for each pair the candidate RFDs with higher confidence value, which will be used to define new individuals. In particular, each φ_{i_j} with $i, j \in \{1, 2\}$ is added to the new population (lines 23-26) iff it has a confidence value greater than or equal to the fitness value objective ε. On the contrary, if none of such four individuals can be added, then the procedure constructs the new individual φ_{final} according to the approach defined in Section 4.3.4, and adds it to the new population (lines 27-32). At the end of the CROSSOVER, the procedure checks the minimality between

Algorithm 4 CROSSOVER

```
INPUT:
    Init_Pop → Initial population
    ε → Fitness value
    p_C → Crossover probability
    q_C → Percentage of individuals to cross
    P → Compressed version of difference dataset
    M → Number of attributes of pattern tuples in P
OUTPUT: Pop → Final Population
1: RHS ← 1
2: Final_Pop ← ∅
3: while RHS ≤ m do
4:   CHRs_TO_CROSS ← EXTRACT(Init_Pop.get(RHS), q_C)
5:   if |CHRs_TO_CROSS| ≥ 2 then
6:     i ← 0
7:     for i < ⌊|CHRs_TO_CROSS| / 2⌋ do
8:       P ← randomValue(0,1)
9:       if P ≤ p_C then
10:        φ_1, φ_2 ← randomSelection(CHRs_TO_CROSS)
11:        C_{p_1} ← randomValue(1, |φ_1|)
12:        C_{p_2} ← randomValue(1, |φ_2|)
13:        φ_{1_1} ← RHS ∪ {φ_1[1 : C_{p_1}]}
14:        φ_{1_2} ← RHS ∪ {φ_1[C_{p_1} + 1 : |φ_1| − 1]}
15:        ε_{1_1} ← EVALUATE_FITNESS(φ_{1_1}, P)
16:        ε_{1_2} ← EVALUATE_FITNESS(φ_{1_2}, P)
17:        φ_{2_1} ← RHS ∪ {φ_2[1 : C_{p_2}]}
18:        φ_{2_2} ← RHS ∪ {φ_2[C_{p_2} + 1 : |φ_2| − 1]}
19:        ε_{2_1} ← EVALUATE_FITNESS(φ_{2_1}, P)
20:        ε_{2_2} ← EVALUATE_FITNESS(φ_{2_2}, P)
21:        φ_final ← RHS                      ▷ Creation of the new chromosome
22:        Pop ← ∅
23:        if ε_{1_1} ≥ ε then Pop ← Pop ∪ {φ_{1_1}} end if
24:        if ε_{1_2} ≥ ε then Pop ← Pop ∪ {φ_{1_2}} end if
25:        if ε_{2_1} ≥ ε then Pop ← Pop ∪ {φ_{2_1}} end if
26:        if ε_{2_2} ≥ ε then Pop ← Pop ∪ {φ_{2_2}} end if
27:        if Pop is empty then
28:          if ε_{1_1} > ε_{1_2} then φ_final ← φ_final ∪ {φ_{1_1} \ RHS}
29:          else φ_final ← φ_final ∪ {φ_{1_2} \ RHS}
30:          if ε_{2_1} > ε_{2_2} then φ_final ← φ_final ∪ {φ_{2_1} \ RHS}
31:          else φ_final ← φ_final ∪ {φ_{2_2} \ RHS}
32:          Pop ← Pop ∪ {φ_final}
33:        Final_Pop ← Final_Pop ∪ {Pop}
34:      i ← i + 1
35:   RHS ← RHS + 1
36: Final_Pop ← MINIMALITY(Final_Pop)
37: return Final_Pop
```

the candidate RFDs in `Final_Pop` and it returns the new population (lines 36-37).

4.3.6.5 Mutation *procedure*

The MUTATION procedure shown in Algorithm 5 creates a new population of individuals by mutating a gene in some individuals. For each individual into the initial population, the procedure verifies the possibility to perform a mutation on it, according to the mutation probability p_m (lines 3-4). For those having a randomly generated value not greater than p_m, the procedure evaluates their fitness value of a candidate RFD φ (line 5) in order to define the mutation strategy to be performed, according to the approach defined in Section 4.3.5. More in detail, if their fitness value of φ is greater than the objective fitness value ε, then the procedure randomly removes an attribute from their LHS (lines 7-10). Otherwise, the procedure checks if there exists at least one attribute that is not already considered in their LHS (line 12). In this case, the procedure tries to randomly select a new attribute to replace a randomly selected attribute from the LHS (lines 13-17). The so derived candidate RFD φ will be added to the final population at the end of each iteration (line 21). Finally, the procedure returns the new population of individuals resulting from the mutation step (line 19).

4.3.6.6 Termination *procedure*

The TERMINATION procedure shown in Algorithm 6 allows REDEVO to monitor the number of individuals of each generation step for verifying if it remains stable for multiple generation steps. In particular, given a number of generations T, this procedure allows REDEVO to automatically stop the discovery process if there exist at least T populations with a similar number of individuals. Thus, the TERMINATION procedure starts by including the size of the new population together with the already stored ones in a set E (lines 1-2). Then, it checks if there are at least T populations already stored in E (line 3), and if so the procedure calculates the ratio between the `sum` of the sizes of the population in E and the

Algorithm 5 MUTATION

```
INPUT:
    Init_Pop → Initial population
    ε → Fitness value
    p_m → Mutation probability
    P → Compressed version of difference dataset
    M → Number of attributes of P
OUTPUT: Pop → Final Population
 1: Pop ← ∅
 2: for each φ ∈ Init_Pop do
 3:   P ← randomValue(0,1)
 4:   if P ≤ p_m then
 5:     ε1 ← EVALUATE_FITNESS(φ,P)
 6:     RHS ← φ[0]
 7:     if ε1 > ε then
 8:       ▷ Remove an attribute from the LHS
 9:       C_p ← randomValue(1,|φ|)
10:       φ ← {RHS} ∪ {φ[1 : C_p]} ∪ {φ[C_p + 2 : |φ|]}
11:     else
12:       if |φ| ≠ m then
13:         i ← randomValue(1, |φ|)
14:         New_Gene ← φ[i]
15:         while New_Gene in φ do
16:           New_Gene ← randomValue(1,|φ|)
17:         φ[i] ← New_Gene
18:   Pop ← Pop ∪ {φ}
19: return Pop
```

maximum size of the populations added to E in the latest T generation steps (lines 4-7). If the resulting threshold t_E is greater than or equal to the percentage of stabilization q_T provided in input, the procedure returns *True*, leading REDEVO to stop its execution (lines 8-9); otherwise, REDEVO continues the discovery process (line 10) (see Algorithm 1).

Algorithm 6 TERMINATION

INPUT:
Pop → Resulting population from one generation
T → Number of generations to be considered for the calculation of p_T
q_T → Percentage of stabilization of the results
OUTPUT:
True → If the termination criterion is met
False → Otherwise

```
1:  E ← GET_LAST_EXECUTIONS(T − 1)
2:  E ← E ∪ |Pop|
3:  if |E| == T then
4:      sum ← 0
5:      for each e ∈ E[:T] do
6:          sum ← sum + e
7:      t_E ← sum / (MAX(E) · T)
8:      if t_E ≥ q_T then
9:          return True
10: return False
```

4.3.7 *Experimental Evaluation*

In this section, we show the results obtained from the execution of REDEVO on several real-world datasets, by varying its configuration parameters. We also present a comparative evaluation of the execution performances of REDEVO with the only existing algorithm for discovering hybrid RFDS DIMε [37].

All the experiments have been executed on an iMac Pro with an Intel Xeon CPU at 3.20GHz, 18-core, and 128GB of memory, running macOS Mojave 6.4 and Python 3.9. In particular, to compute difference values, we used the absolute difference for numerical attributes, and the Levenshtein distance for textual attributes [165]. Moreover, in order to make REDEVO comparable to other discovery algorithms, we used the g3-error coverage measure for relaxing the extent criterion, which corresponds to $1 -$

Dataset	Cols [#]	Rows [#]	FDs [#]	Size [KB]
Iris	5	150	5	5
Balance-Scale	5	625	1	7
Abalone	9	4177	137	192
Breast-cancer	11	699	46	21
Bridges	13	108	142	7
Echocardiogram	13	132	538	7
Fd-reduced	15	8000	1122	109
Lymphography	19	148	2730	6
Parkinsons	24	195	1724	40
Ionosphere	34	351	1122	149
Sonar	60	208	97750	86
Movement-Libras	91	360	2473105	251

Table 4.2: Characteristics of the considered real-world datasets.

$conf(X \rightarrow A)$, with $X_{\Phi_1} \xrightarrow{\Psi \geq \epsilon} A_{\phi_2}$ candidate RFD. All the considered real-world datasets have been previously used for evaluating other RFD discovery algorithms, whose characteristics are shown in Table 4.2.

4.3.7.1 *Discovery Performances*

Our first experiment measured the execution times and the number of RFDs discovered by REDEVO on the different real-world datasets. The latter have been mapped to a pattern map, according to the strategy described in Section 4.3.1.1, by varying difference thresholds from 0 to 9 (named $C0, \ldots, C9$, resp.), and the g3-error thresholds from 0.0 to 0.9 (named $E0.0, \ldots, E0.9$, resp.). Other configuration settings have been fixed to 500 for the maximum number of iterations admitted, 0.4 for the probability of both crossover and mutation, and 10% for the percentage of individuals to be considered in the initial population (see Section 4.3.6.2).

Analysis of Results. Figure 4.6 summarizes the execution times of REDEVO for each dataset in terms of line plots, according to the considered difference thresholds, whereby each line represents one of the considered g3-error thresholds.

As we can notice, the execution times are almost always less than 20 seconds, except for some of the biggest datasets. In fact, as expected, when the number of columns is high also the execution times increase, as occurred for the *Sonar* and the *Movement-libras* datasets. The execution times resulting from the *Lymphography* dataset show unexpected peaks when the difference threshold is set to 0 (i.e., $C0$). This could be due to the fact that the discovered RFDS almost always present a high number of attributes on the LHS, yielding a difficult convergence to the optimal solution. Moreover, among the different extent thresholds, we can notice that lower error thresholds require higher execution times since they constraint the discovery process to manage a higher number of possible invalidations than when the error threshold is higher.

Concerning the number of RFDS (see Figure 4.7), we can notice different trends according to the size of the datasets. In particular, datasets with a high number of attributes registered a similar number of discovered RFDS among the different g3-error thresholds. More specifically, *Sonar* and *Movement-libras* datasets often provide a higher number of RFDS with g3-error threshold equal to 0. This could be due to the fact that by admitting errors in the validation of candidate RFDS through increased g3-error thresholds, the discovery process could extract many valid RFDS that are minimal with respect to the ones discovered with a threshold equal to 0, yielding a lower number of discovered RFDS. On the contrary, by considering datasets with a lower number of attributes, as we expected, a higher extent threshold potentially increases the number of valid RFDS. Obviously, some exceptions can be noticed, but these are included within a small range of variability, typically related to the nature of the dataset itself, as occurred for *Breast-Cancer* and *Balance-Scale*. Instead, concerning the variation of tuple comparison thresholds, in most cases the number of discovered RFDS drastically drops when increasing thresholds from 0 to 1. This is mainly due to the presence of many key dependencies, which are likely to be invalidated when the tuple comparison thresholds become greater than 0 (e.g., *Abalone*, *Echocardiogram*, *Lymphography*, *Movement-Libras*, *Parkinsons*, and *Sonar* datasets). On the other hand, this trend does not appear for datasets containing a small set of key dependencies.

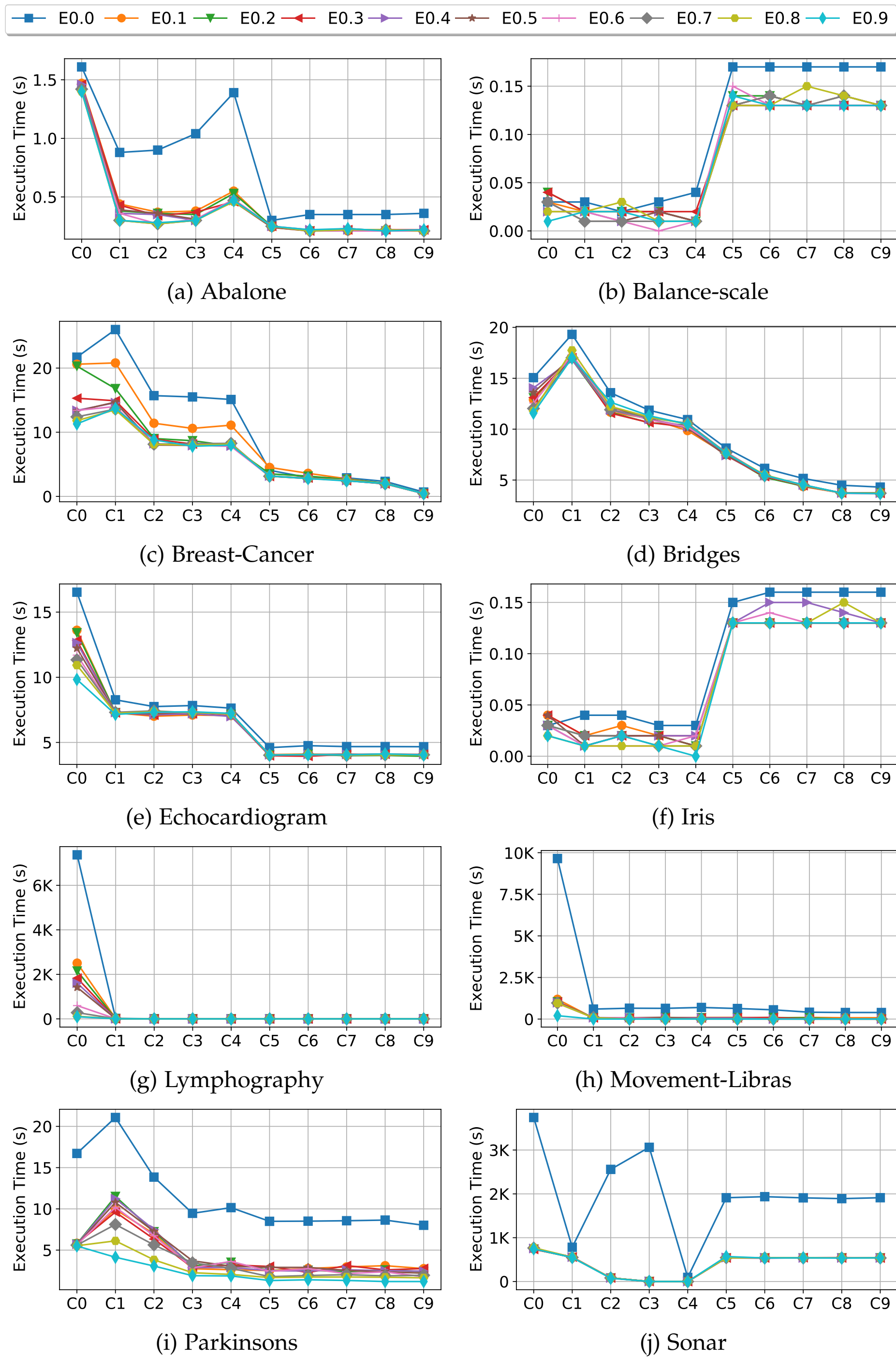


Figure 4.6: Time performances by varying tuple comparison and extent thresholds.

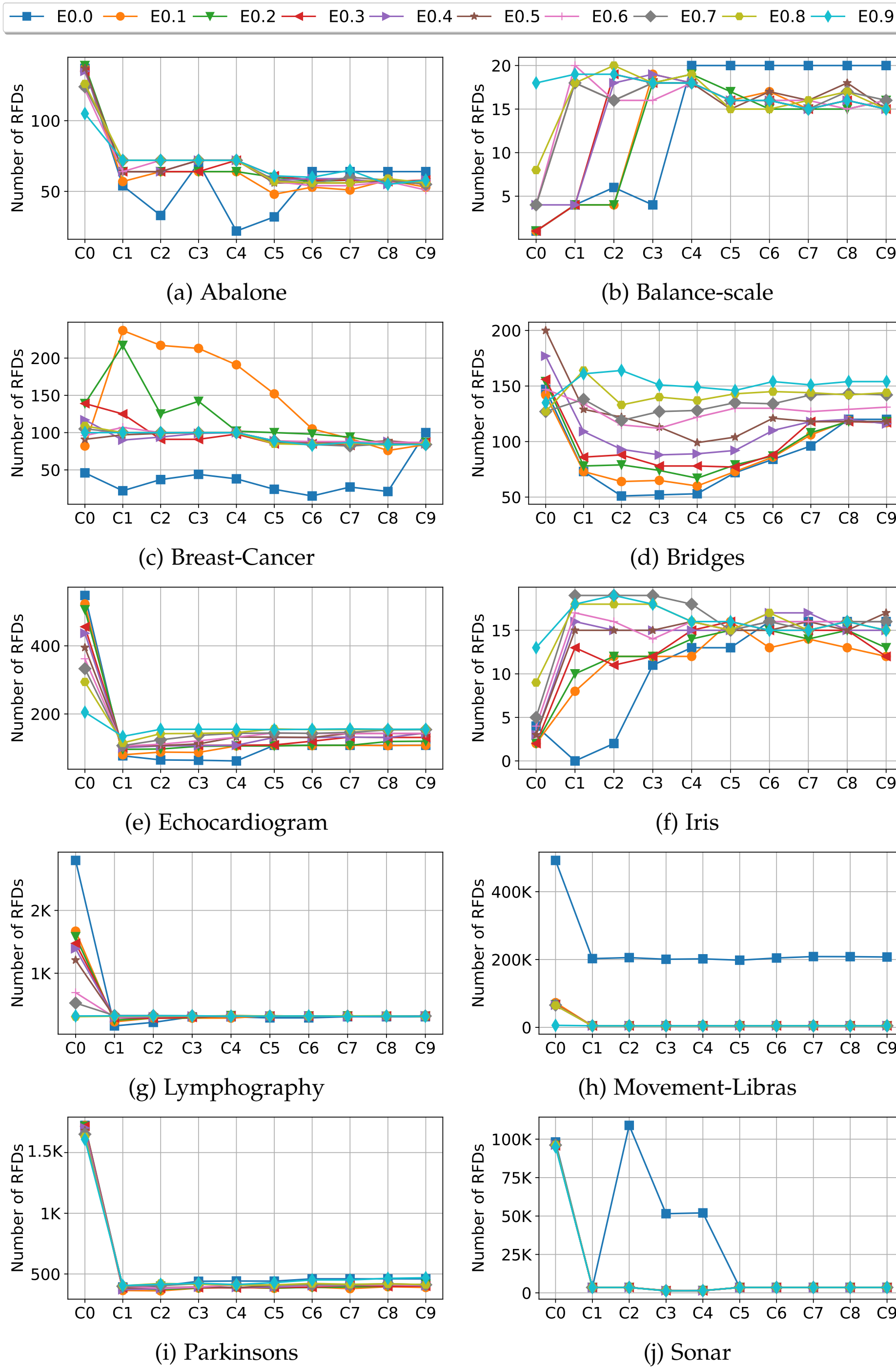


Figure 4.7: Number of RFDs by varying tuple comparison and extent thresholds.

4.3.8 *Evaluation on configuration settings*

In this section, we show the performances of REDEVO in terms of execution times and number of discovered RFDs, by evaluating the impact of evolutionary configuration settings (see Figure 4.8). In particular, we evaluated the performances of REDEVO on the Ionosphere dataset (characterized by 34 attributes and 351 tuples), and by considering the following baseline configuration: 10% as the size of the initial population, 100 as the number of iterations, and 0.4 as the probability of both crossover and mutation operations. Therefore, we varied one of the above-described configuration parameters, yielding four different experimental sessions by considering: 1) the variation in the size of the initial population; 2) the variation in the number of maximum iterations; 3) the variation in the crossover probability; and 4) the variation in the mutation probability. Furthermore, in order to analyze how REDEVO performs independently from a specific generation of individuals, for each experimental session we run it five times and compared the average results based on the execution times and the number of discovered RFDs. In particular, in Figure 4.8 we report the average values by means of lines, by also showing the variability range of execution times.

The size of the initial population is computed in terms of percentage on all possible candidate RFDs in a lattice level, according to the strategy defined in Section 4.3.6.2. As expected, Figure 4.8a shows that the execution times increase when the size of the initial population grows. However, we can notice that after the value 20% in the size of the population, the number of discovered RFDs remains stable. This means that it is not useful to consider high percentages of initial populations, since it will not improve the capability of the evolutionary strategy to converge towards holding RFDs.

Concerning the variation in the number of iterations (see Figure 4.8b), we can notice that the execution times show an increasing trend by varying the number of iterations. Nevertheless, this is not related to the number of discovered RFDs, since the number is quite similar, i.e., ranging from 705 to 720, across the variation of the maximum number of iterations, and follows a non-monotonic trend.

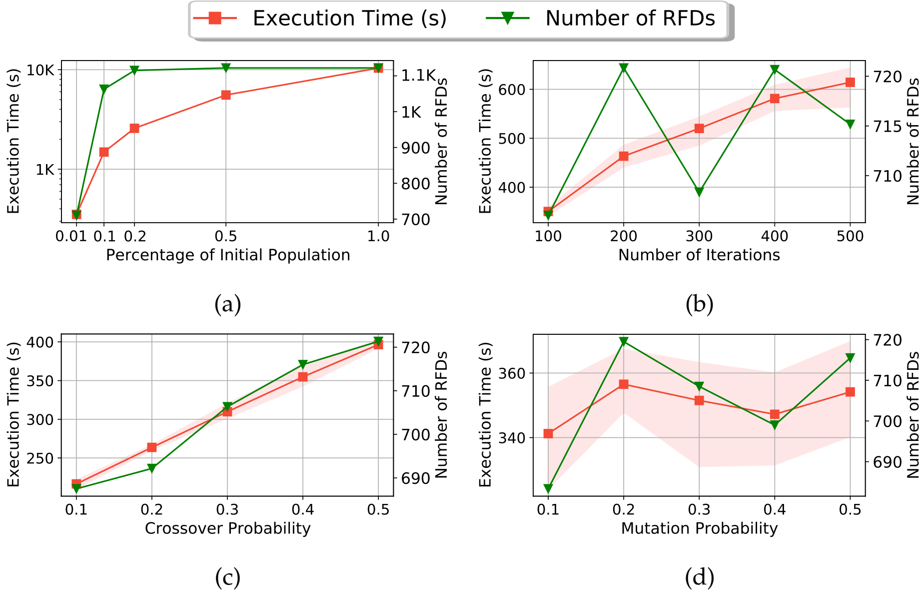


Figure 4.8: Variation of genetic configuration parameters.

Figures 4.8c and 4.8d show the results obtained by varying the probability of crossover and mutation operations, respectively. In particular, both the execution times and the number of RFDs generally grow when the probability of crossover increases. Instead, the execution times concerning the variation on the mutation probability registered two peaks, i.e., for 0.2 and 0.5, which seems to be related to the number of discovered RFDs.

4.3.8.1 *Comparative Evaluation*

In this section, we show the results of a comparative evaluation between REDEVO and the only existing discovery algorithm for hybrid RFDs proposed in the literature, i.e., DIME [37]. The latter is one of the first proposals for discovering RFDs capable of validating them according to the differences between tuple pairs. In particular, we evaluated the performances of both algorithms on the Fd-Reduced dataset (characterized by 15 attributes and $8K$ tuples), by accomplishing two different types

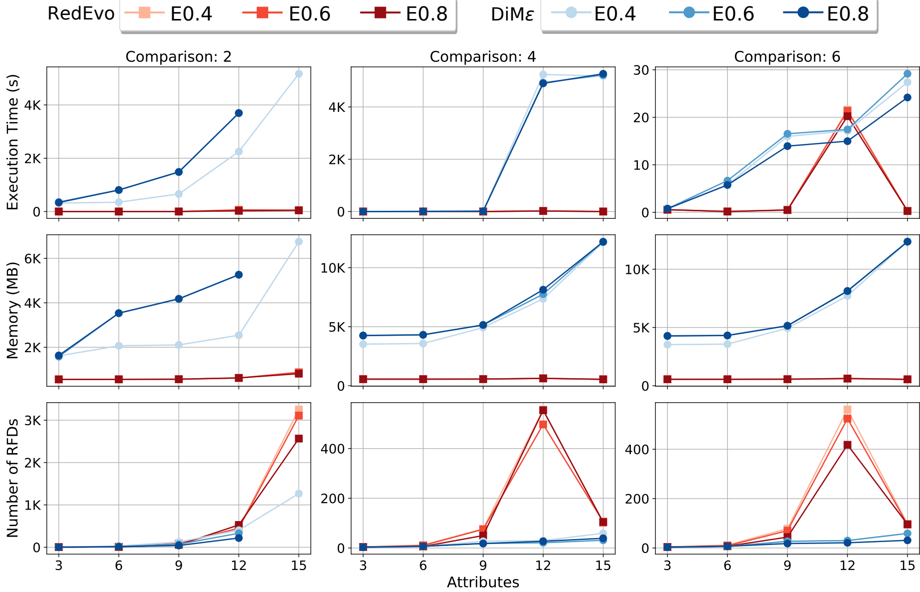


Figure 4.9: Comparative evaluation on Fd-Reduced dataset by varying the number of attributes.

of experimental sessions: 1) by varying the number of attributes in the range $[3 - 15]$ with step 3, and 2) by varying the number of tuples in the range $[2K - 8K]$ with step $2K$. Moreover, for each experimental session, we varied the extent thresholds in the range $0.4 - 0.8$ with step 0.2, and the tuple comparison thresholds in the range $[2 - 6]$ with step 2. Finally, we set a time limit of 2 hours, after which the algorithm execution is stopped.

Figures 4.9 and 4.10 show the obtained results, in which the red and the blue lines represent the performances of REDEVO and DiMε algorithms, respectively. More specifically, Figure 4.9 shows performances of both algorithms in terms of execution times, memory requirements, and the number of discovered RFDs, which are obtained by varying the number of attributes. As we can see, the execution times of REDEVO are always lower than DiMε when the number of attributes increases, except with a

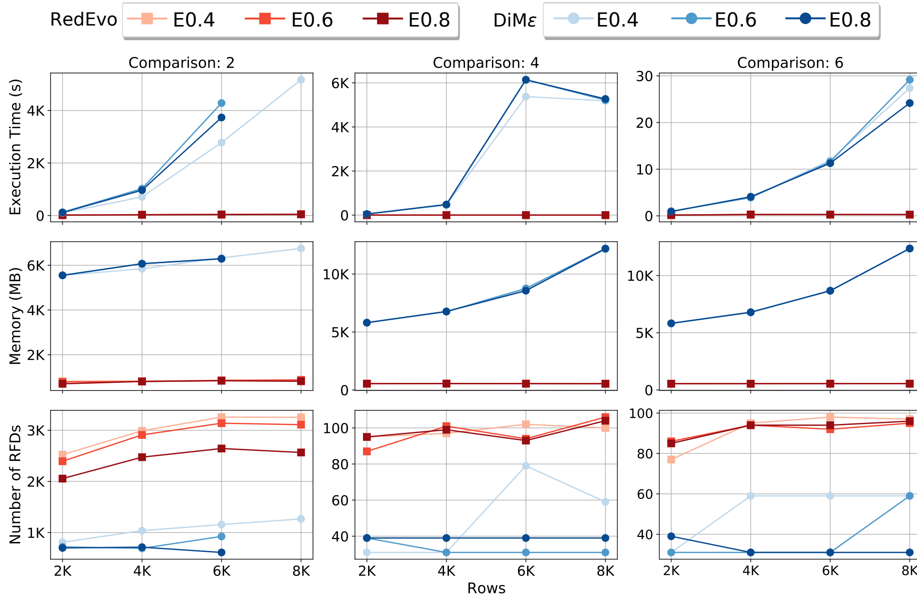


Figure 4.10: Comparative evaluation on Fd-Reduced dataset by varying the number of tuples.

comparison threshold equal to 6 for the dataset containing 12 attributes. This exception is probably due to the higher number of discovered RFDs, even if the difference in execution times is quite low, i.e., maximum 5 seconds worse for REDEVO. In general, the higher execution times of DiMε are probably due to the deterministic nature of DiMε, which yields an exponential growth of possible candidate RFDs to be evaluated when the number of attributes increases. In fact, it is possible to notice that some DiMε execution performances exceed the time limit defined in the configuration settings. On the contrary, the execution times of REDEVO tend to remain stable in almost all executions and to slightly grow when the number of attributes increases. Concerning the extent thresholds, we can notice that both DiMε and REDEVO achieve similar performances for all thresholds, except for performances of DiMε executions with comparison threshold equals to 2, where a lower extent threshold registered better performances on both execution times and memory load.

In general, concerning the memory load, DiMε requires the use of much more memory than REDEVO, probably due to compressed representations of similarities in terms of pattern map introduced in REDEVO, and the exploitation of several caches that avoid the re-computation of support and confidence values, so reducing the waste of temporary memory. Additionally, the memory requirements of REDEVO remain stable even when the number of discovered RFDs increases. In fact, the high dimensionality of a dataset causes a growth in the number of discovered RFDs, regardless of extent and comparison thresholds.

Figure 4.10 shows performances of both algorithms in terms of execution times, memory requirements, and the number of discovered RFDs, which are obtained by varying the number of tuples. Similarly to the previous experimental session (see Figure 4.9), REDEVO achieves better performances in terms of execution times and memory requirements than DiMε also when varying the number of tuples. In particular, the execution times of REDEVO are extremely low for all comparison and extent thresholds, regardless of the number of considered tuples, and the gap grows when the number of tuples increases. Also in this case, DiMε exceeded the time limit on the dataset with the highest number of tuples, a comparison threshold equal to 2, and extent thresholds greater than 0.4. Instead, execution times and memory requirements remain stable for REDEVO, since the latter exploits the pattern map, which permits to aggregate similar tuples according to the strategy defined in Section 4.3.1.1. In fact, this strategy allows to speed up the computation of support and confidence values for validating RFDs.

Concerning the number of RFDs, this experimental session highlights the possible differences in the result sets, due to the different natures of the two compared algorithms. In fact, as expected, being REDEVO an evolutionary algorithm, it could not guarantee the minimality of all the RFDs it discovers. Nevertheless, the RFDs discovered by REDEVO are always correct, but for a minimal RFD discovered by DiMε could be several RFDs discovered by REDEVO generalizing it.

5 DISCOVERY ALGORITHMS IN DYNAMIC SCENARIOS

The discovery of functional dependencies (FDs) and relaxed functional dependencies (RFDs) from data is facing novel challenges, among which there is the necessity of monitoring sets of data that are produced and updated over time. For instance, the proliferation of IoT sensors and technologies is leading to an increasing need to define methodologies to continuously discover metadata from data streams, which represent a kind of dynamic datasets. In this context, incremental FD (RFD) discovery algorithms have to efficiently verify which of the previously discovered FDs (RFDs) still hold on upon the updates on the dataset, and also infer possible new valid FDs (RFDs). In this chapter, we first formalize the issues related to the problem of discovering FDs (RFDs) in dynamic scenarios, by also discussing differences with static discovery processes, and then we describe several novel search strategies and validation methods composing new incremental (continuous) discovery algorithms for FDs (RFDs). In particular, the first algorithm addresses the problem of discovering FDs in incremental scenarios without having to restart the discovery process from scratch, when new data are inserted. It represents the baseline of the second algorithm, which offers a novel and efficient validation method relying on Regular Expressions (RegExs). The last two algorithms deal with two specific problems more complex than the incremental discovery of FDs, i.e., continuous discovery of FDs and incremental discovery of RFD_es, respectively.

5.1 PROBLEM DESCRIPTION

This section presents the fundamentals of incremental discovery methodologies for FDs and RFDs, by also describing the data representation, search strategies, and properties behind them. Moreover, the theoretical

evaluation of the minimality and completeness of a discovery algorithm in an incremental scenario is provided in Section 5.6.3. Other methodologies presented in this thesis exploits well-known proofs concerning validation and pruning properties that can be found in [84, 163].

5.1.1 *Incremental Discovery of* FDs

The FD discovery problem aims at finding the set of all *minimal* FDs holding on an instance r of a database schema R. It entails searching FDs whose LHSs (RHSs) yielding tuple partitions of data sharing the same values on RHS attributes whenever they share the same values on LHS ones [37, 85]. Most of the FD discovery algorithms defined in the literature [140] operate on static sets of data, and require to be re-executed from scratch whenever the data are updated.

In general, the discovery of FDs is a computationally expensive process, especially for database instances with a large number of rows and columns. In particular, to perform the discovery process, it is possible to first generate FD candidates, and then verify their validity and minimality, yielding a column-based strategy. More specifically, column-based strategies model the search space as a graph representation of a lattice, which contains a collection of attribute sets, where level zero maps the empty set level one singleton sets (e.g., one for each attribute), level two the pair sets (e.g., 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 attributes from R. It permits to consider a specific attribute set Z at a level l, and then to formulate all the possible FDs $X \rightarrow A$, with $X = Z \backslash \{A\}$ and $A \in Z$, to be successively validated. Moreover, the validation relies on the tuple partitions of attributes in X and Z, and verifies the satisfiability of a specific property on the cardinalities of partitions, namely refinement property [85]. This validation process is particularly useful when it is possible to follow a level-by-level search strategy from level one to level M, and to gradually construct partitions over the ever-increasing attribute set sizes. This could not be guaranteed when the discovery process has to work in dynamic scenarios. In fact, one of the main problems with the

dynamic updating of database instances is that FDs found at a specific time instant τ could be invalidated at the time $\tau + 1$. To this end, it is necessary to implement incremental methodologies that consider FDs holding at time τ as starting points for (re-)validating previously holding FDs and searching possible new FDs holding at $\tau + 1$.

In other words, dynamic scenarios require to update a set of minimal FDs Σ_τ discovered over data collected until time τ according to the set of tuple updates $D_{\tau+1}$ at time $\tau + 1$. This yields the necessity to (re)-consider all FDs in Σ_τ, and possibly generate new candidate FDs according to validation results. For instance, the insertion of a new tuple can yield the *invalidation* of a previously holding FD, or *the confirmation of its minimality*, whereas the deletion of a tuple can yield the loss of *minimality* for a previously holding FD. Notice that, updates of tuple values can always be managed as a deletion and a consequent insertion of a tuple containing the modified values [140]. More formally, let $X \rightarrow A$ be a minimal FD holding at time τ on an instance r of R, with $X, A \subseteq attr(R)$, we need to consider the following possible effects whenever the instance is updated.

- **Invalidation of functional dependencies**. If $X \rightarrow A$ holds at time τ, then for each tuple pair (t_1, t_2) of r $t_1[A] = t_2[A]$ whenever $t_1[X] = t_2[X]$. Thus, $X \rightarrow A$ is invalidated at time $\tau + 1$ if and only if at least one of the following two cases occur:
  - A tuple t_3 is inserted at time $\tau + 1$, and
  $$t_1[X] = t_3[X] \wedge t_1[A] \neq t_3[A]$$
  - Tuples t_4 and t_5 are inserted at time $\tau + 1$, and
  $$t_4[X] = t_5[X] \wedge t_4[A] \neq t_5[A]$$
- **Refutation of minimality**. If $X \rightarrow A$ is minimal at time τ, then any $X \setminus B \rightarrow A$ with $B \in X$ is not valid at time τ. In other words, there exists at least one tuple pair (t_1, t_2) such that $t_1[X \setminus B] = t_2[X \setminus B] \wedge t_1[A] \neq t_2[A]$. Let us suppose that there is only one tuple pair (t_1, t_2) invalidating $X \setminus B \rightarrow A$ at time τ, and that t_1 is deleted

from r at time $\tau + 1$, denoted as t_1^-. Consequently, $X \to A$ is no longer minimal, since $X \setminus B \to A$ is valid at time $\tau + 1$.

Thus, if a minimal FD $X \to A$ holding on r at time τ is invalidated at time $\tau + 1$, then new candidate FDs need to be analysed, i.e., it is necessary to check the validation of all non-trivial FDs $XB \to A$ for each $B \in attr(R) \setminus (X \cup A)$. This means that a higher level in the lattice search strategy is considered. In fact, only candidates of higher levels with respect to the holding FDs at time τ can hold at time $\tau + 1$. Moreover, it is not necessary to analyze candidate FDs like $X \setminus B \to A$, with $B \in X$, since such FDs were not valid at time τ, and could not be valid at time $\tau + 1$. On the contrary, if a minimal FD $X \to A$ at time τ is no longer minimal at time $\tau + 1$, then new candidate FDs like $X \setminus B \to A$, with $B \in X$, need to be checked. Moreover, it is not necessary to check FDs like $XB \to A$, with $B \in attr(R) \setminus (X \cup A)$, since they were not minimal at time τ, and cannot be minimal at time $\tau + 1$, given that $X \to A$ is not minimal.

For each FD that becomes valid or potentially minimal at time $\tau + 1$, it is necessary to check its minimality compared with candidate FDs that can be minimal with respect to it, that is FDs generalizing it. More formally, let $X \to A$ be an FD valid on an instance r of R at time $\tau + 1$, with $X, A \subseteq attr(R)$, we need to consider the following possible effect whenever a candidate FD becomes valid or potentially minimal at time $\tau + 1$.

- **Inference.** Let $X \to A$ be an FD becoming valid at time $\tau + 1$ on an instance r of a database schema R, has become valid at time $\tau + 1$, it is necessary to check that no $X \setminus Z \to A$, with $Z \subset X$ is valid at time $\tau + 1$.

In other words, if an FD $X \to A$ becomes valid or potentially minimal at time $\tau + 1$, then all possible candidate FDs $X \setminus B \to A$, with $B \in X$, need to be analysed to verify that no $X \setminus B \to A$ is valid at time $\tau + 1$. On the contrary, if at least one $X \setminus B \to A$ is valid, this becomes the new potentially minimal FD to be checked.

Invalidation, refutation, and inference allow the incremental search strategy to determine how candidate FDs should be generated and man-

aged throughout the search space, according to previously holding FDs and validation results. This entails moving the focus on different parts of the search space, by promptly considering ever changing data partitions for validating new candidate FDs.

5.1.2 *Continuous Discovery of* FDs *from dynamic sources*

Nowadays there are many data stream sources whose data are generated from traffic or health sensors, transaction logs, and so on. Typically, such sources keep sending data in extremely short time intervals, creating a continuous stream of data that must be rapidly processed, without the possibility to entirely store them, and with no control over the order in which data elements arrive [72]. For this reason, there is an increasing necessity to dynamically extract information from streams of data originating from sensors and other devices. To this end, several research communities are trying to develop efficient methodologies to continuously monitor the quality of the data read from data streams [52, 66, 91]. Such methodologies need to handle real-time analysis processes, without creating information queues, and guaranteeing information integrity in order to avoid data loss on the stream. Furthermore, each model must view a data stream as a sequence of single tuples to simplify the information management.

FDs represent fundamental metadata to detect quality problems in dynamic data sources, such as data streams. However, techniques using this type of metadata require methodologies to automatically discover them. As introduced in Section 2, discovering FDs from data is per se a complex problem, since candidate FDs can be exponential in the number of attributes, and their detection requires analyzing a huge number of attribute combinations [1]. With respect to this scenario, the continuous discovery of FDs from data streams entails the continuous update of the set of discovered FDs as new data are read from the stream. Differently from the incremental discovery problem, the continuous profiling requires to design of extremely fast discovery processes, which should also manage considerably long, possibly infinite, executions. Moreover,

while in traditional databases it is important to manage insertion, update, and deletion operations, even if they occur less frequently than queries, in a data stream context only the insertion of new tuples matters [72]. Nevertheless, although the focus is mainly placed on fast and continuous insertion operations, it would be useful to give the possibility to forget information related to tuples processed less recently, leading to the necessity of performing the discovery process only on a temporal window of the data [1].

A data stream considers a sequence of data items, also known as tuples, $\langle t_1, \ldots, t_N, \ldots \rangle$, which need to be processed while minimizing the usage of memory space and the average time for processing each stream element, having the possibility to scan the stream only once [11]. The typical approach is to maintain a light summary of the processed information by building a data structure capable of guaranteeing reduced memory usage with the respect the stream size. More formally, given an ordered and infinite set T of discrete time instants in which data items are read from the stream, forming an infinite set r of data items, a *data stream* S is a mapping $S : T \rightarrow 2^{|r|}$ that at each instant $\tau \in T$ returns a finite subset of data items from r, all sharing a common schema R [129]. Moreover, each data item t is related to a specific timestamp $t.\lambda$, which continuously increases as new data items are read from the stream.

Given a data stream S, we need to consider the stream contents, which can vary according to the specific stream settings. In particular, by default the *current stream contents* $S(\tau)$ of a data stream S at time τ is the set $S(\tau) = \{t \in S : t.\lambda \leq \tau\}$ [129]. Moreover, in order to limit the data items to the most recent ones, in the context of data stream management sliding windows are usually applied. Thus, when the collection of items is limited by a *sliding window* with size w, then the *current stream contents* $S(\tau)_w$ of S at time τ is the set $S(\tau)_w = \{t \in S : (\tau - w) < t.\lambda \leq \tau\}$. Consequently, a data item t^- is said to be *expired* if and only if $t \notin S(\tau)_w$. Furthermore, we can define the *expired stream contents* after the last sliding of the window over a stream S from time τ to time $\tau + 1$ as the set $S^-(\tau + 1)_w = \{t \in S : t \in S(\tau)_w \wedge t \notin S(\tau + 1)_w\}$.

In the context of FD discovery, sliding windows can be used to forget extremely old data items possibly causing the invalidation of some FDs.

To this end, before evaluating novel data items in the new time instant $\tau+1$, when a sliding window with size w is considered, it is necessary to evaluate new potential FDs becoming valid because of the deleted data items included in $S^-(\tau+1)_w$. More formally, let $X \to A$ be a minimal FD holding on the set of data items $S(\tau)_w$ with common schema R, with $X, A \subseteq attr(R)$, we need to consider the following possible effects whenever a data item is managed (inserted or deleted) at time $\tau+1$.

- **Refutation of minimality**. Since $X \to A$ is minimal at time τ, then any $X \setminus B \to A$, with $B \in X$, is not valid at time τ also known as maximal NON-FD. In other words, there exists at least a pair of data items (t_1, t_2) in $S^-(\tau)_w$ such that $t_1[X \setminus B] = t_2[X \setminus B] \wedge t_1[A] \neq t_2[A]$. Let us suppose that (t_1, t_2) is the only pair of data items in $S(\tau)_w$ invalidating $X \setminus B \to A$, and that t_1 expires at time $\tau+1$, that is $t_1 \in S^-(\tau+1)_w$, denoted as t_1^-. Consequently, $X \to A$ is no longer minimal, since $X \setminus B \to A$ has becoming valid at time $\tau+1$.
- **Invalidation of functional dependencies**. Since $X \to A$ holds at time τ, then for each pair of data items (t_1, t_2) in $S(\tau)_w$, $t_1[A] = t_2[A]$ whenever $t_1[X] = t_2[X]$. Thus, $X \to A$ is invalidated at time $\tau+1$ if and only if at least one of the following two cases occur:
  - A new data item t_3 is read at time $\tau+1$, and

    $$t_1[X] = t_3[X] \wedge t_1[A] \neq t_3[A]$$

  - New data items t_4 and t_5 are read at time $\tau+1$, and

    $$t_4[X] = t_5[X] \wedge t_4[A] \neq t_5[A]$$

By considering the above discussed effects that have to be managed during the FD discovery over data streams, it is possible to follow the candidate generation strategies introduced in Section 5.1.1. Moreover, for each candidate FD that becomes potentially minimal at time $\tau+1$, also the *inference* issue has to be considered in order to guarantee the minimality of resulting FDs. This issue does not depend on the type of

data sources and on how the latter are dynamically updated, and it can be treated as described in Section 5.1.1.

The exponential complexity of the FD discovery problem becomes particularly challenging in the context of data streams. In fact, new tuples read from the stream can make existing minimal FDs no longer valid, possibly yielding new candidate FDs. Thus, even if reading of tuples from the data streams requires high processing speed, in the worst case even one tuple can require a complete exploration of the search space, entailing the same asymptotic complexity of the general problem (Section 2). Moreover, another important challenge for the FD discovery over data streams concerns the necessity of efficiently validating each candidate FD, without considering the complete set of not expired data items.

In Section 5.5, we will discuss a new FD discovery algorithm that allows to continuously infer and update FDs holding on a data stream, as the data are read from it. The algorithm relies on new data structures and a new validation method to handle a dynamic discovery process and reduce the data load inbound stream.

5.1.3 *Incremental Discovery of* RFD$_e$*s*

As discussed in Section 2, RFD$_e$ discovery is a complex problem even in static scenarios. Moreover, if we consider an incremental scenario in which new tuples could be inserted between consecutive time instants, i.e., from τ to $\tau + 1$, then the RFD$_e$s detected at time τ might be invalidated. Thus, it is necessary to re-execute discovery algorithms on the data instance updated at time $\tau + 1$. To this end, incremental RFD$_e$ discovery algorithms aim to update the set of RFD$_e$s without requiring the complete re-execution of the discovery algorithm.

Given a set Σ_τ of all minimal RFD$_e$s holding at time τ on an instance r of a database schema R, the discovery algorithm must update Σ_τ whenever one or more tuples are added from time τ to time $\tau + 1$. In particular, for each minimal RFD$_e$ in Σ_τ the validation process must be re-executed, since the $g3$-error e of an RFD$_e$ φ holding at time τ might increase or decrease upon the insertion of new tuples. More specifically, when the $g3$-error

increases, it might exceed the extent threshold, yielding the invalidation of φ. On the contrary, when the error decreases, it might also decrease for previous candidate RFD$_e$s generalizing φ, then the minimality of φ might be refuted.

More formally, let $\varphi : X \xrightarrow{\Psi \leq \varepsilon} A$ be a minimal RFD$_e$ holding at time τ on an instance r of R, with $X, A \subseteq attr(R)$, we need to consider the following possible effects whenever new tuples are added.

- **Invalidation of** RFD$_e$s. If $X \xrightarrow{\Psi \leq \varepsilon} A$ holds at time τ, then the g3-error e_τ is below the threshold ε. Thus, $X \xrightarrow{\Psi \leq \varepsilon} A$ is invalidated at time $\tau + 1$ if and only if the error $e_{\tau+1}$ is greater then e_τ and exceeds the threshold ε, i.e., $e_{\tau+1} > \varepsilon \geq e_\tau$.

- **Refutation of minimality**. If $X \xrightarrow{\Psi \leq \varepsilon} A$ is minimal at time τ, then any $X \setminus B \xrightarrow{\Psi \leq \varepsilon} A$ with $B \in X$ is not valid at time τ. In other words, the g3-error e computed at time τ on each $X \setminus B \xrightarrow{\Psi \leq \varepsilon} A$ is always greater than ε. Let us suppose that among them there exists at least one candidate RFD$_e$ $X \setminus B \xrightarrow{\Psi \leq \varepsilon} A$ whose g3-error $e_{\tau+1}$ is lower than or equal to ε, then $X \xrightarrow{\Psi \leq \varepsilon} A$ is no longer minimal at time $\tau + 1$.

Thus, if a minimal RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$ is invalidated at time $\tau + 1$, then new candidate RFD$_e$s need to be analysed, by considering all possible candidate RFD$_e$s $XB \xrightarrow{\Psi \leq \varepsilon} A$, such that $B \notin (X \cup A)$. On the contrary, if a RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$ is minimal at time τ but not at time $\tau + 1$, then it is necessary to check the validation of all possible RFD$_e$ candidates $X \setminus B \xrightarrow{\Psi \leq \varepsilon} A$, with $B \in X$. Notice that, the verification of minimality also needs to be iterated starting from the new validated RFD$_e$s (see the inference issue in Section 5.1.1).

The necessity to perform validation and the consequent effects described above primarily refers to all minimal RFD$_e$s detected at time τ. In general, an incremental discovery strategy aims to reduce the possible navigation steps over the search space, even if it requires the generation of new candidate RFD$_e$s 1) by using previously holding RFD$_e$s, and 2) by exploring the parts of the search space that are not reachable from them.

In the first case, starting from RFD$_e$s holding at time τ, it is possible to limit the search space to their neighbors, which represent the new candidate RFD$_e$s at time $\tau + 1$. Here, given an RFD$_e$ $\varphi : X \xrightarrow{\Psi \leq \varepsilon} A$ holding at time τ, its neighbors can have one of the following forms $\varphi' : XB \xrightarrow{\Psi \leq \varepsilon} A$ with $B \in attr(R) \backslash (X \cup A)$, or $\varphi'' : X \backslash B \xrightarrow{\Psi \leq \varepsilon} A$, with $B \in X$.

Instead, since the second case would catch candidate RFD$_e$s that are not neighbors of any RFD$_e$ holding at time τ, it is necessary to consider possible additional candidate RFD$_e$s, according to the three following properties:

Property 1. Let Σ_τ be the set of all RFD$_e$ holding at time τ, and let

$$Z_\tau = \left\{ \bigcup_{A \in \text{attr(R)}} A \mid \exists X \xrightarrow{\Psi \leq \varepsilon} A \in \Sigma_\tau \right\}$$

if $|Z_\tau| \neq |attr(R)|$, then for each attribute $A \notin Z_\tau$ it is necessary to add all possible RFD$_e$ candidates $B \xrightarrow{\Psi \leq \varepsilon} A$, with $B \neq A$.

Property 2. Let Σ_τ be the set of all RFD$_e$ holding at time τ, and for each $A \in attr(R)$ let

$$S_\tau = \left\{ \bigcup_{B \in \text{attr(R)}} B \mid \exists (X \xrightarrow{\Psi \leq \varepsilon} A \in \Sigma_\tau) \wedge (B \in X) \right\}$$

if $|S_\tau| \neq |attr(R)|$, then for each attribute $B \notin S_\tau$ it is necessary to add a candidate RFD$_e$ $B \xrightarrow{\Psi \leq \varepsilon} A$.

Property 3. Let Σ_τ be the set of all RFD$_e$ holding at time τ, for each $A, B \in attr(R)$, $B \neq A$, let $l = min\{|X| \mid X \xrightarrow{\Psi \leq \varepsilon} A \in \Sigma_\tau \wedge B \in X\}$, and let

$$N_B^l = \left\{ \bigcup_{X \text{with} |X| = l} X \mid (X \xrightarrow{\Psi \leq \varepsilon} A \notin \Sigma_\tau) \wedge (B \in X) \right\}$$

then for each candidate LHS $X \in N^l_B$ it is necessary to add a candidate RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$ if and only if there is no $X' \subset attr(R)$ such that $X' \xrightarrow{\Psi \leq \varepsilon} A \in \Sigma_\tau$ and $X \subseteq X'$.

In other words, the exploration of the search space starts from all RFD$_e$s holding at time τ, and from those generated through Properties 1, 2, and 3. In Section 5.6, we describe a new incremental discovery algorithm for RFD$_e$s, which exploits these properties to perform an efficient validation process.

5.2 LITERATURE REVIEW

While in the previous chapter we reviewed methodologies and algorithms for RFD discovery on static datasets, in this section we review the literature concerning FD discovery algorithms, including incremental ones, and incremental RFD$_e$ discovery algorithms. Since to the best of our knowledge there are no incremental RFD$_c$ discovery algorithms, we will focus on incremental RFD$_e$ discovery algorithms. The reviewed incremental algorithms and their theoretical foundations represent the seeds for defining new discovery processes and applying them in new domains, such as data stream scenarios.

FD *discovery.* The FD discovery problem dates back to the '80s, when the first discovery algorithm was defined [111], from which other well-known proposals were derived [84, 85]. As seen in the previous section, FD discovery is an extremely complex problem, since the number of potential FDs can be exponential, and their detection might requires analyzing a huge number of column combinations [1].

In the literature there are two main categories of automatic methods for discovering FD from data, namely *column-based* and *row-based* algorithms. Column-based algorithms model the search space as an attribute lattice, which permits to consider candidate FDs at each lattice level in terms of edges. Such candidates need be validated, which entails evaluating value combinations or efficient representations of them. The whole process is made efficient by exploiting FDs already validated, in order to

prune the search space for the generation of new candidates. Examples of column-based algorithms, also known as top-down algorithms, include the algorithms TANE [85], FD_Mine [164], FUN [121], and DFD [3]. On the other hand, row-based algorithms derive candidate FDs by analyzing the cross product between all possible combinations of tuple pairs, aiming to derive two attribute subsets, namely *agree-set* and *difference-set*. In particular, they search for attribute sets that agree on the values of certain tuple pairs, since they can functionally determine other attributes agreeing on the same tuple pairs. Once all the agree sets are computed, it is possible to derive all valid FDs from them. Examples of row-based algorithms, also known as bottom-up algorithms, include the algorithms DepMiner [109], FastFD [162], and FDep [64].

Column-based algorithms usually outperform row-based ones on datasets with many rows and few columns, whereas on datasets with few rows and many columns the row-based algorithms usually perform better, as demonstrated in extensive experimental results [126]. In order to obtain better performances in all cases, a hybrid algorithm has recently been proposed, namely HyFD [128]. It combines row- and column-efficient discovery techniques by managing two separated phases, one in which it calculates FDs on a randomly selected small subset of tuples (column-efficiency), and the other in which it validates the discovered FDs on the entire dataset.

Incremental discovery. The discovery algorithms presented above need to process the whole dataset. Thus, whenever the datasets are updated, they need to be re-executed from scratch, whereas it would be desirable to have some incremental discovery strategies. One of the first theoretical proposals of an incremental algorithm for FD discovery has been provided in [161]. It exploits the concepts of tuple partitions and monotonicity of FDs to avoid the re-scanning of the entire database. Another proposal is based on the concept of functional *independency* through which the set of FDs updated over time is maintained [14]. A recent algorithm, named DynFD has been proposed in [140], which permits to discover and maintain FDs in dynamic datasets. It continuously adapts the validation structures of FDs in order to evolve them with batches of insertions, updates, and deletions of data. Moreover, another approach for discovering *Order De-*

pendencies (ODs) after the insertion of new tuples is defined in [166]. It is based on a strategy enabling an intelligent traversal process and reduced access to the whole dataset. Furthermore, the only incremental algorithm for the discovery of RFD$_e$s proposed in the literature is AD-MINER [59]. It permits to incrementally updating dependency information exploiting several logical operations.

The problem of automatically inferring metadata over incremental scenarios has also been addressed in the context of association rule mining [117], which is somehow related to RFD$_e$s discovery. In particular, an association rule holding on a given dataset is defined in terms of *support* and *confidence* measures. In the literature, there exist two different types of incremental algorithms for association rule mining: single-objective [41] and multi-objective [56]; they both use confidence as the main measure to be optimized. However, multi-objective algorithms also attempt to optimize other measures, such as comprehensibility and interestingness. Similarly to those for incremental RFD$_e$s discovery, incremental algorithms for association rule mining start using information gathered from previous executions, trying to reduce the search space. In general, the main strategy prescribes to verify how rules change after updating the data, and when it is necessary to re-scan the entire dataset in order to find new valid rules. More specifically, if a rule is valid on the old dataset, and it is still valid on the new increment, then it is not necessary to re-scan the entire dataset. Instead, if a rule is no longer valid, then it is necessary to execute a scan process on the entire dataset.

Some of the methodologies surveyed above (i.e., [14, 161]) are not implemented, whereas others need to store data structures produced during the discovery process. Thus, new efficient approaches need to be designed in order to tackle the discovery of metadata in incremental scenarios. To this end, several new incremental discovery algorithms for FDs and RFD$_e$s are discussed in the next sections, by also describing new efficient validation and pruning approaches that allow the proposed algorithms to perform an efficient discovery process. Moreover, we discuss a new FD discovery algorithm, which allows to continuously infer and update FDs holding on a data stream, as data are read from it. Finally, we present an incremental discovery algorithm for RFDs relaxing on the

extent (RFD_es) that is able to manage the validation of candidate RFDs and the generation of possibly new RFD candidates upon the insertion of new tuples, while limiting the size of the overall search space.

5.3 INCREMENTAL-FD: AN INCREMENTAL DISCOVERY ALGORITHM FOR FDS

This section presents the first incremental approach we propose for FDs discovery, named INCREMENTAL-FD, which is able to update the set of holding FDs upon the insertion of new tuples to a relation instance, without having to restart the discovery process from scratch. In particular, we introduce the data representation and the search strategies underlying INCREMENTAL-FD, and then detail data structures, pruning, and validation processes for discovering FDs.

5.3.1 *Methodology*

The methodology behind the proposed incremental discovery algorithm reads in input the FDs associated to an instance r of a relation schema R with M attributes, and represents them by using a binary vector of M elements, aiming to perform an efficient strategy for traversing the search space. Figure 5.1 shows the binary representation of an FD. In particular, each FD $X \rightarrow A$ is represented by means of two binary vectors v_X and v_A of M elements, such that the v_X contains all the LHS attributes, whereas the v_A contains the RHS attribute of the FD. Each location of such vectors represents an attribute of the relation schema R. In particular, given an FD $X \rightarrow A$ on r, if $v_X[i] = 1$ then the i-th attribute in r belongs to X. In this way, it is possible to represent FDs with hundreds of attributes in a compact and lightweight data structure.

In order to provide an efficient way to store FDs in incremental scenarios, we propose a new data structure, named *linked map*, which allows the proposed algorithm to ensure that FDs are quickly saved and retrieved. In particular, the map uses the binary vector representation of FDs as keys, and the pointer to the next FD according to an ordering criterion,

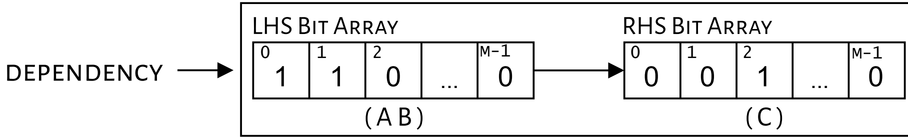


Figure 5.1: Binary vector representation of a functional dependency.

based on the LHS cardinalities. Moreover, the *linked map* exploits two arrays to store the first and the last entries of the map, aiming to reduce the time for insertion operations. This strategy allows to easily perform a level-wise discovery strategy based on a lattice representation of the search space.

INCREMENTAL-FD algorithm follows the well-known APRIORI strategy for the generation of candidate FDS [84]. It models the search space as a graph representation of a lattice, which is partitioned into levels, where level L_i contains all attribute combinations of size i (see Section 5.1.1).

Figure 5.2 shows an example of lattice in which 5 attributes have been considered. 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

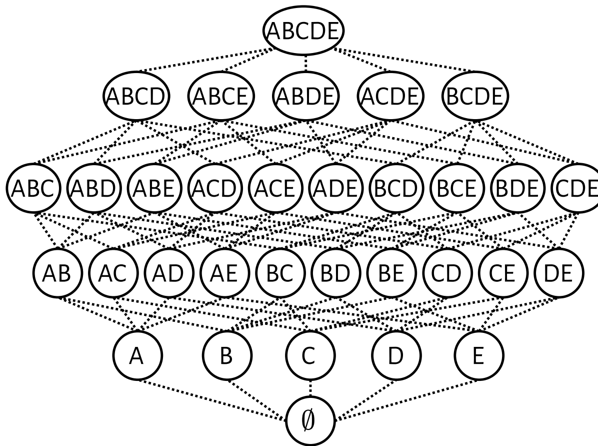
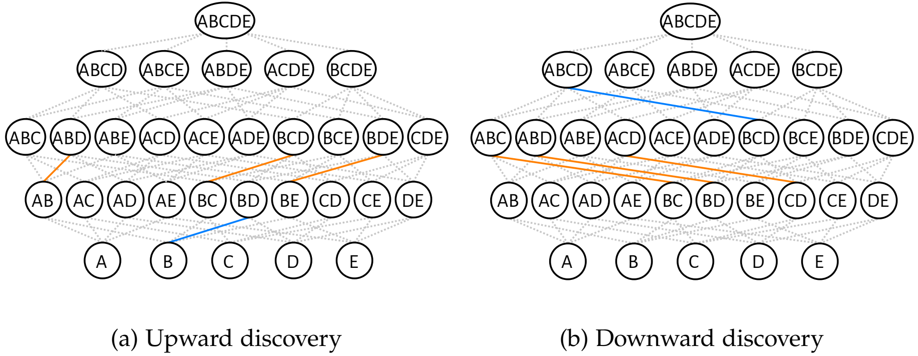


Figure 5.2: The lattice search space representation for the attribute set {A,B,C,D,E}.



(a) Upward discovery

(b) Downward discovery

Figure 5.3: Discovery steps on the lattice search space representation.

attributes. In other words, each edge refers to the inclusion relation between two attribute sets. Thus, a lattice permits to consider candidate FDs at each level in terms of lattice's edges, allowing to represent the LHS and the RHS of an FD [126]. Based on the APRIORI search strategy, INCREMENTAL-FD algorithm starts from the candidate FDs at the lowest lattice level, i.e., candidate FDs with one attribute on the LHS, and it performs *upward* and/or *downward* discovery processes to explore the search space. More specifically, given a candidate FD φ, if it is not valid at time $\tau + 1$, then INCREMENTAL-FD performs an *upward* discovery step to consider possible new candidate FDs; otherwise it performs a *downward* discovery to check the minimality of φ. The *upward* discovery step consists of the specialization of φ accomplished by adding a new attribute on its LHS. As an example, let us consider the lattice in Figure 5.3(a). If the candidate FD $B \rightarrow D$ (highlighted in blue) does not hold at time $\tau + 1$, then the upward discovery step identifies the following three candidate FDs (highlighted in orange): $AB \rightarrow D$, $BC \rightarrow D$, and $BE \rightarrow D$. Conversely, the *downward* discovery step consists of the generalization of φ accomplished by iteratively removing one attribute from its LHS. As an example, let us consider the FD $BCD \rightarrow A$ highlighted in blue in Figure 5.3(b); to check its minimality, the downward discovery step verifies the following three candidate FDs (highlighted in orange): $BC \rightarrow A$, $BD \rightarrow A$, and $CD \rightarrow A$.

After defining the discovery strategy, it is necessary to focus the discussion on the approach adopted for representing and storing data. INCREMENTAL-FD uses partitions to store a lightweight reference of the tuples. To this end, let us review the formal definition of partitions extracted from [85]:

Definition 5.3.1 (Partitions). Let u and w be two tuples of a relation r, then u and w are equivalent with respect to a given set X of attributes if $u[A] = w[A]$ for all A in X. Any attribute set X partitions the tuples of the relation into equivalence classes. We denote the equivalence class of a tuple $u \in r$ with respect to a given set $X \subseteq R$ by $[u]_X$, i.e., $[u]_X = \{u \in r \mid u[A] = w[A]$ for all $A \in X\}$. The set $\pi_X = \{[u]_X \mid u \in r\}$ of equivalence classes is a partition of r under X. That is, π_X is a collection of disjoint sets of tuples, namely equivalence classes, such that each set has a unique value for the attribute set X and the union of the sets equals the relation r. The rank $|\pi|$ of a partition π is the number of equivalence classes in π.

The use of tuple partitions avoids accessing data during the discovery process and it allows to quickly update the set of data, by inserting o removing one or more items from the partitions. Thanks to this data representation, INCREMENTAL-FD performs an efficient validation of FDS through the refinement property [85]:

Definition 5.3.2 (Refinement). A functional dependency $X \rightarrow A$ holds on r iff $|\pi_X| = |\pi_{(X \cup A)}|$, where π_X and $\pi_{(X \cup A)}$ are the sets of equivalence classes, partitioning r on X and $X \cup A$, respectively.

Example 1. Table 5.1 shows a relation instance r in which three new tuples have been inserted at time $\tau + 1$. Starting from the partitions of the instance r at time τ, that is, $\pi_A = \{[0,1],[3,4],[2]\}$, $\pi_B = \{[3,4],[0],[1,2]\}$, $\pi_C = \{[0,2],[4],[1,3]\}$, $\pi_D = \{[0,3],[4],[2],[1]\}$, the incremental algorithm computes the updated partitions at time $\tau + 1$, that is, $\pi'_A = \{[0,1],[3,4,7],[2,5,6]\}$, $\pi'_B = \{[3,4],[0],[6,7],[1,2,5]\}$, $\pi'_C = \{[0,2,5,7],[4],[1,3,6]\}$, $\pi'_D = \{[0,3,5],[7],[4],[2],[6],[1]\}$, and loads the minimal FDS holding at time τ. The latter are represented in terms of binary vectors and inserted into the *linked map*.

	ID	A	B	C	D
	t_0	Andorra	1	French	French
	t_1	Andorra	2	English	Russian
	t_2	San Marino	2	French	Italian
	t_3	Cipro	4	English	French
	t_4	Cipro	4	Greek	Spanish
+	t_5	San Marino	2	French	French
+	t_6	San Marino	3	English	German
+	t_7	Cipro	3	French	Arabic

Table 5.1: An example of a relation instance updated at time $\tau + 1$.

Figure 5.4 shows the *linked map* obtained at time $\tau + 1$ for the new instance of Table 5.1. It visualizes all the considered candidate FDs. In particular, each binary vector in Figure 5.4 represents: 1) a minimal FD holding at time $\tau + 1$, 2) a candidate FD that has been invalidated at time $\tau + 1$, or 3) a candidate FD that is not minimal at time $\tau + 1$.

Discovery Process. As said above, the *linked map* contains a fast array to directly link the first FD of each LHS cardinality. Thus, the algorithm starts by considering the FD with the lowest LHS cardinality, which is selected by using the fast array, as shown in Figure 5.4 for the FD $(0110) \rightarrow (0001)$. Consequently, using the refinement property, this FD is removed because $|\pi_X| \neq |\pi_{(X \cup A)}|$. Starting from this, the algorithm calculates the next candidate FDs, by considering LHS supersets (i.e., according to the upward search strategy), and verifying if there are no other minimal FDs with respect to them. Thus, only the candidate FDs that cannot be inferred from any other one are added to the map. Therefore, in the Figure 5.4, the candidate FD $(1110) \rightarrow (0001)$ is added to the map, since it is generated from the invalidation of $(0110) \rightarrow (0001)$ and

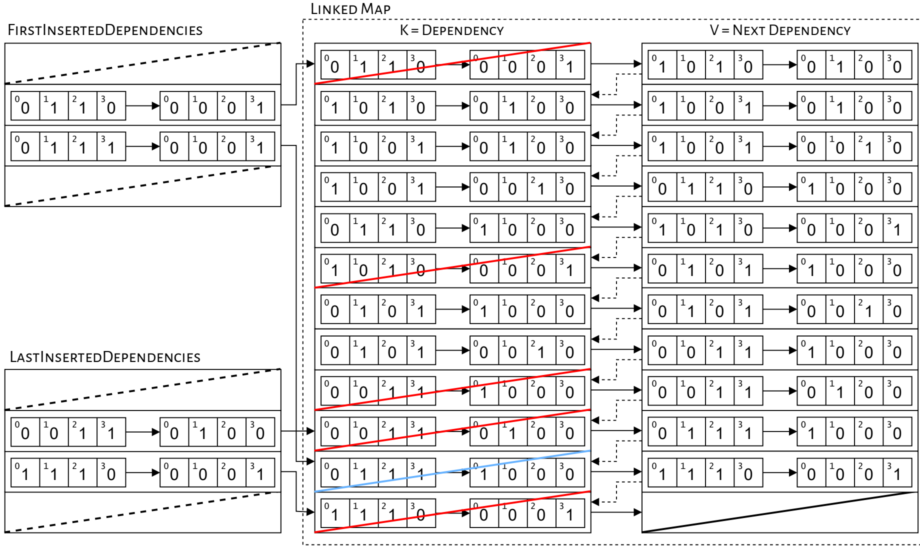


Figure 5.4: The *linked map* related to Example 1.

$(1010) \rightarrow (0001)$. Vice versa, the candidate FD $(0111) \rightarrow (1000)$ (deleted with a blue line) generated from the invalidation of $(0011) \rightarrow (1000)$, is not added to the map, since there are the FDs $(0110) \rightarrow (1000)$ and $(0101) \rightarrow (1000)$ that are minimal with respect to it. The algorithm proceeds until all FDs are explored. In fact, the FD $(1110) \rightarrow (0001)$ is not valid at time $\tau + 1$, and consequently it is removed from the map. Finally, the algorithm returns the new minimal set of FDs holding on the relation instance at time $\tau + 1$.

The complete discovery process defined according to the proposed methodology is described in Algorithm 7. In particular, for each FD holding at time τ (line 1), it verifies if $X \rightarrow A$ also holds at time $\tau + 1$ through the `REFINEMENT` function (line 2). Next, if $X \rightarrow A$ is not valid, the algorithm generates new candidate FDs at a higher level, by excluding those that can be inferred (lines 3-6). Instead, if $X \rightarrow A$ is valid, the latter is added to the result set if and only if it cannot be inferred by other holding FDs at a lower level (lines 8-11).

Algorithm 7 Incremental Discovery Algorithm

INPUT:
$\Sigma_\tau \rightarrow$ a set of valid and minimal FDs at time τ
OUTPUT: $\Sigma_{\tau+1} \rightarrow$ a set of valid and minimal FDs at time $\tau + 1$

```
1:  for all X → A ∈ Σ_τ do
2:      if REFINEMENT(X → A) is not valid then
3:          L_{l+1} ← NEXTLEVEL(X → A)
4:          for all X_{l+1} ∈ L_{l+1} do
5:              if not INFERENCE(X_{l+1} → A then)
6:                  Σ_τ ← Σ_τ ∪ {X_{l+1} → A}
7:      else
8:          L_{l−1} ← PREVLEVEL(X → A)
9:          for all X_{l−1} ∈ L_{l−1} do
10:             if not INFERENCE(X_{l−1} → A) then
11:                 Σ_τ ← Σ_τ \ {X → A}
12: Σ_{τ+1} ← Σ_τ
13: return Σ_τ
```

5.3.2 *Experimental Evaluation*

In what follows, we present experimental results concerning the performance of the proposed approach in discovering FDs. In particular, the performed tests show how the new proposed approach can improve time performances with respect to the complete re-execution of the FD discovery algorithm upon updates to the relation instance.

Implementation details. The algorithm has been developed in Java 11.0.2. In particular, in order to improve the performance of the algorithm and avoid the re-calculation of partitions, we also introduced a methodology for caching partitions, since the latter are widely used for the validation of candidate FDs.

Hardware and datasets. Tests have been accomplished on a Mac with an Intel Xeon processor at 3.20 GHz 8-core and 64GB of RAM. Moreover, to ensure a proper execution on the considered datasets, the Java memory

heap size allocation has been set to 40GB to permit a proper execution on datasets with a high number of tuples/attributes.

We evaluated the proposed approach on several real-world datasets[1], previously used for testing FD discovery algorithms [126]. Statistics on the characteristics of the considered datasets are shown in Table 5.2. Such datasets are composed of one relation, since FD discovery algorithms always consider de-normalized databases.

Evaluation process. We carried out different tests by using the minimal FDs extracted through the TANE algorithm [85] as a starting point. All the tests were performed on datasets split into two parts. The first part represents the relation instance at time τ, and it has been given in input to TANE. The complete dataset represents the relation instance at time $\tau + 1$ analyzed by the proposed incremental discovery algorithm. Moreover, we re-executed the TANE algorithm on the complete dataset. This allowed us to analyze the resulting FDs, and to compare execution times of our approach with respect to a complete re-execution of TANE. Moreover, we executed other two experiments: 1) by varying the number of tuples inserted at time $\tau + 1$; 2) by varying the number of rows/columns of the dataset.

Analysis of the results. The first test has been conducted by simulating the insertion of 50% of tuples in the complete dataset. The results are reported in Table 5.2, where comparisons between the times of TANE and INCREMENTAL-FD are shown. We can notice that INCREMENTAL-FD is, in general, more efficient, despite the variability of the number of rows/columns. This is mainly due to the pruning strategies introduced by the incremental approach. However, there are few cases in which the execution times are equivalent to those of TANE. This happens when there is a huge number of FDs holding at time τ.

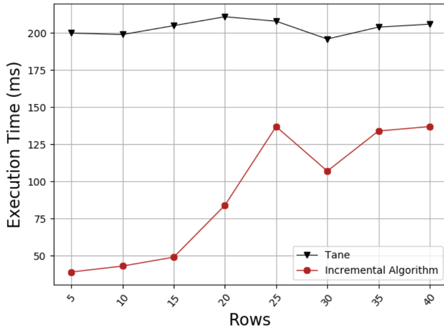
The second test aimed to evaluate the relationship between the execution times and the sizes of the datasets. In particular, we selected datasets by varying the number of tuples inserted at time $\tau + 1$ in the range 5%-40% with respect to the complete dataset. Results are shown in Figure 5.5. In particular, the results for *Nursey* (Figure 5.5d) and *Citeseer* (Figure

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

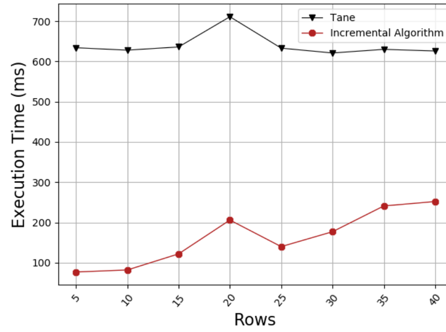
Dataset	Rows	Cols	%Inserts	#FDS	TANE Time (ms)	Incremental-FD Time (ms)
Abalone	4148	9	50 %	137	308	**183**
Adult	32561	15	50 %	78	43033	**1085**
Balance-scale	624	5	50 %	1	130	**6**
Breast-cancer W.	699	11	50 %	46	316	**284**
Breast-cancer	285	10	50 %	1	230	**132**
Bridges	108	13	50 %	142	218	**195**
Bupa	344	7	50 %	25	134	**19**
Callt	10081	3	50 %	1	176	**30**
Cars	407	9	50 %	67	162	**48**
Car_data	1727	7	50 %	1	197	**7**
Chess	1999	7	50 %	6	184	**7**
Citations	1000	9	50 %	76	203	**182**
Citeseer	10000	7	50 %	10	456	**116**
Citeseer	20000	6	50 %	4	721	**816**
Cmc	1472	10	50 %	1	477	**15**
DBLP	20000	8	50 %	28	364	**172**
Echocardiogram	132	13	50 %	538	191	**363**
Ecoli	335	9	50 %	54	139	**24**
Haberman	305	4	50 %	0	110	**1**
Hayes-roth	132	6	50 %	5	119	**2**
Iris	149	5	50 %	4	116	**6**
Letter	20000	17	50 %	61	177087	**6203**
Mammography	960	6	50 %	0	150	**1**
Nursery	12960	9	50 %	1	1065	**45**
Servo	166	5	50 %	1	115	**3**
Tae	150	6	50 %	2	122	**14**
Tax	100000	15	50 %	364	29368	**28598**
Wine	178	14	50 %	1374	209	**131**

Table 5.2: Characteristics of the used datasets and discovery results.

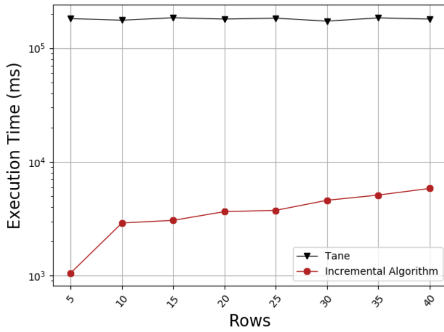
5.5b) datasets show a considerable reduction of the execution times when considering the incremental approach. This is probably due to the fact that there were few minimal FDs holding at time τ. Moreover, although



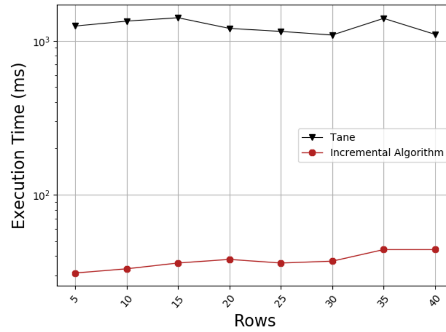
(a) Citations dataset.



(b) Citeseer dataset.



(c) Letter dataset.



(d) Nursery dataset.

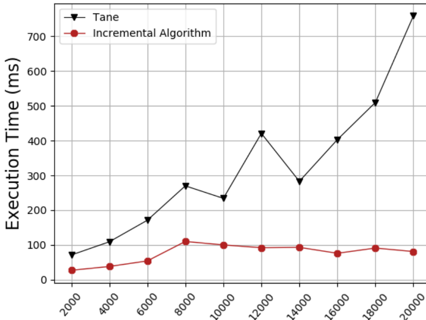
Figure 5.5: A comparison between execution times of TANE with respect to our proposal by varying the percentage of inserted tuples at time $\tau + 1$.

this number is higher for *Citations* and *Letter*, the time performances of our approach are still better than those of TANE (Figures 5.5a and 5.5c).

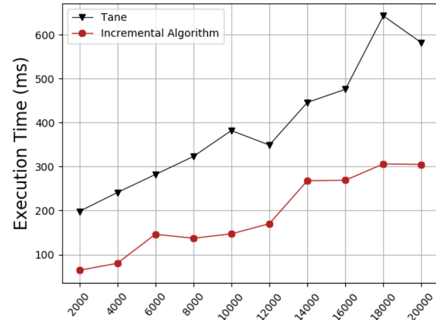
The last test allowed us to evaluate the discovery time of INCREMENTAL-FD on datasets with a fixed number of tuples, but with a variable number of columns (first session), and with a fixed number of attributes, but a variable number of tuples within an interval range of $[2000 - 20000]$ tuples, with step 2000 (second session). For these experimental evaluations *DBLP* and *Citeseer* datasets have been selected. Table 5.3 contains the characteristics of the datasets and the results of the experiments with a variable number of attributes. Instead, the results of the second session

Dataset	Tuples	Attributes	%Inserts	TANE Time (ms)	Incremental-FD Time (ms)
DBLP	20000	2	3%	220	**32**
DBLP	20000	3	3%	277	**78**
DBLP	20000	4	3%	293	**126**
DBLP	20000	5	3%	366	**162**
DBLP	20000	6	3%	418	**232**
DBLP	20000	7	3%	490	**245**
Citeseer	20000	2	3%	175	**2**
Citeseer	20000	3	3%	203	**2**
Citeseer	20000	4	3%	360	**3**
Citeseer	20000	5	3%	376	**45**
Citeseer	20000	6	3%	435	**73**
Citeseer	20000	7	3%	456	**89**

Table 5.3: A comparison between execution times of TANE with respect to our proposal by varying the number of attributes.



(a) Citeseer dataset.



(b) DBLP dataset.

Figure 5.6: A comparison between execution times of TANE with respect to our proposal by varying the number of tuples.

are shown in Figure 5.6. By analyzing Table 5.3, we can notice that the execution times of the proposed approach are always lower than the execution times of TANE.

Moreover, we noticed that for Incremental-FD the *DBLP* dataset is more critical than *Citeseer* when the number of attributes increases. Also in this case, this is probably due to the higher number of FDs holding at time $\tau + 1$. Different results have been achieved when considering the variation in the number of tuples (Figure 5.6), since although this is non-monotonic, the time trend grows faster for TANE.

5.4 REXY: AN INCREMENTAL FD DISCOVERY ALGORITHM BASED ON AN EFFICIENT REGEX VALIDATION PROCESS

This section presents a new incremental discovery algorithm for FDs named REXY (RegEX-based incremental discoverY), which represents an extension of the INCREMENTAL-FD algorithm introduced in Section 5.3. It includes a new validation method exploiting regular expressions (RegExs) to extract the subset of data affecting discovery results. In particular, we first describe the new compressed data structures used to optimize time and space usage of the discovery algorithm. Then, we introduce the validation method, by discussing how it can be adopted within an incremental FD discovery process. Finally, we demonstrate the effectiveness of the proposed algorithm on real-world datasets adapted for incremental scenarios, also by comparing it with the previously described algorithm INCREMENTAL-FD.

5.4.1 *Methodology*

The methodology underlying REXY represents a dataset by using lightweight references to its tuples, without losing significance. To this end, we map each attribute value to a unique numeric value, which represents an ID in the context of that attribute and it allows us to efficiently identify data updates and support the validation process. In particular, whenever new tuples are added to the dataset, the mapping of their values must be accomplished consistently with the previous attribute mappings. This representation yields a fast tuple comparison during the FD discovery process, and permits to create the RegEx, avoiding encoding issues over textual attribute values.

Example 1. Let us consider the snippet of the *Cars* dataset in Table 5.4a containing 9 attributes. As shown in Table 5.4b, after the mapping phase, each value has been mapped to a unique value for each attribute. For instance, let us suppose that the following two tuples are inserted in the dataset:

t_7	Ford Torino	17.0	8	302.0	140.0	3449.	10.5	70	US
t_8	BMW	26.0	4	121.0	113.0	2234.	12.5	70	EU

their values are parsed according to the previous values of the dataset, and mapped to the following two tuples:

t_7	7	7	3	7	7	7	6	1	3
t_8	3	3	1	3	3	3	3	1	1

We can notice that, after the mapping step, the tuple t_8 has become equal to t_3. Thus, it is not necessary to store t_8, whereas t_7 is stored because it represents a new value combination for the considered dataset.

The mapping step permits to lighten the representation of data, but it is necessary to define a novel data structure enabling a fast retrieval of candidate FDs to support the validation process. To this end, the latter exploits a hashmap, namely *RegExHashMap* (see Figure 5.7), which contains keys representing unique value combinations of tuples, and values representing the number of occurrences of them. In this way, it is possible to avoid duplicate entries and to quickly perform insertion and/or removal operations.

An efficient discovery methodology should permit to quickly validate candidate FDs on the relation instance under analysis, and ensure high adaptability to possible data changes. The proposed validation method relies on RegExs and exploits the RegExHashMap for fast retrieval operations. In other words, the RegExHashMap provides a compact string representation of data. Thus, a validation method relying on RegExs permits to efficiently discover possible violations according to the candidate FDs under analysis. More specifically, let $\varphi : X \rightarrow A$ be a candidate FD on a relation instance r of a relation schema R, the proposed approach creates a RegEx ρ to validate φ r of R. For sake of clarity, we describe the creation of ρ by considering a single change operation, consisting in the insertion of a single new tuple t. However, the strategy could be easily adapted to consider multiple new tuples by chaining different RegExs. In particular, the validation approach considers the projections $t[X]$ and

	Car (A)	MPG (B)	Cylinders (C)	Displacement (D)	Horsepower (E)	Weight (F)	Acceleration (G)	Model (H)	Origin (I)
t_1	Peugeot 504	25.0	4	110.0	87.00	2672	17.5	70	EU
t_2	Audi LS	24.0	4	107.0	90.00	2430	14.5	70	EU
t_3	BMW	26.0	4	121.0	113.0	2234	12.5	70	EU
t_4	Toyota C.	31.0	4	71.00	65.00	1773	19.0	71	JPN
t_5	Fiat 124B	30.0	4	88.00	76.00	2065	14.5	71	EU
t_6	Ford M.	21.0	6	200.0	85.00	2587	16.0	70	US
t_7	Ford Torino	17.0	8	302.0	140.0	3449	10.5	70	US

(a) Before the mapping step.

	Car (A)	MPG (B)	Cylinders (C)	Displacement (D)	Horsepower (E)	Weight (F)	Acceleration (G)	Model (H)	Origin (I)
t_1	1	1	1	1	1	1	1	1	1
t_2	2	2	1	2	2	2	2	1	1
t_3	3	3	1	3	3	3	3	1	1
t_4	4	4	1	4	4	4	4	2	2
t_5	5	5	1	5	5	5	2	2	1
t_6	6	6	2	6	6	6	5	1	3
t_7	7	7	3	7	7	7	6	1	3

(b) After the mapping step.

Table 5.4: Snippet of the *Cars* dataset to illustrate validation and discovery strategies of REXY algorithm.

$t[A]$ of the tuple t on the attributes of X and on A, respectively, to select value combinations that must be involved in the validation process.

Formally, to validate φ it is necessary to create a RegEx for the attribute set $X = B_1, B_2, \ldots, B_k$ and the attribute A, by considering $t[B_1], t[B_2], \ldots,$ $t[B_k]$, and $t[A]$ in the following way:

ρ_{LHS} = t[B_1][,]([0-9]+[,])+t[B_2][,]([0-9]+[,])+... ([0-9]+[,])+t[B_k]
ρ_{RHS} = (?!.+t[A]).+ We can notice that ρ_{LHS} contains the comma character

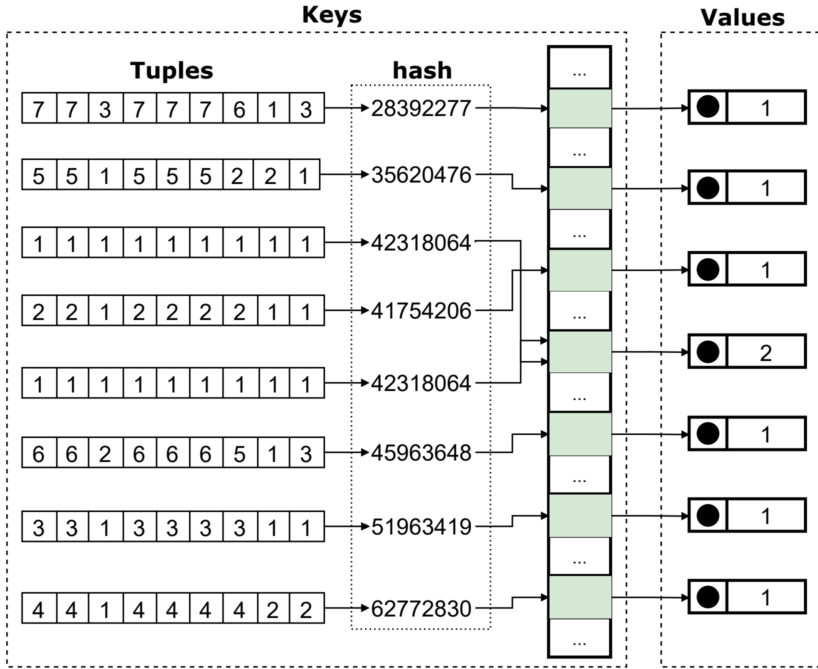


Figure 5.7: An example of *RegexHashMap* data structure.

as a separator between numerical values. The latter can represent $t[B_i]$, with $i = 1, \ldots, k$, or a sequence of numeric characters, i.e., $[0-9]+$, representing all the possible values for the attributes that are not in φ. In fact, one of the strengths of this approach is that it avoids to consider specific values for attributes not included in the candidate FDs under analysis. Similarly to the LHS, it is necessary to define a RegEx for the RHS, i.e., ρ_{RHS}. To this end, ρ_{RHS} represents the negative look ahead of $t[A]$ [15], enabling to consider only the tuples in which the value $t[A]$ does not appear. The combination of ρ_{LHS} and ρ_{RHS} yields a new RegEx ρ_φ, which is used to validate the candidate φ after the insertion of the tuple t.

Example 2. Let us consider the snippet of the *Cars* dataset after the mapping step (Table 5.4b), a candidate FD $\varphi : \{A, C, F\} \rightarrow H$, and the new inserted tuple t_7 defined in Example 1. The validation algorithm

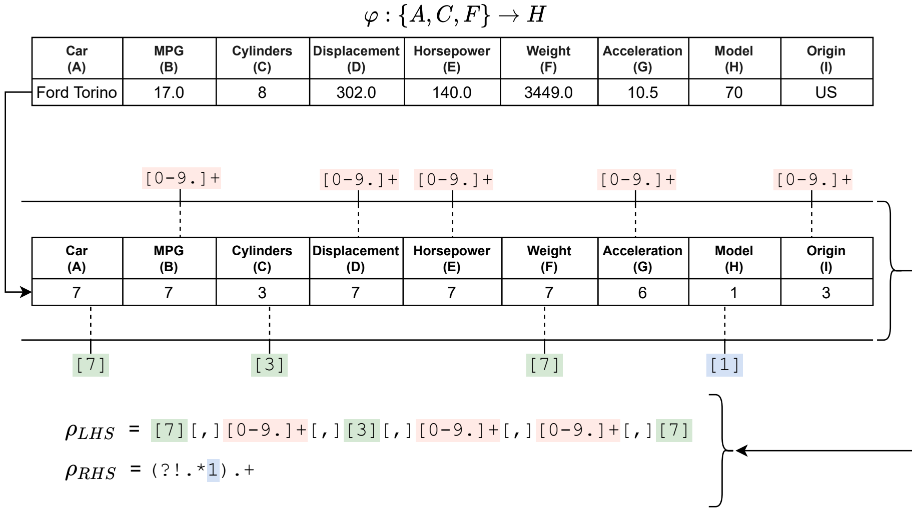


Figure 5.8: An example of RegEx creation for *Cars* dataset.

should check if this new tuple yields the invalidation of φ, according to the validation approach defined above. To this end, it is necessary to define the RegEx ρ_φ to validate φ, as shown in Figure 5.8. We can observe that the validation approach permits to construct ρ_{LHS} according to the new values 7, 3, and 7, projected by means of $t_7[A]$, $t_7[C]$, and $t_7[F]$ for all the attributes included in the LHS of φ, and by considering a generic numeric value (i.e., $[0-9]^*$) for the remaining attributes (see Figure 5.8). Moreover, it is necessary to define the regex ρ_{RHS} for the attribute H, by considering the negative look ahead of $t_7[H]$, in order to check if there exists at least one combination of values that is equal to the projection of the new tuple on the attributes A, C, and F, but with a different value for attribute H.

5.4.2 *The* REXY *Algorithm*

Algorithm 8 describes the main procedure of REXY. It takes in input a set of minimal FDs Σ_τ holding at time τ on a relation instance r, the

Algorithm 8 The REXY algorithm

INPUT:
$\Sigma_\tau \rightarrow$ a set of minimal FDs at time τ
$D_{\tau+1} \rightarrow$ the set of new tuples for the dataset D at time $\tau+1$
$D^-_{\tau+1} \rightarrow$ the set of removed tuples for the dataset D at time $\tau+1$
$\Gamma_\tau \rightarrow$ an instance of the RegExHashMap at time τ
OUTPUT: $\Sigma_{\tau+1} \rightarrow$ a set of new valid and minimal FDs at time $\tau+1$

1: $\Gamma_{\tau+1} \leftarrow \Gamma_{\tau+1} \setminus D^-_{\tau+1}$ ▷ RegExHashMap update
2: $\Gamma_{\tau+1} \leftarrow \Gamma_{\tau+1} \cup D_{\tau+1}$
3: **for each** $\varphi : \{X \rightarrow A\} \in \Sigma_\tau$ **in ascending order of** X **do**
4: **if** VALIDATION_REGEX(φ, Γ_τ, $D_{\tau+1}$) **is True then**
5: $\Sigma_{\tau+1} \leftarrow \Sigma_{\tau+1} \cup \varphi$
6: $\Sigma_d \leftarrow$ **do** DOWNWARD_DISCOVERY(φ) *as long as possible*
7: $\Sigma_\tau \leftarrow \Sigma_\tau \cup$ *select all valid candidates from* Σ_d
8: **else**
9: $\Sigma_\tau \leftarrow \Sigma_\tau \setminus \varphi$
10: $\Sigma_u \leftarrow$ UPWARD_DISCOVERY(φ)
11: $\Sigma_\tau \leftarrow \Sigma_\tau \cup$ SELECT_MINIMAL_FDS(Σ_u)
12: $\Sigma_{\tau+1} \leftarrow \Sigma_{\tau+1} \cup$ SELECT_MINIMAL_FDS(Σ_τ)
13: **return** $\Sigma_{\tau+1}$

sets $D_{\tau+1}$ and $D^-_{\tau+1}$ updating r at time $\tau+1$, and an instance of the RegExHashMap Γ_τ at time τ.

REXY starts updating the RegExHashMap by removing all tuples contained in $D^-_{\tau+1}$ (line 1), and by inserting those contained in $D_{\tau+1}$ (line 2). Then, it starts the discovery process by considering Σ_τ as the set of candidate FDs. In particular, REXY performs a discovery step in ascending order of the LHS cardinalities in Σ_τ, by selecting at each step from Σ_τ all the candidate FDs $\varphi : X \rightarrow A$ from Σ_τ with a specific LHS cardinality (line 3). For each φ, REXY checks if it is still valid at time $\tau+1$, according to the validation approach defined in Section 5.4.1 (line 4). If this is the case, REXY looks for other FDs on a lower lattice level,

by performing a downward discovery step (lines 5-7). Otherwise, REXY removes φ from the set of analyzed candidate FDs (line 9), and then it generates new candidate FDs on a higher lattice level, by performing an upward discovery step (line 10), filtering out the non-minimal candidate FDs (line 11). At the end, REXY returns all the FDs that are not minimal at time $\tau + 1$ with respect to the FDs already validated in the previous iteration step (lines 12 and 13).

Algorithm 9 provides the validation procedure of REXY. Let us consider a candidate FD $\varphi : X \rightarrow A$ at time $\tau + 1$, an instance of the RegExHashMap Σ_τ at time τ, and the set of the new tuples $D_{\tau+1}$ at time $\tau + 1$. The procedure allows REXY to create a new regular expression (RegEx) for each tuple t received at time $\tau + 1$, and to check the validity of each candidate FD on the updated instance. The validation procedure starts by considering each new tuple in $D_{\tau+1}$ (line 2). Then, for each of them, REXY defines a new RegEx according to the validation approach defined in Section 5.4.1. In particular, for each attribute, $B \in attr(R)$, if B does not belong to the attributes of X and it differs from attribute A, then the procedure adds a generic value to the final regex ρ_φ (lines 7-10). Otherwise, if B belongs to the attributes of X, then the procedure adds the corresponding value for the analyzed tuple $t[B]$ to the final RegEx ρ_φ (lines 12-13). However, if none of the previous cases occurs, it means that the attribute B is equal to A, so the procedure adds the negative look ahead of this value to ρ_φ (lines 14-15). After the creation of ρ_φ, REXY checks if there exists at least a tuple in the RegExHashMap Γ_τ that matches this RegEx. If so, φ is not valid on the updated instance at time $\tau + 1$, since there exists at least one pair of tuples that invalidates φ (lines 19-20). Otherwise, if no regex matches after updating the instance, it means that the candidate φ is valid at time $\tau + 1$ (line 21).

It is worth notice that, while the implementation of REXY considers several code optimizations and takes advantage of the efficient data structures defined above, for sake of clarity the pseudo-codes of Algorithms 8 and 9 do not provide details on such optimizations.

Algorithm 9 VALIDATION_REGEX

INPUT:

$\varphi : X \to A \to$ a candidate FD to validate
$D_{\tau+1} \to$ the set of new tuples for the dataset D at time $\tau + 1$
$\Gamma_\tau \to$ an instance of the RegExHashMap at time τ

OUTPUT:

True $\to$ If the FD is valid
False $\to$ Otherwise

```
1:  ρ_φ ← ∅
2:  for each t ∈ D_{τ+1} do
3:      ATTRS ← attr(R)
4:      i ← 0
5:      for each B ∈ attr(R) do                          ▷ RegEx creation
6:          if B ∉ X ∧ B ≠ A then
7:              if i ≠ 0 then
8:                  ρ_φ ← ρ_φ ∪ ([,][0 − 9]+)*
9:              else
10:                 ρ_φ ← ρ_φ ∪ ([0 − 9] + [,])*
11:         else
12:             if B ∈ X then
13:                 ρ_φ ← ρ_φ ∪ t[B]
14:             else if B == A then
15:                 ρ_φ ← ρ_φ ∪ (?!.*t[B]).+
16:             if i < |attr(R)| − 1 then
17:                 ρ_φ ← ρ_φ ∪ [,]
18:         i ← i + 1
19:     if Γ_τ.EXIST_MATCHING(ρ_φ) is True then
20:         return False
21: return True
```

5.4.3 *Experimental Evaluation*

In the following, we present experimental results concerning the performance of REXY in discovering FDs. In particular, the performed tests

show how the new proposed approach can improve time performances with respect to the incremental discovery algorithm INCREMENTAL-FD.

Implementation details. REXY algorithm has been developed in Java 15 and evaluated on an iMac Pro, with an Intel Xeon processor at 3.40 GHz, 36-core, and 128GB of RAM.

Datasets. Table 5.5 reports statistics on the real-world datasets[2] considered for the evaluation of REXY, including the corresponding execution times and memory consumption.

Evaluation process. For each dataset D we simulated an incremental scenario by executing REXY on the datasets D_i, $0 \leq i \leq |D|$, each derived by adding the *i-th* tuple of D to the dataset D_{i-1}, where D_0 is the empty dataset. This allowed us to analyze the performances of the validation process on each candidate FD. To this end, Table 5.5 reports the minimum, maximum, and average times of the validation process for each dataset. It also reports the number of resulting FDs, the number of performed validations, and the memory peaks registered at the end of the last execution of REXY on each dataset.

Analysis of the results. The results highlight that the average execution times are almost always less than 1 second, except for the *Zoo*, *Letter*, *Lymphography*, *Hepatitis*, and *Parkinson* datasets. The higher execution times are probably due to the large number of FDs invalidated during the discovery process, which leads to a large number of new candidate FDs to be validated. In fact, regarding the number of validations that REXY performed, we can observe that often it is high, and it tends to increase when the number of columns and/or rows increases. Nevertheless, the memory peaks were low, except for the *Hepatitis* dataset. This could be due to the high number of different numeric values of some of its attributes, yielding a big number of value combinations.

Concerning the validation process, the average times of this process are almost always less than 30 milliseconds per extracted FD, and the maximum time almost never exceeds 500 milliseconds, except for the *Credit Screening*, *Letter*, and *Parkinson* datasets. Despite these time peaks, the

2 All datasets adopted for this experimental evaluation are available on a GitHub repository: https://github.com/DastLab/TestDataset

#	Dataset	Cols [#]	Rows [#]	FDs [#]	Execution (Avg) [ms]	Validations [#]	(Min) [ms]	(Max) [ms]	(Avg) [ms]	Memory [MB]
1	Hayes-roth	5	132	4	1	648	0.88	9.71	0.61	103
2	Iris	5	150	4	1	777	0.64	13.51	0.63	95
3	Balance-scale	5	625	1	1	1343	0.39	12.63	0.47	100
4	Bupa	6	345	16	1	7006	0.18	7.29	0.14	40
5	Appendicitis	7	107	72	4	7891	0.39	12	0.46	40
6	Chess	7	28056	1	0.88	47257	0.21	9.16	0.85	66
7	Ecoli	8	336	37	16	13360	0.27	16.81	0.43	128
8	Cars	9	406	67	10	26080	0.48	10.38	0.87	43
9	Tic-tac-toe	9	958	9	2	14652	0.37	11.43	0.60	42
10	Yeast	9	1484	37	4	69618	0.36	10.76	0.37	44
11	Abalone	9	4148	137	164	710156	0.42	213	11.12	114
12	Nursey	9	12960	1	7	59160	0.35	49	9.43	51
13	Glass	10	214	124	23	28950	0.51	28.16	1.57	42
14	Cmc	10	1473	1	2.97	17832	0.66	13.51	0.81	43
15	Breast-cancer	11	699	46	11	57336	0.32	14.12	0.66	66
16	Fraud-detection	11	28000	48	332	1723242	0.17	483	49.28	168
17	Poker-hand	11	264000	1	19	2401240	0.93	96.67	6.21	206
18	Heart-failure	12	299	463	68	139463	0.13	6.65	2.04	169
19	Echocardiogram	13	132	538	142	66115	0.31	86	1.98	59
20	Bridges	13	108	142	80	34111	0.46	45	1.23	75
21	Wine	14	178	1374	25	171074	0.69	126	25.86	137
22	Australian	14	690	535	331	503115	0.53	199	3.06	124
23	Credit-screening	15	690	761	550	622360	0.76	561.37	7.77	686
24	Adult	15	32560	60	281	3747386	0.83	23	12.53	803
25	Tax	15	6848	310	680	2378734	0.76	427	16.12	127
26	Zoo	17	101	231	4848	592278	0.66	358	4.13	2010
27	Letter	17	20000	61	3799	592278	0.82	1049	7.81	2592
28	Lymphography	19	147	2730	71844	5156312	14.26	34	20.34	4921
29	Hepatitis	20	155	8250	194553	4528681	11.58	170	126.57	48878
30	Parkinson	24	195	1724	524195	874789	16.28	16470	10492.83	127

Table 5.5: Characteristics of the considered real-world datasets and REXY performances on them.

average validation times demonstrate that such values can be considered as outliers (except for *Parkinson*). Figure 5.9 shows the variability of the validation times for each dataset considered in our evaluation. It is worth noting that, the number of rows and columns in a dataset does not affect

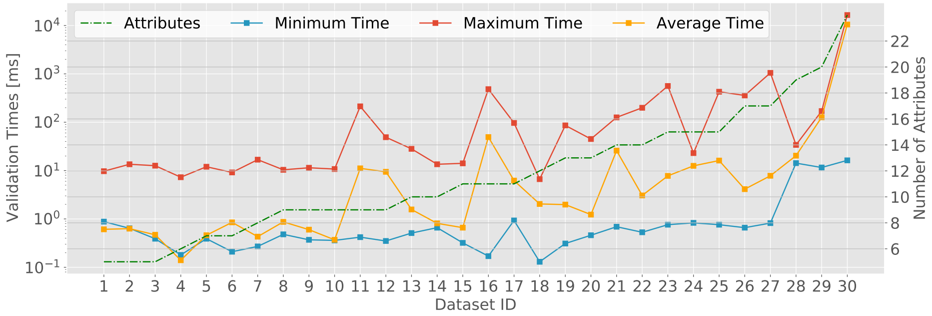


Figure 5.9: Validation times for each dataset.

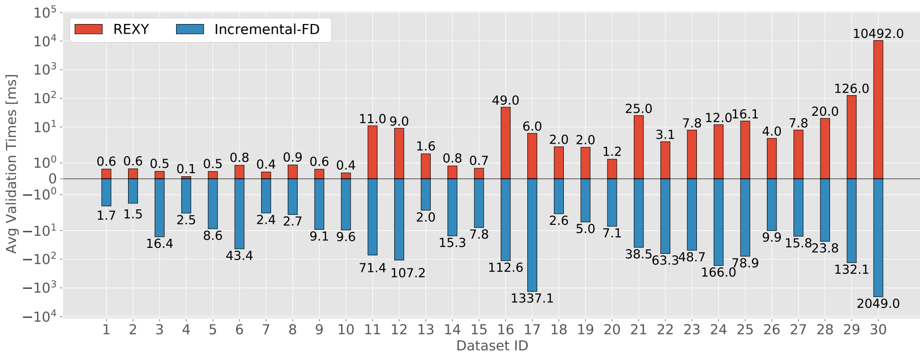


Figure 5.10: Validation times of REXY and Incremental-FD.

the average execution time for validating the candidate FDs, although the complete execution times increase on average when the number of attributes increases.

Finally, we performed a comparative evaluation of REXY with respect to the algorithm Incremental-FD presented in Section 5.3. Since both the algorithms use the same search strategy, the comparison allowed us to highlight the improvements in terms of execution times and memory consumption of the validation strategy proposed in REXY. Figure 5.10 shows the average validation times of both algorithms on the 30 considered datasets. We can notice that REXY outperforms Incremental-FD by several orders of magnitude, ranging from 2 to 7, except for the

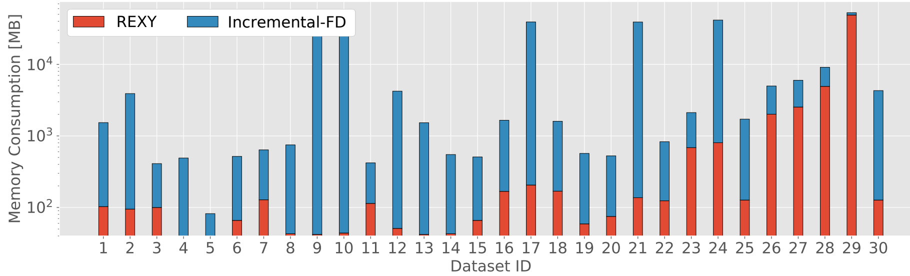


Figure 5.11: Memory consumption of REXY and Incremental-FD.

Parkinson dataset (number 30) for which Incremental-FD takes advantage of its caching strategy that enables it to reuse the computational results performed in previous iterations. Concerning memory consumption, Figure 5.11 highlights that REXY requires less memory with respect to Incremental-FD, even though this gap is less for most of the datasets with a high number of columns. In particular, for *Hepatitis* (number 29) the two algorithms require almost the same amount of memory.

5.5 COD3: CONTINUOUS DISCOVERY OF FD FROM DATA STREAMS

This section presents a new FD discovery algorithm, named COD3 (Continuous Discovery of FD from Data Streams), which allows to continuously infer and update FDs holding on a data stream, as the data are read from it. To the best of our knowledge, COD3 represents the first proposal to use a non-blocking architectural model for this problem. In what follows, we first provide an overview of the discovery strategy underlying COD3, its architecture, and its new validation methodology.

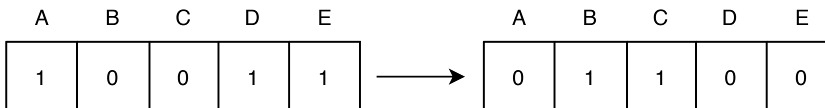


Figure 5.12: Binary vector representation of the FD $ADE \rightarrow BC$.

5.5.1 *Methodology*

The proposed algorithm follows the column-based strategy, which considers the search space as an attribute lattice, where each node contains a unique set of attributes directly connected to supersets or subsets of them (Figure 5.2).

Similarly to the algorithm INCREMENTAL-FD, COD3 uses a binary vector representation that permits to encode FDs in a compact way (see Figure 5.12). It consists of two binary vectors v_X and v_Y, representing the LHS and RHS, respectively, of an FD. Such binary vectors contain as many elements as the number of attributes in a relation schema R, so that if an attribute appears in the LHS (resp. RHS) of an FD the element of v_X (resp. v_Y) associated to it will contain a 1. In this way, all the candidate FDs $X \rightarrow A$ sharing the same LHS are compressed in a single pair of vectors (v_X, v_Y), where v_Y contains a 1 for all the attributes determined by X.

Figure 5.13 provides an overview of the discovery strategy underlying COD3. It starts by considering a tuple read from the stream and the set Σ_τ of all minimal FDs holding at time τ. Notice that, Σ_τ becomes the set of candidate FDs, and will be processed through a *linked map* (see Section 5.3), which permits to perform a discovery process in ascending or descending order of the LHS cardinality.

As discussed in Section 5.1.2, when a sliding window is enabled, the algorithm can read a novel tuple or an expired one according to the sliding window mechanism. COD3 manages this type of mechanism through the *Negative Tuple Approach (NTA)* [67], which yields the tuple expiration notification on the stream. Thus, expired tuples will be read from the stream as particular tuples, i.e., t^-. Nevertheless, from a theoretical point of view, no novel issue might be considered with expired tuples. However, the above described issues must be taken into account in an alternative way, i.e., by also considering the set of maximal NON-FDS, Q_τ, which can be always determined by using the set of minimal FDS Σ_τ at time τ [140]. Thus, when COD3 receives an expired tuple t^-, its underlying strategy firstly calculates the set Q_τ of maximal NON-FDS at time τ, which is in turn

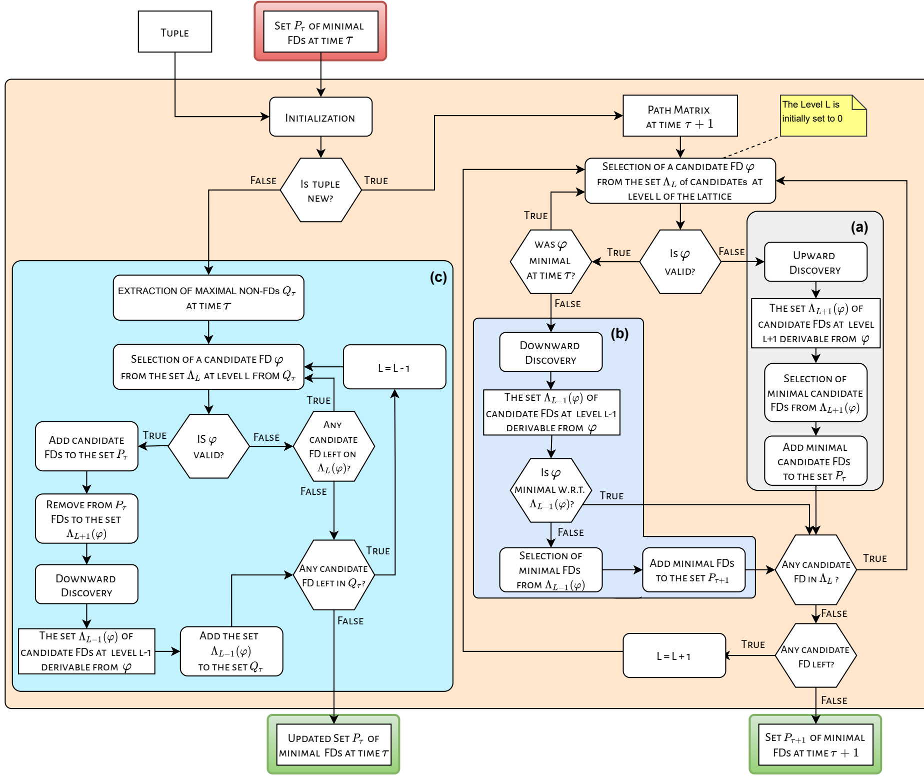


Figure 5.13: Overview of the incremental discovery strategy of COD3.

processed to a linked map[3]. Thus, COD3 performs a discovery step by processing the NON-FDS in descending order of their LHS cardinality. It is important to notice that, when a tuple expires no FDS will be invalidated but some NON-FDS can become valid. To this end, the strategy underlying COD3 first checks if there exists at least one NON-FD that has become valid after the window sliding step. More formally, for each candidate FD $\varphi : X \rightarrow A$, if φ is valid after the window sliding step, then COD3 checks its minimality with respect to the already validated FDS in the set Σ_τ, and removes those that have become not minimal, i.e., those that can be inferred by the newly validated one (Figure 5.13c). Then, COD3 performs

3 A procedure to compute the set of maximal NON-FDS is shown in [140].

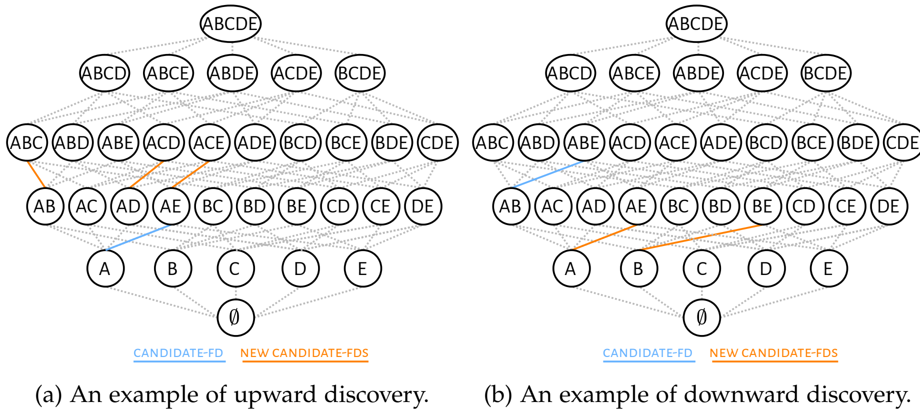
Sepal Length (A)	**Sepal Width (B)**	**Petal Length (C)**	**Petal Width (D)**	**Class Label (E)**
4.8	3.0	1.4	0.1	Iris-setosa
4.8	3.0	1.4	0.3	Iris-setosa
5.0	2.0	3.5	1.0	Iris-versicolor
5.0	2.3	3.3	1.0	Iris-versicolor

Table 5.6: Snippet of *iris* to illustrate the discovery strategy.

a downward discovery to verify if there exist other NON-FDS at lowest lattice levels that are minimal with respect to the already considered ones. The output of this step is the updated set Σ_τ of minimal and valid FDS at time τ, after processing a window sliding step.

In the case of a novel tuple t read from the stream, the strategy underlying COD3 also considers a *path matrix* evaluating the impact of the new tuple on the already processed ones, as discussed in the following. COD3 considers the set Σ_τ of minimal FDS at time τ as candidates at time $\tau + 1$, and it performs the discovery step by processing them in ascending order of their LHS cardinality. Only for the first tuple t read from the stream at time 0, the set of candidates Σ_τ will include all the FDS associated to the edges connecting lattice level 1 nodes to level 2 ones, that is, those connecting nodes with one attribute to those with two attributes. In particular, for each candidate FD φ at time $\tau + 1$, the process tries to validate it, and if it does not hold (Figure 5.13a), it generates new candidate FDS by considering the direct supersets of its LHS. This step is named *upward discovery* (Figure 5.3a). Vice versa, if φ is valid (Figure 5.13b), and it is not contained in Σ_τ (i.e., it was not minimal at time τ), then it undergoes the minimality check, in which COD3 evaluates all the direct subsets of its LHS on the previous lattice level. Such a step is named *downward discovery* (Figure 5.3a).

At each subsequent iteration, the process verifies the *invalidation* and *refutation of minimality* issues defined in Section 5.1.2. In particular, let φ:



(a) An example of upward discovery. (b) An example of downward discovery.

Figure 5.14: An example of lattice for *Iris* dataset.

$X \rightarrow A$ be the analyzed candidate FD, based on the result of validation, COD3 generates the set of candidate FDs by performing the following steps: for each attribute B of the dataset not contained in the LHS of φ, generate a candidate FD $X \cup B \rightarrow A$ (upward discovery step), and for attribute $B \in X$ generates a candidate FD $X \setminus B \rightarrow A$ (downward discovery step). However, both steps exploit a refutation of minimality check strategy in order to discard new candidates that are not minimal. The output of this step is a new set $\Sigma_{\tau+1}$ of minimal and valid FDs at time $\tau + 1$.

Example 1. Let us consider the snippet of the *Iris* dataset[4] shown in Table 5.6. If the FD $B \rightarrow C$ is not valid at time $\tau + 1$, then one or more candidate FDs on the next lattice level could be valid (e.g., $AB \rightarrow C$, $AD \rightarrow C$, and $AE \rightarrow C$) as shown in Figure 5.14a. Vice versa, if the FD $AB \rightarrow E$ is valid at time $\tau + 1$, then it is necessary to check if one or more valid FDs on the previous lattice level have not already been validated (e.g., $A \rightarrow E$ and $B \rightarrow E$). If so, the FD $AB \rightarrow E$ is valid but not minimal (Figure 5.14b).

4 https://archive.ics.uci.edu/ml/datasets/iris

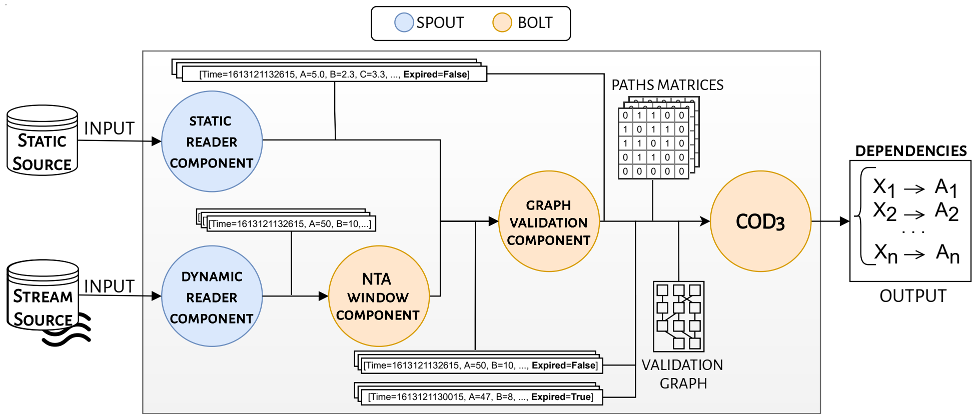


Figure 5.15: The COD3 pipeline.

5.5.1.1 *The pipeline of* COD3

COD3 is a novel algorithm conceived to enable an FD discovery process suitable for data streams. It relies on a non-blocking strategy (also known as pipeline processing), in which it is not necessary to wait for the complete execution of the process on a tuple to start processing the next tuple.

Figure 5.15 shows the components of the COD3 pipeline. It consists of two components representing data stream sources enabling the reading of data from static or dynamic sources: *Static Reader Component* and *Dynamic Reader Component*, respectively. The former reads data from any database instance (e.g. a real-world dataset), whereas the latter reads data from external data providers (e.g. Sensors, Social Networks, Streaming APIs, and so forth). Both these components contain a mapping mechanism transforming information into a stream of tuples, by splitting each tuple into a list of values, where each value is assigned a name.

In general, COD3 uses a single stream source type at a time whenever it considers heterogeneous sources. Each component reads the data and sends them to the next component devoted to the execution of the preprocessing steps in order to start the discovery process. In particular, in case of a dynamic source, upon reading a new tuple from the stream

COD3 exploits a negative tuple approach (NTA) to check for expired tuples, also integrating a sliding window mechanism within an *NTA Window Component*, which is responsible for reading new tuples from the source. As described above, this mechanism allows COD3 to define a time interval within which the validity and minimality of the discovered FDs are guaranteed. Thus, the window scrolls according to the time interval, possibly causing the expiration of some tuples (denoted by t^{-}), which will be sent on the stream before processing the new ones, since their expiration might make valid previously invalidated FDs. Notice that, the sliding window mechanism is activated according to the COD3 execution settings. By default, COD3 will perform the discovery process by only considering tuple insertions. The *NTA Window Component* sends both new and expired tuples to the *Graph Validation Component*, which in turn updates the data structures (e.g., the validation graph) based on their values. The *Graph Validation Component* can also receive tuples from static sources (e.g., real-world datasets), by means of a *Static Reader Component*, but in this case no expired tuple is considered.

At the end of the process, the *Graph Validation Component* outputs the updated data structures and sends the read tuples to the next component that is responsible for executing the discovery process. The latter executes the discovery process according to the strategy described in Figure 5.13, and extracts all the FDs holding at time $\tau + 1$. Notice that, while the discovery algorithm updates the set of minimal FDs, all the previous components continue to perform their tasks by processing the new tuples received from the stream.

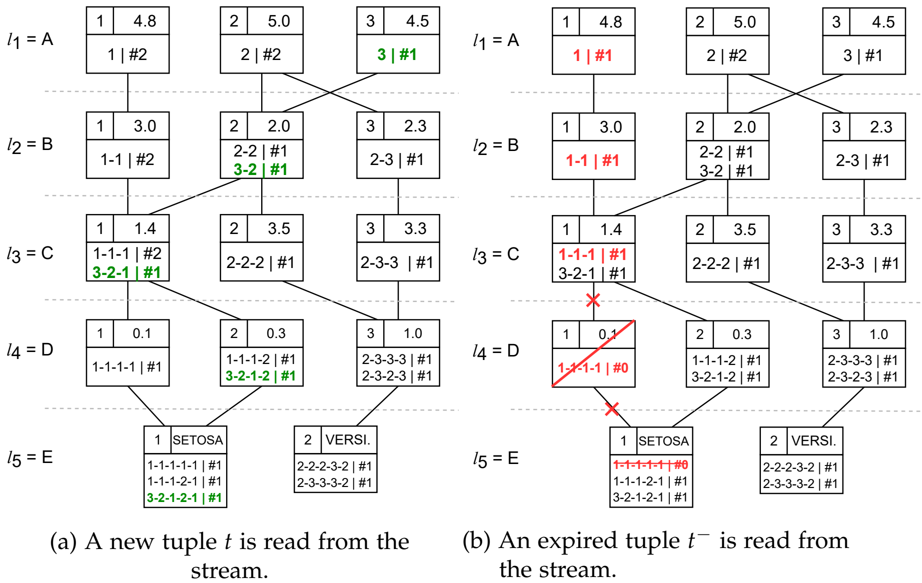
The pipeline behind COD3 relies on the Apache Storm framework[5], which is one of the most widely used technologies for managing data streams, mainly due to its adaptability. Apache Storm manages sequences of raw tuples continuously received from data providers as a collection of key-value items, and uses several control mechanisms to minimize the loss of tuples by automatically reintroducing them in the stream. The architecture of an application based on Storm is modeled as a directed acyclic graph (DAG), named *topology*, which represents a graph

5 https://storm.apache.org/

of independent execution modules, where nodes are some individual components, and edges represent the data passing through nodes. The components can be of type *spout* or *bolt*. A spout normally reads data from an external data source (e.g. messages, database updates, and any other static or dynamic data source), and inserts tuples into the topology. Instead, a bolt receives a set of tuples from its input stream, performs some computations on them, and then optionally inserts a new set of tuples into its output stream. A bolt processes tuples in order to send them to other bolts for further processing steps.

5.5.2 *Graph-based* FD *Validation*

The validation process underlying COD3 relies on a new graph structure, named *validation graph*, which stores lightweight references to the tuples that continuously arrive from the stream. More formally, let M be the number of attributes of a relation schema R, a validation graph G is a



(a) A new tuple t is read from the stream.

(b) An expired tuple t^{-} is read from the stream.

Figure 5.16: An example of validation graph.

structure with M levels, where each node at level l_i is associated to a value for attribute A_i that is instantiated in the stream, whereas each edge connects nodes on adjacent levels only if they represent attribute values appearing at least once in the same tuple read from the stream. Thus, the nodes at level l_i in G contain the value distribution of attribute A_i in a stream of tuples instantiating R.

A node g of G can be defined as a quintuple $g = (A_i, a_i, id, l_i, \Gamma)$, where A_i is an attribute of R, a_i one of its values instantiated in the stream, id is an identifier to distinguish a_i from all other values instantiated in the stream for attribute A_i, l_i the level of A_i, and Γ the set of all distinct paths connecting g to a node at level 1. More specifically, an element of the set Γ is a pair containing a path and a counter, which states that the path has occurred that number of times. Clearly, for a node at the l_i-th level, Γ will contain paths of length i.

In what follows, we explain how the validation graph is modified according to the type of tuple read from the stream, i.e., a new tuple or an expired one, t and t^-, respectively.

When a new tuple t is read from the stream, the existing validation graph can be updated as follows:

1) for each level l_i, $1 \leq i \leq M$, a new vertex might be added to l_i if $\Pi_{A_i}(t)$ has never occurred before in the stream;

2) for each level l_i, $1 < i \leq M$, a new edge might be added between node id_{i_j} at level l_i and node id_{i-1_k} at level l_{i-1} only if the projection $\Pi_{A_{i-1},A_i}(t)$ has never occurred in the stream.

3) for each level l_i, $1 < i \leq M$, and for each of its nodes id_{i_j}, a new path of length i might be added to its Γ set only if the projection $\Pi_{A_1,\ldots,A_i}(t)$ has never occurred in the stream.

Example 2. Let us consider the validation graph derived after reading the tuples shown in Table 5.6 from a stream, and let us suppose that the following tuple is successively read:

t = [A="4.5", B="2.0", C="1.4", D="0.3", E="Iris-setosa", Expired="False"]

The resulting graph is shown in Figure 5.16a. Since only the value 4.5 for attribute A has never occurred before in the stream, only a new node $(A, 4.5, 3, 1, \{3| \#1\})$ is added at level 1; since the values of $\Pi_{A,B}(t)$ have never occurred before, such new node is connected to node with $id = 2$ at level 2, which is in turn connected to node with $id = 1$ at level 3, because also the values of $\Pi_{B,C}(t)$ have never occurred. Finally, the new paths added to the Γ sets of some nodes are highlighted in green.

When an expired tuple t^- is read from the stream, the existing validation graph can be updated as follows:

1) for each level l_i, $1 \leq i \leq M$, a vertex might be removed from l_i if $\Pi_{A_i}(t^-)$ has at most one occurrence, i.e., it is no longer valid in the stream;
2) for each level l_i, $1 < i \leq M$, an edge might be removed between node id_{i_j} at level l_i and node id_{i-1_k} at level l_{i-1} only if the projection $\Pi_{A_{i-1},A_i}(t^-)$ has at least one occurrence, i.e., it is no longer valid in the stream;
3) for each level l_i, $1 < i \leq M$, and for each of its nodes id_{i_j}, a path of length i might be removed from its Γ set only if the projection $\Pi_{A_1,\ldots,A_i}(t^-)$ has at least one occurrence, i.e., it is no longer valid in the stream.

Example 3. Let us consider the validation graph derived after reading the tuples of Table 5.6 and the one of *Example 2* from the stream (Figure 5.16a), and suppose that before processing new tuples the window scrolls and the following tuple is no longer valid:

t^- = [A="4.8", B="3.0", C="1.4", D="0.1", E="Iris-setosa", Expired="True"]

The resulting graph is shown in Figure 5.16b, whose updates are highlighted in red. Starting from the bottom of the validation graph, since the values of $\Pi_{A,B,C,D,E}(t^-)$ had one occurrence in the stream, the path $\{1-1-1-1-1| \#0\}$ is removed from the Γ of the node with $id = 1$ at level 5. Moreover, given that only the node with $id = 1$ and value 0.1 at level 4 had one path in Γ with one occurrence, the node $(D, 0.1, 1, 4, \{1-1-1-1| \#0\})$ is removed from level 4. Instead,

since the values of t^- have occurred more than ones on the previous levels by means of other tuples read from the stream, it is necessary to only decrease the number of occurrences on the correspondent nodes in those levels. In particular, the expiration of t^- entails a decrease of the counter for: *i*) $\Pi_{A,B,C}(t^-)$, i.e., $(C, 1.4, 1, 3, \{1-1-1| \#1\})$; *ii*) $\Pi_{A,B}(t^-)$, i.e., $(B, 3.0, 1, 2, \{1-1| \#1\})$; and *iii*) $\Pi_A(t^-)$, i.e., $(A, 4.8, 1, 1, \{1| \#1\})$.

As mentioned above, in the case of reading a new tuple from the stream, COD3 exploits an additional structure, named *path matrix*, to further optimize the validation process. The path matrix is a lightweight data structure containing information related to the new nodes and edges that are added to the validation graph after the insertion of a new tuple. It is used by COD3 to isolate cases in which the validation of an FD can be performed instantly. More specifically, when a new tuple t is read from a stream, COD3 creates a binary matrix of paths for each of its attribute values before updating the validation graph G.

Formally, let us consider a stream whose tuples contain M attributes. Upon reading a new tuple t from the stream, a path matrix $H(M+1, M+1)$ with the following properties will be built for t:

1) $H[0][0] = 0$;

2) $H[0][i] = H[i][0]$; $H[0][i] = 1$ if and only if $t[A_i]$ has never occurred in the stream for attribute A_i, 0 otherwise;

3) $H[i][j] = 1$ if and only if $\Pi_{A_i,A_j}(t)$ has never occurred in the stream for attributes A_i and A_j, 0 otherwise.

The path matrix allows the validation process to prune the search space by catching borderline cases. In what follows, we provide more details on the use of the path matrix for the validation process.

5.5.2.1 *Validation Process*

The validation process of COD3 exploits the validation graph to check the validity of candidate FDs both in case of insertion of new tuples and in case of expired ones. More specifically, let $\varphi : X \rightarrow A$ be a candidate

FD after the insertion of a new tuple t at time $\tau + 1$, the validation process considers the following four different cases:

- **Case 1.** φ is valid at time $\tau + 1$ if at least one node or an edge linking a pair of attributes in X has been created on G. As an example, let us consider the FD $AC \to E$ holding on the sample dataset shown in Table 5.6. Since the insertion of the tuple of *Example* 2 yields the creation of a new node for the attribute value $A = 4.5$, $H[1][A]$ ($H[A][1]$, respectively) will be set to 1 as shown in Table 5.7, and the candidate FD $AC \to E$ remains valid. The validation process for this FD is also shown in Table 5.8a, which represents a snippet of the dataset *iris* for the attributes A, C, and E, where the bottom tuple is the newly inserted tuple. Thus, since the value for the attribute A, e.g. 4.5, is new, the added tuple does not invalidate the FD $AC \to E$.
- **Case 2.** φ is not valid at time $\tau + 1$ if no new path has been added across attributes in X, and a new node for attribute A has been added to G. As an example, let $C \to A$ be an FD holding on the sample dataset shown in Table 5.6. Since the insertion of the tuple of *Example* 2 yields the creation of a new node for the attribute value $A = 4.5$, and a path from A to C, then the values of $H[1][A]$ ($H[A][1]$, respectively), and $H[A][C]$ ($H[C][A]$, respectively) are set to 1 (as shown in Table 5.7), invalidating the FD $C \to A$ (Table 5.8b).

		A	B	C	D	E
	0	1	0	0	0	0
A	1	0	1	1	1	1
B	0	1	0	1	1	1
C	0	1	1	0	0	0
D	0	1	1	0	0	0
E	0	1	1	0	0	0

Table 5.7: The path matrix after the insertion of the tuple reported in *Example* 2.

Sepal L. (A)	Petal L. (C)	Class (E)
4.8	1.4	Iris-setosa
4.8	1.4	Iris-setosa
5.0	3.5	Iris-versicolor
5.0	3.3	Iris-versicolor
4.5	1.4	Iris-setosa

(a) $AC \rightarrow E$

Sepal L. (A)	Petal L. (C)
4.8	1.4
4.8	1.4
5.0	3.5
5.0	3.3
4.5	1.4

(b) $C \rightarrow A$

Sepal W. (B)	Class (E)
3.0	Iris-setosa
3.0	Iris-setosa
2.0	Iris-versicolor
2.3	Iris-versicolor
2.0	Iris-setosa

(c) $B \rightarrow E$

Petal L. (C)	Petal W. (D)	Class (E)
1.4	0.1	Iris-setosa
1.4	0.3	Iris-setosa
3.5	1.0	Iris-versicolor
3.3	1.0	Iris-versicolor
1.4	0.3	Iris-setosa

(d) $CD \rightarrow E$

Table 5.8: Snippet of the dataset *iris* for some candidate FDs.

- **Case 3.** φ is not valid at time $\tau + 1$ if no new node or edge has been added for attributes in X, and no new node has been added for A, but at least one edge linking a node for an attribute in X to a node on A has been added to G. As an example, let us consider the FD $B \rightarrow E$ holding on the sample dataset shown in Table 5.6. Since the insertion of the tuple of *Example 2* yields the creation of a new edge between the existing attribute values $B = 2.0$ and $E =$ *Iris-setosa*, the value $H[B][E]$ ($H[E][B]$, respectively) will be set to 1, as shown in Table 5.7, invalidating the FD, as also highlighted in Table 5.8c.
- **Case 4.** If none of the above cases occurs, then it is necessary to check how the value paths of all the attributes in a candidate FD are linked to each other. The goal of this case is to verify if a path

connecting attributes represent values of at least one previously analyzed tuple. This is due to the fact that the way in which the tuple values are linked does not comply with the transitive property, which could not otherwise be verified if the validation graph does not store value paths on the attribute nodes. For these reasons, when all other cases do not occur, the candidate FD φ is valid at time $\tau + 1$ if and only if the projection of t on $X \cup A$ (i.e., $\Pi_{X,A}(t)$) forms a path contained into the Γ set associated to the deepest node among those related to the attributes in $X \cup A$. As an example, let $CD \rightarrow E$ be an FD holding on the sample dataset of Table 5.6, then after the insertion of the tuple of *Example 2* it is not necessary to generate any new edge connecting the nodes associated to the attributes of the FD (see sub-matrix composed of C, D, E in Table 5.7). Thus, let g_C, g_D, and g_E be the nodes associated to the attributes C, D, and E, respectively, then it is necessary to check if there exists at least one path in Γ_E, e.g., the set of paths at time τ of the deepest node among those associated to the attributes in $CD \rightarrow E$. In particular, since Γ contains paths of id values, it will be necessary to satisfy the pattern *{?-?-1-2-1}*, where the ? represents any possible value, and $1-2-1$ are the ids obtained from the projection $\Pi_{C,D,E}(t)$ of t on the attributes **C, D, E**. Therefore, since there exists an item in $\Gamma_E =$ [*{1-1-1-1-1}*,*{1-1-1-2-1}*] satisfying the pattern *{?-?-1-2-1}*, then $CD \rightarrow E$ is valid, as also highlighted in Table 5.8d.

Notice that, cases 1, 2, and 3 permit to perform the validation process by simply checking the path matrix. Instead, for case 4 it is necessary to analyze the paths of id values contained in the nodes. However, the validation process is restricted to the analysis of the set Γ of a single node. This strategy permits to quickly validate each candidate FD.

The above described validation process represents the case in which a new tuple is read from the stream. Nevertheless, when a sliding window is set according to a time interval, COD3 also manages a proper validation process when the tuple read from the stream is an expired one, i.e., t^-. In particular, let $\varphi : X \rightarrow A$ be a candidate FD, similarly to Case 4, the validation process requires to check how the value paths of all the

attributes in a candidate FD are linked to each other. However, in this case, it is necessary to explore all the nodes at level l_i of G, where i represents the index of the deepest level among those involved in $X \cup A$. Since possible violations of φ can reside in multiple nodes, it is necessary to check if there exist at least two distinct paths (t_1, t_2) within all paths Γ of all nodes in l_i, such that $\Pi_X(t_1) = \Pi_X(t_2)$ and $\Pi_A(t_1) \neq \Pi_A(t_2)$. As an example, let $B \rightarrow A$ be a candidate FD for the sample dataset of Table 5.6, after the expiration of the tuple of *Example 3* it is necessary to check the validation of φ. Thus, let $l_2 = B$ be the deepest level involved in φ, it is possible to verify that there exist the two paths $[\{2-2\}, \{3-2\}]$, among the paths Γ_B of all nodes in l_2, which invalidate φ.

5.5.3 *The* COD3 *Algorithm*

COD3 permits not only to update FDs when a new tuple is read from the stream, but also when a tuple expires as a consequence of a scrolling step of a sliding window. Thus, for sake of simplicity, we present the general procedure of COD3 in two parts, on the basis of the type of tuple read from the stream: a novel tuple or an expired one. Both the general procedures follow the strategy presented in Section 5.5.1.

The procedure of COD3 for processing a new tuple read from the stream is shown in Algorithm 10. Given Σ_τ the set of all FDs holding at time τ, a tuple t, a path matrix H related to t, and a validation graph G at time τ, COD3 starts by analyzing the candidate FDs with lowest LHS cardinality (line 1). Then, for each FD $X \rightarrow A$ holding at time τ, COD3 uses the `INSERTION_VALIDATION` process (line 3) to verify whether $X \rightarrow A$ still holds at time $\tau + 1$. This process is performed according to the validation process described in Section 5.5.2. Thus, if the analyzed FD is valid at time $\tau + 1$, the algorithm checks if there exist other FDs in the previous lattice level that have been already validated at time $\tau + 1$ (lines 4-8). In particular, for each candidate FD on the previous lattice level (`PREVCANDIDATES`), if none of them have already been validated, i.e. each of them cannot infer the analyzed FD (`INFERENCE`), then this is also minimal at time $\tau + 1$ (lines 6-8). Vice versa, if the analyzed FD is not

Algorithm 10 COD3 Algorithm

INPUT:
$\Sigma_\tau \rightarrow$ A set of minimal FDs at time τ
$t \rightarrow$ A new tuple t
$H \rightarrow$ A path matrix related to the tuple t
$G \rightarrow$ A validation graph holding at time τ
OUTPUT: $\Sigma_{\tau+1} \rightarrow$ A set of new valid and minimal FDs at time $\tau + 1$

```
1:  Σ ← P_τ
2:  for each X → A ∈ Σ do
3:      if INSERTION_VALIDATION(X → A, t, H, G) is valid then
4:          if X → A ∉ P_τ then
5:              L_{l−1} ← PREVCANDIDATES(X → A)
6:              for each Z → A ∈ L_{l−1} do
7:                  if not INFERENCE(Z → A) then
8:                      Σ ← Σ \ {X → A}
9:      else
10:         L_{l+1} ← NEXTCANDIDATES(X → A)
11:         for each W → A ∈ L_{l+1} do
12:             if not INFERENCE(W → A) then
13:                 Σ ← Σ ∪ {W → A}
14: Σ_{τ+1} ← Σ
15: return Σ_{τ+1}
```

valid at time $\tau + 1$, COD3 generates new candidate FDs at a higher lattice level (NEXTCANDIDATES), by discarding those that can be inferred from other FDs (INFERENCE) already validated at time $\tau + 1$ (lines 10-13). Notice that, the ordered discovery of the FDs allows COD3 to avoid multiple validations of some candidate FDs.

The procedure of COD3 for updating the set of candidate FDs Σ_τ whenever at least one expired tuple is read from the stream is shown in Algorithm 11. Given Q_τ the set of all NON-FDs at time τ, a tuple t, and a validation graph G at time τ, COD3 starts by analyzing the new candidate FDs from Q_τ in descending order of their LHS cardinalities (line 1-2). Then, for each candidate FD, the algorithm checks if it is valid after the expiration of t, by means of the EXPIRATION_VALIDATION process

(line 3). Thus, if the candidate FD is valid, COD3 first removes it from the set Q_τ, and then removes all the FDS on the next level from Σ_τ (NEXTCANDIDATES), i.e., those that can be inferred from the new validated one (lines 5-8), in order to ensure the minimality of the FDS in Σ_τ. The new FD is then added to Σ_τ (line 9). Successively, COD3 calculates the candidate FDS at the next lower level (PREVCANDIDATES), which will be added to the set of candidate FDS to be processed (lines 10). Vice versa, if the candidate FD is not valid, COD3 simply removes it from the set of candidate FDS (lines 11-12).

Similarly to the general procedure of COD3, and according to the validation strategies described in Section 5.5.2, also the validation procedure is divided into two strategies with respect to either the insertion or the expiration of a tuple, described in Algorithm 12 and 13, respectively.

Algorithm 11 COD3 Algorithm for an expired tuple

INPUT:
$Q_\tau \rightarrow$ A set Q_τ of NON-FDS at time τ
$t \rightarrow$ A new tuple
$G \rightarrow$ A validation graph holding at time τ
OUTPUT: $\Sigma_\tau \rightarrow$ Updated set of minimal FDS at time τ

```
1:  Σ ← Q_τ
2:  for each W → A ∈ Σ do
3:      if EXPIRATION_VALIDATION(W → A, G) is valid then
4:          Q_τ ← Q_τ \ {W → A}
5:          L_{L+1} ← NEXTCANDIDATES(W → A)
6:          for each Z → A ∈ L_{l+1} do
7:              if Z → A ∈ P_τ then
8:                  Σ_τ ← P_τ \ {Z → A}
9:          Σ_τ ← P_τ ∪ {W → A}
10:         Σ ← Σ ∪ PREVCANDIDATES(W → A)
11:     else
12:         Σ ← Σ \ {W → A}
13: return Σ_τ
```

Algorithm 12 `INSERTION_VALIDATION`

INPUT:
$X \to A \to$ An FD holding at time $\tau + 1$
$t \to$ A new tuple
$H \to$ A path matrix related to the tuple t
$G \to$ A validation graph holding at time τ
OUTPUT:
True $\to$ If the FD is valid
False $\to$ Otherwise

```
1:  if M.containsNewEdges() then
2:      for each Z ∈ X do                                   ▷ Case 1
3:          if H[0][Z] = 1 then
4:              return true
5:          for each W ∈ X do
6:              if Z ≠ W ∧ H[Z][W] = 1 then
7:                  return true
8:      if H[0][A] = 1 then                                 ▷ Case 2
9:          return false
10:     for each Z ∈ X do                                   ▷ Case 3
11:         if H[Z][A] = 1 then
12:             return false
13: W ← ∅, d ← -1
14: for each Z ∈ (X ∪ A) do
15:     if Z.depth() < d then
16:         W ← Z
17:         d ← Z.depth()
18: g ← G.getNode(W, t(W))                                  ▷ Case 4
19: for each p ∈ Γ_v do
20:     if p.contains(t[X ∪ A]) then
21:         return true
22: return false
```

In particular, in case of tuple insertion, given an FD $X \to A$, a new tuple t, a path matrix M related to the tuple t, and a validation graph G, Algorithm 12 implements the FD validation method by exploiting the path

Algorithm 13 EXPIRATION_VALIDATION

```
INPUT:
    X → A → An FD holding at time τ
    G → A validation graph holding at time τ
OUTPUT:
    True → If the FD is valid
    False → Otherwise
1: i ← -1
2: for each Z ∈ (X ∪ A) do
3:   if Z.depth() < i then
4:     i ← Z.depth()
5: V ← G.getNodesByDepth(i)
6: for each g ∈ V do
7:   if |Γ_v| > 1 then
8:     for each p ∈ Γ_v do
9:       for each h ∈ Γ_v do
10:        if p ≠ h then
11:          if Π_X(p) == Π_X(h) ∧ Π_A(p) ≠ Π_A(h) then
12:            return false
13: return true
```

matrix and the validation graph. First of all, it is necessary to check if the path matrix related to the tuple t contains at least one new node/edge (line 1). If this is the case, it is possible to apply one of the cases 1, 2, or 3 defined above. More specifically, if the new tuple t has generated at least one node/edge according to Case 1, then $X \rightarrow A$ is valid at time $\tau + 1$ (lines 2-7). However, if Case 1 does not occur and attribute A generates a new node, then $X \rightarrow A$ is not valid according to Case 2 (lines 8-9). Otherwise, the algorithm checks whether the values in X are already linked to the values in the RHS of the analyzed FD, according to Case 3 (lines 10-12), yielding the invalidation of the candidate FD. If none of the above cases occurs, the algorithm checks whether $\Pi_{X,A}(t)$ already exists on the validation graph, and only in this case the candidate FD remains valid (lines 13-21). More specifically, it identifies the deepest

node g between the attributes in $X \cup A$ (lines 13-17), which allows the algorithm to validate the candidate FD by searching for a path in Γ_g containing the values in $X \cup A$, according to Case 4 (lines 18-21).

On the other hand, in case of tuple expiration, given an FD $X \rightarrow A$ and a validation graph G, Algorithm 13 implements the FD validation method by exploiting the validation graph. First of all, it is necessary to find the level l_i of the deepest node g between the attributes in $X \cup A$ (lines 1-4). Then, COD3 extracts all the nodes at level l_i from the validation graph G (line 5), and for each of them it checks if the node contains at least two distinct paths (line 7). If true, COD3 compares each pair of distinct paths, and checks whether there exists at least a pair of paths p and h, respectively, such that $\Pi_X(p)$ equals $\Pi_X(h)$, and $\Pi_A(p)$ differs from $\Pi_A(h)$ (lines 8-12), which yields $X \rightarrow A$ not to be valid at time τ (line 12). If none of the nodes at level i invalidate $X \rightarrow A$, then the FD continues to be valid at time τ (line 13).

5.5.4 *Experimental Evaluation*

In this section, we describe the evaluation of COD3 on several real-world datasets and real-time streams, by also providing a qualitative analysis of the metadata extracted from a sensor-based data stream, with the aim of analyzing how metadata evolve over time.

Implementation details. COD3 has been developed in Java 12, and it has been integrated into Apache Storm 2.1.0. Furthermore, COD3 exploits the pipeline programming model of Apache Storm in order to guarantee suitable performances, continuous processing, and a trade-off among consistency, speed, and durability.

Hardware and Datasets. The experiments have been executed on an iMac Pro with an Intel Xeon CPU at 3.20 GHz, 18-core, and 128GB of memory, running macOS Mojave 6.4 and OpenJDK 12.0.2 as Java environment. The experiments were performed on several real-world datasets[6], previously used for evaluating FD discovery algorithms. Table 5.9 shows the characteristics of the evaluation datasets.

6 https://archive.ics.uci.edu/ml/datasets.php

Evaluation process. In our experimental session, we performed two different types of tests to evaluate COD3 on static and dynamic sources. In the first experiment, we simulated a scenario of continuous tuple insertions by transforming datasets into dynamic sources through the COD3 pipeline components. Although in this kind of experiment COD3 is not used for the purpose it has been conceived, we considered it in order to perform a sort of comparative evaluation with respect to well-known FD discovery algorithms, since to the best of our knowledge there is no similar algorithm capable of directly extracting FDs from data streams. In particular, we compared COD3 with the FD discovery algorithms HyFD [128] and DynFD [140], which focus on the discovery of FDs from static and dynamic datasets, respectively. Instead, in the second experiment, we evaluated the effectiveness of COD3 on a data stream of sensors provided by the *AQICN portal*[7].

5.5.4.1 *Performances of* COD3 *on real-world datasets*

Our first experiment measured the execution times of COD3 on different real-world datasets, which are mapped into a continuous stream of data by following the strategy described in Section 5.5.1.1. The considered datasets have a different number of rows and columns in order to highlight how the COD3 execution times vary according to such parameters. In our test, we evaluated the execution times of COD3 by considering the first tuple and the initial runs of the algorithm, up to the last run on the last tuple.

Analysis of Results. Figure 5.17 summarizes the time performances of COD3 for each dataset, by means of boxplots built on the distribution of execution times per tuple. The figure also reports the memory peaks reached by COD3 on each execution. As we can see, the median values of the execution times are almost always less than 10 milliseconds per tuple, except for some of the biggest datasets. In particular, for *Sonar* and *Gas-sensors* the median values fall in the range $[1 - 10]$ seconds, whereas on the *Movement-libras* dataset exceeds other execution times, mainly due to the thousands of FDs to be validated in many executions.

7 https://aqicn.org/

Dataset	Cols [#]	Rows [#]	FDs [#]
Chess	7	28056	1
Abalone	9	4148	137
Electricity	9	45312	61
Poker-hand	11	264027	1
Echocardiogram	13	132	538
Tsa-claims	13	25023	129
Adult	14	32562	60
Fd-reduced	15	250000	4908
Ncvoter	19	1001	3179
Lymphography	19	148	2730
Parkinsons	24	195	1724
MoCap Postures	38	78095	4094
Sonar	60	208	97750
Movement-libras	91	360	2473105
Gas-sensors	128	4000	302705

Table 5.9: Characteristics of the considered real-world datasets.

In general, we notice that the execution times of COD3 present small distributions (upper and lower quartiles), even though some outliers occurred, especially with datasets containing many tuples.

With respect to memory peaks, the results show that no relationship can be derived between the memory load and the dataset characteristics in terms of the number of rows and columns. However, we can observe that the memory peaks slightly depend on the amount of discovered FDs. For instance, the worst memory loads occur for datasets exceeding two thousand of holding FDs. More specifically, the memory bound is related to the number of invalidations caused by the arrival of new tuples, which more likely occur as the number of holding FDs increases.

In general, COD3 works extremely well when the insertion of new tuples yields few invalidations. In fact, since the discovery process is performed level-by-level, when one or more FDs are invalidated, COD3 considers new FD candidates from the next level of the invalidated ones.

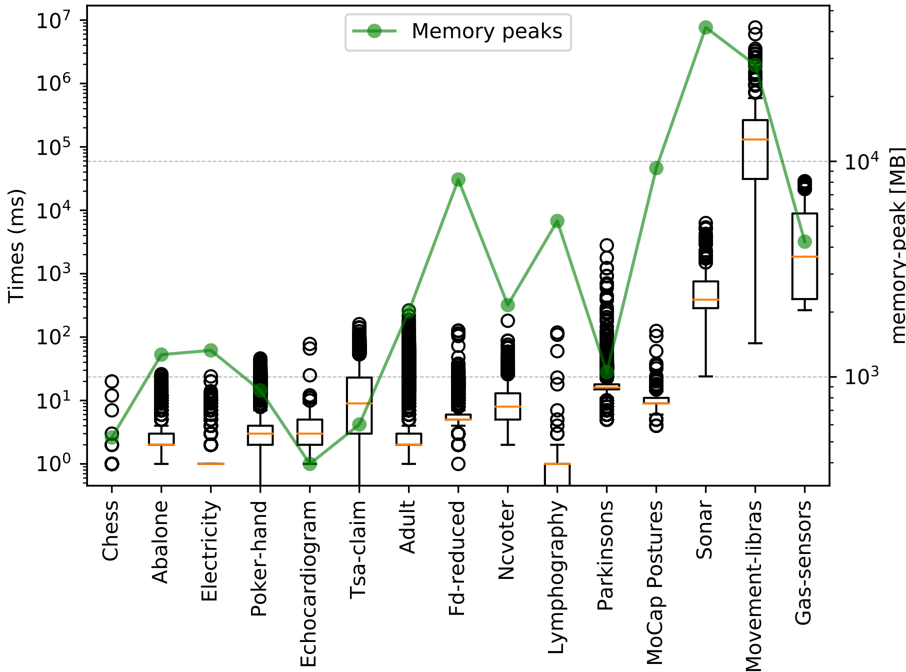


Figure 5.17: Performances of COD3 over real-world dataset.

Figure 5.18 reports the average execution times of the validation process for each FD upon the insertion of a tuple (green line), also compared to the average number of validations performed for each of the four cases (colored bars). The results highlight that the average time is always less than 1 millisecond per FD, but in most cases it does not exceed 0.2 milliseconds. The only exceptions are for *Poker-hand*, *Tsa-claims*, and *Sonar* datasets.

Concerning the number of times a specific case is executed during all validations, we can notice that for each dataset the majority of validations only exploit path matrices (Cases 1, 2, and 3), which makes the process faster. Particularly interesting are the results achieved on the two biggest datasets, i.e., *Sonar* and *Gas-sensors*, where the validation process instanti-

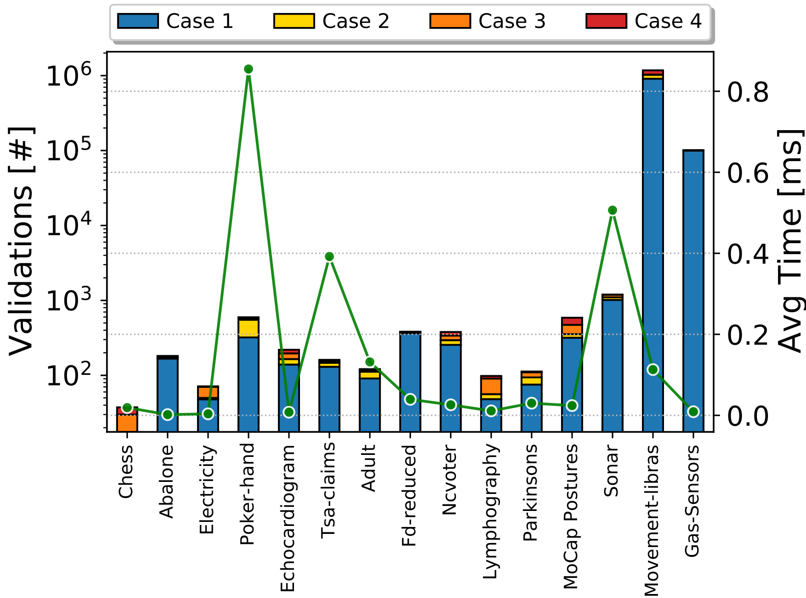


Figure 5.18: Number of validations for each case defined in Algorithm 12.

ates Case 4 only a few times, yielding extremely low average execution times, despite a huge number of holding FDs.

5.5.4.2 *Comparative evaluation*

As a further experiment, we compared the execution performances of COD3 to those of one of the best-performing non-incremental discovery algorithm, namely HyFD [128], and an analogous incremental one, namely DynFD [140]. In particular, HyFD is a static FD discovery algorithm that combines approximation techniques with several validation strategies, in order to discover the set of all minimal FDs holding on a static dataset. In particular, we analyzed all the conditions in which COD3 under- or out-performs such a static discovery algorithm. To this end, we gradually scaled up the size of the dataset, starting with a dataset containing one tuple and adding one tuple at a time, each time executing HyFD

on the augmented dataset. Instead, DynFD is an incremental discovery algorithm, which extends HyFD with the possibility of updating the set of holding FDs in accordance with insert, delete, and update operations collected in batch mode. In order to compare COD3 and DynFD, we set up an insertion operation in the batch file for each tuple inserted in the dataset. More specifically, we started with a dataset containing only one tuple, and simulated the insertion of one tuple at a time. To execute COD3 over static datasets, we transformed the considered datasets into a continuous data flow, according to the pipeline components in Section 5.5.1.1.

Figure 5.19 shows the results of the comparative evaluation in terms of the variability of average execution times (plot at the top) and the memory load (plot at the bottom). In particular, the results show that COD3 is almost always faster than HyFD as the number of processed tuples grows, especially on the *Poker-hand*, *Fd-reduced*, and *MoCap Postures* datasets. Furthermore, we can notice that COD3 has poor performances during the first runs on *Echocardiogram*, *Ncvoter*, *Parkinsons*, *Lymphography*, and *Sonar* datasets. This is probably due to the fact that the number of validations and invalidations is quite high when the datasets contain few tuples. Moreover, COD3's poor performances on the *Movement-libras* and the *Gas-sensors* datasets are probably due to the fact that these datasets have a high number of columns and many FDs are discovered during early initial runs. An exception is the *Sonar* dataset, in which the execution times of both algorithms appear similar as the number of tuples increases, even though *Sonar* represents one of the biggest datasets.

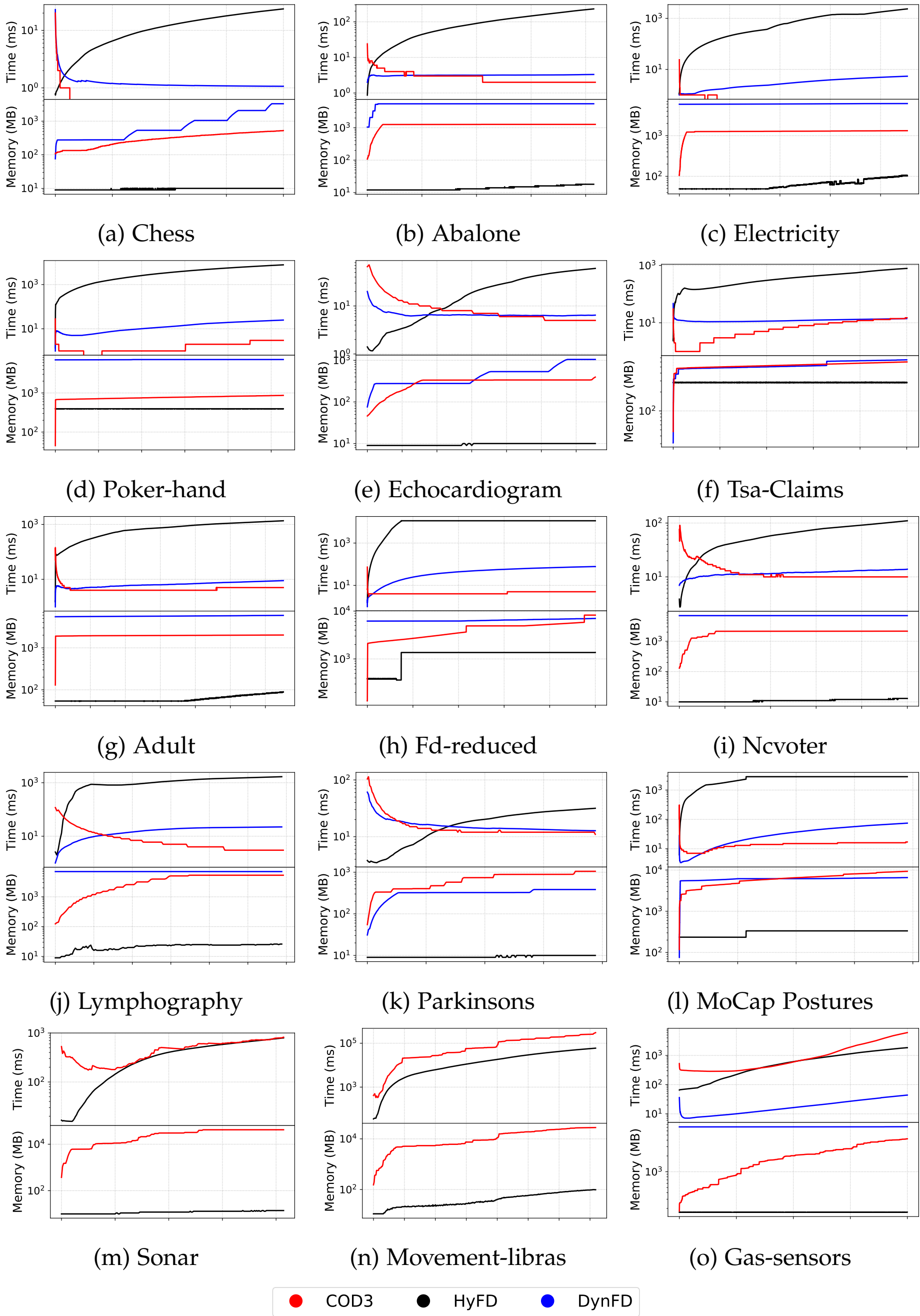


Figure 5.19: Time performances and memory load of COD3 by considering the variation of FDs at any time.

Concerning the comparison between COD3 and DynFD, the results show that COD3 always outperforms DynFD in terms of execution times, except for the *Gas-sensors* dataset. Similarly to HyFD, DynFD often achieves lower execution times during the first runs. This is mainly due to the fact that DynFD integrates the HyFD algorithm during initial steps to define all the underlying configurations, in order to successively work in a dynamic scenario.

It is worth noting that for the *Movement-libras* and *Sonar* datasets no results can be discussed for DynFD, since its executions exceeded the memory limit, which has been set to $50GB$. As expected, the incremental discovery algorithms (i.e., COD3 and DynFD) always require a greater amount of memory with respect to HyFD, due to the information of the previous executions that they manage. Nevertheless, COD3 almost always outperforms DynFD in terms of memory load, except for the *Parkinsons* and *MoCap Postures* datasets.

In general, time performances show that COD3 can process a huge number of rows with suitable execution times, without raising any memory issues. This makes COD3 particularly useful for the data stream context, where the number of rows can be extremely large. Moreover, the number of attributes considered in the data stream context is generally not large, unlike the number of rows.

5.5.4.3 *Discovery results of* COD3 *over data streams*

In this experiment we measured the effectiveness of COD3 on a real-world data stream. More specifically, this experiment exploits the data of over 200 real sensors spread throughout Italy, made available by the *AQICN* portal, which monitors and shares the air quality information during the day. In particular, we selected the following 13 attributes from the data stream:

- Particles $PM_{2.5}$ and PM_{10}: atmospheric aerosol particles, also known as "floating dust" or particulate matter (PM) with a diameter of 2.5 ($PM_{2.5}$), or 10 (PM_{10}) micrometers;
- Nitrogen dioxide (NO_2): the nitrogen dioxide concentration;
- Sulfur dioxide (SO_2): the sulfur dioxide concentration;

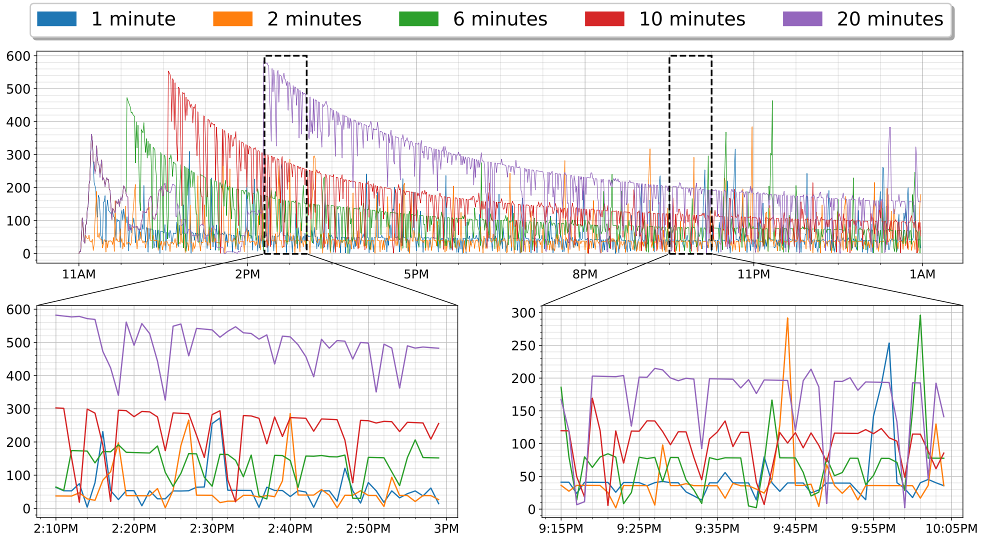


Figure 5.20: Number of FDs discovered by COD3 considering five time intervals.

- Dew point (dew): the temperature to which air must be cooled to become saturated with water vapour;
- Ozone (O_3): the ozone concentration;
- Temperature (t): the temperature in centigrade degrees;
- Humidity (h): the rate of humidity;
- Wind direction (w) and Wind speed (wg): the information on wind direction and speed, respectively;
- Pressure (p): the value of the atmospheric pressure;
- Rain (r): the amount of rain that fallen at the time of measurement;
- Carbon monoxide (CO): a colorless, odorless, and tasteless flammable gas that is slightly less dense than air.

We considered execution sessions of COD3 on the air quality data streams, lasting about 13 hours. In particular, we set up five parallel executions by considering five sliding windows, i.e., 1, 2, 6, 10, and 20 minutes, which determine the expiration times of the tuples.

The curves shown at the top of Figure 5.20 highlight the variability of holding FDs discovered with the different sliding windows, whereas the

zooms at the bottom provide more details in two different time windows. In particular, the line of each sliding window shows the standard deviation in the number of FDs with respect to the average number of holding FDs computed up to that time instant. As expected, the results show that the trend undertakes a bigger variability at the beginning of the discovery process, and tends to converge throughout the execution. This is particularly true for larger sliding windows. In fact, with sliding windows of 1 and 2 minutes the trends remain almost stable right after processing the initial tuples. This can be due to the fact that FD invalidations do not significantly impact on the number of holding FDs, since each analyzed tuple expires in a short time. Larger sliding windows obtain their peaks afterwards, due to the fact that the sliding windows start to move later, entailing a cold start of the expiration discovery process.

To extract different information from air quality sensor data, and to describe the existing relations among the analyzed parameters, we compared the set of FDs discovered during each execution. The results are summarized in Table 5.10, where for each sliding window we grouped the results into time periods representing the two different halves of the execution period, i.e., 13 hours, and the whole period. For each of them, we report the most frequent attribute on the left- and right-hand-side, and the average LHS cardinality, among all FDs holding in a specific time period. Top-5 FDs in such periods are also shown, representing the most validated FDs across the different time instants. From the results of Table 5.10 we can notice that Particles (PM_{10}) appears as the most frequent attribute on the LHS when the sliding window is set to 1 or 2 minutes. Instead, with the larger sliding windows the attribute Particles (PM_{10}) disappears, and the attribute Pressure (p) is the most frequent LHS. On the contrary, the attribute Rain (r) represents the most frequent RHS attribute (e.g., the most implied) with all sliding windows. As we expected, the cardinality of LHSs increases in average with larger sliding windows. This is mainly due to the fact that more tuples possibly induce more violations, yielding the inclusion of more attributes on the LHS of holding FDs.

Sliding Window	Time Period	Most common attribute		$\lvert\overline{LHS}\rvert$	Top 5 FDs
		LHS	RHS		
1 (min)	11 AM - 5 PM	Particles (PM_{10})	Rain (r)	3	(p, dew, h)→(r) (PM_{10}, p, h)→(r) (p, t, h)→(r) (p, h, w)→(r) (p, O_3, h)→(r)
	5 PM - 12 AM	Pressure (p)	Rain (r)	4	(p, t, h)→(r) (PM_{10}, t, h)→(r) ($PM_{2.5}$, p, h)→(r) (p, t, dew, SO_2)→(r) (PM_{10}, p, h)→(r)
	13 h	Particles (PM_{10})	Rain (r)	5	(p, t, h)→(r) (p, dew, h)→(r) (PM_{10}, p, h)→(r) (NO_2, p, t, w)→(r) (p, t, CO, w)→(r)
2 (min)	11 AM - 5 PM	Particles (PM_{10})	Rain (r)	3	(p, CO, h)→(r) ($PM_{2.5}$, p, h)→(r) (p, t, h)→(r) (p, SO_2, w)→(r) (p, O_3, h)→(r)
	5 PM - 12 AM	Pressure (p)	Rain (r)	6	(PM_{10}, p, dew, h, w)→(r) (p, O_3, dew, h, w)→(r) (p, dew, SO_2, h, w)→(r) (PM_{10}, $PM_{2.5}$, NO_2, p, O_3, h, w)→(SO_2) ($PM_{2.5}$, p, dew, h, w)→(r)
	13 h	Particles (PM_{10})	Rain (r)	5	(p, dew, h)→(r) ($PM_{2.5}$, p, h)→(r) (NO_2, p, t, dew)→(r) (p, t, h)→(r) (p, CO, w)→(r)
6 (min)	11 AM - 5 PM	Wind (w)	Rain (r)	5	($PM_{2.5}$, p, h, w)→(r) (PM_{10}, t, dew, SO_2, w)→(r) (p, dew, SO_2, h, w)→(r) (p, O_3, h, w)→(r) (PM_{10}, $PM_{2.5}$, p, O_3)→(SO_2)
	5 PM - 12 AM	Particles ($PM_{2.5}$)	Rain (r)	5	(PM_{10}, $PM_{2.5}$, NO_2, t, SO_2)→(CO) (PM_{10}, t, h)→(r) (p, h, t)→(r) ($PM_{2.5}$, dew, w)→(r) (dew, p, t)→(r)
	13 h	Pressure (p)	Rain (r)	6	(p, dew, h, w)→(r) (p, dew, SO_2, w)→(r) (p, O_3, dew, h)→(r) (NO_2, p, O_3, h, w, wg)→(dew) ($PM_{2.5}$, t, h)→(r)

Sliding Window	Time Period	Most common attribute		$\lvert\overline{\text{LHS}}\rvert$	Top 5 FDs
		LHS	RHS		
10 (min)	11 AM - 5 PM	Dew point (dew)	Rain (r)	5	(PM_{10}, p, t, dew)→(r) (SO_2, dew, PM_{10}, w)→(r) (p, SO_2, h)→(r) ($PM_{2.5}$, t, CO)→(h)
	5 PM - 12 AM	Particles ($PM_{2.5}$)	Rain (r)	4	($PM_{2.5}$, NO_2, w)→(CO) (PM_{10}, t, h)→(r) (p, SO_2, h, t)→(r) ($PM_{2.5}$, dew, w)→(r) ($PM_{2.5}$, dew, p, t)→(r)
	13 h	Pressure (p)	Rain (r)	6	(p, $PM_{2.5}$, t, w)→(r) (p, dew, w)→(r) (p, CO, dew, t)→(r) (NO_2, p, O_3, h, t)→(dew) ($PM_{2.5}$, t, h, dew)→(r)
20 (min)	11 AM - 5 PM	Pressure (p)	Rain (r)	3	(p, dew, h, NO_2)→(r) (p, dew, w, O_3)→(r) (PM_{10}, p, h)→(r) (p, SO_2, h, t)→(r) (p, NO_2, O_3, h, t)→(r)
	5 PM - 12 AM	Pressure (p)	Rain (r)	4	(p, dew, SO_2, h, NO_2)→(r) (p, NO_2 h)→(r) (PM_{10}, t, h, dew)→(r) ($PM_{2.5}$, p, h, NO_2)→(r) ($PM_{2.5}$, t, h, SO_2)→(r)
	13 h	Pressure (p)	Rain (r)	5	(p, CO, t, h)→(r) (p, dew, h, SO_2)→(r) (PM_{10}, p, NO_2, t)→(r) (NO_2, h, dew)→(r) ($PM_{2.5}$, t, h, SO_2, w)→(r)

Table 5.10: Summarized results obtained by COD3 across different execution sessions on real streams.

In what follows, we list some of the most frequently holding FDs with the sliding window of 1 minute, which are shared among the different time periods:

Pressure (p), Dew Point (dew), Humidity (h) → Rain (r),
Pressure (p), Temperature (t), Humidity (h) → Rain (r),
Pressure (p), Particles (PM_{10}), Humidity (h) → Rain (r).

These FDs highlight a strong relationship between humidity, pressure, and rainfall. In particular, the humidity and the pressure with another

attribute can imply the rainfall. In general, Rain (r) always appears as the RHS of the most frequent FDs across all considered time periods. Instead, with the other sliding windows, not only different periods seem to include different FDs among the most frequent ones, but the latter typically have a greater LHS and sometimes imply a RHS attribute different from Rain (r) (e.g., Sulfur dioxide (SO_2), Dew Point (dew), and Carbon monoxide (CO)).

5.6 BIRD: AN INCREMENTAL DISCOVERY ALGORITHM FOR RFD_eS RELAXING ON THE EXTENT

This section presents a new incremental discovery algorithm for RFD_es, named BIRD (Bit-vector based Incremental RFD_e Discoverer). In particular, we present its main steps by introducing the data structures, the search strategies, the pruning techniques, and the validation process. Moreover, we describe the procedures behind BIRD and provide experimental results on real-world datasets.

5.6.1 *Methodology*

BIRD is an incremental RFD_e discovery algorithm that performs a discovery process by traversing a lattice representation of the search space with a column-based strategy. It stores data using partitions [85], which represent a light and compact data structure that allows an efficient validation process of candidate RFD_es by means of the g3-error coverage measure [37]. Partitions were originally defined for a static discovery scenario, in which they were be defined during the pre-processing steps, before executing the discovery process. However, such data structure requires to be updated whenever at least one tuple is inserted, aiming to make it suitable for an incremental scenario.

Example 1. Starting from the example introduced in Figure 5.22, the partitions at time τ for the attribute periodical effusion of the *echocardiogram* dataset are $\pi_D(\tau) = \{[0,4,5,6,7,9],[1,2,3,8]\}$. However, the new partitions at time $\tau+1$ are $\pi_D(\tau+1) = \{[0,4,5,6,7,9,10,11,12,13,14,15,16,17,19],[1,2,3,8,18]\}$.

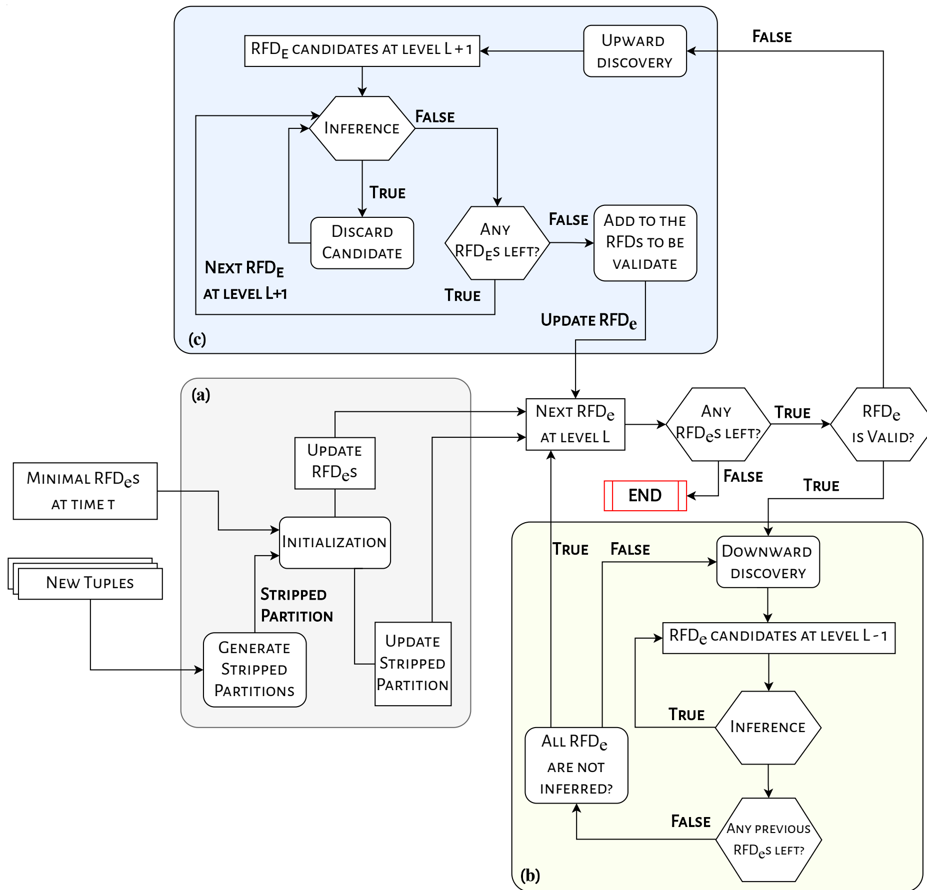


Figure 5.21: An overview of BIRD.

An overview of the BIRD's discovery process is provided in Figure 5.21. Given an instance r of a relation schema R at time τ, BIRD first performs a pre-processing step (Figure 5.21a), in which it pre-computes the partitions, updating all the partitions generated at time τ, according to the new inserted tuples. In order to avoid a long pre-processing phase, only partitions associated with the first lattice level are updated, i.e., those concerning individual attributes. The pre-processing phase also defines the starting point RFD$_e$s according to the properties defined in

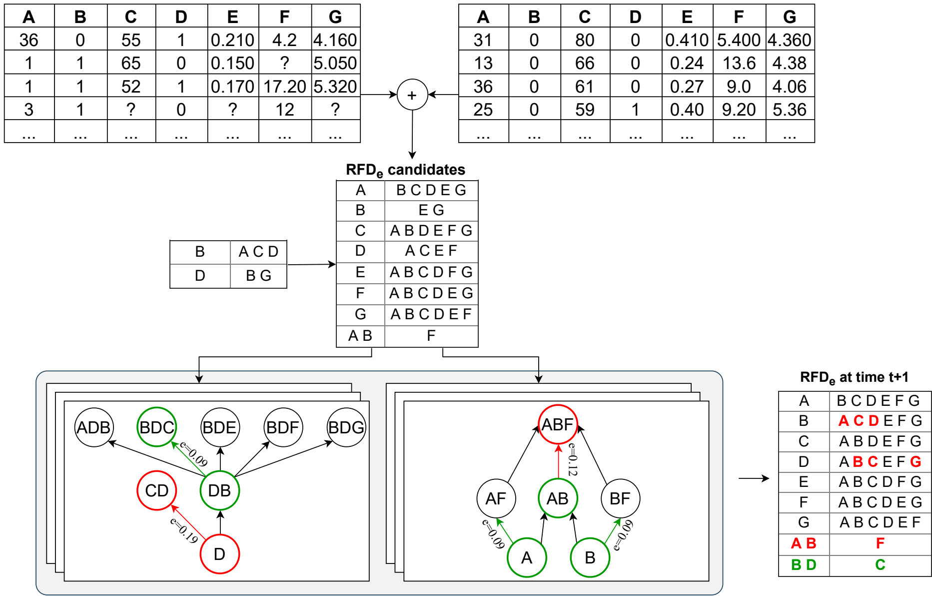


Figure 5.22: An overview of the proposed discovery process by considering $\varepsilon = 0.1$.

Section 5.1.3, and stores them in a data structure named *linked map*. Similarly to the linked map described in Section 5.3.1, this data structure allows performing an ordered discovery phase, on the basis of the LHS cardinalities of RFD_es, according to the APRIORI strategy [84]. Moreover, the linked map also stores all the candidate RFD_es generated during the discovery process.

Example 2. Let us consider the small snippet of the *echocardiogram* dataset shown in Table 5.11. Figure 5.22 shows a simulation on how new RFD_e candidates could be generated according to properties defined is Section 5.1.3, to the relation of Table 5.11, and to an error threshold $\varepsilon = 0.1$. In particular, in the middle of Figure 5.22 a compressed representation of the following minimal RFD_es holding at time τ is shown: $A \xrightarrow{\Psi \leq \varepsilon} BCDEG, B \xrightarrow{\Psi \leq \varepsilon} EG, C \xrightarrow{\Psi \leq \varepsilon} ABDEFG, D \xrightarrow{\Psi \leq \varepsilon} ACEF, E \xrightarrow{\Psi \leq \varepsilon} ABCDFG, F \xrightarrow{\Psi \leq \varepsilon} ABCDEG,\ G \xrightarrow{\Psi \leq \varepsilon} ABCDEF, AB \xrightarrow{\Psi \leq \varepsilon} F$. As said above, all these RFD_es become RFD_e candidates at time $\tau + 1$, and accord-

Survival (A)	Still alive (B)	Age at heart attack (C)	Pericardial effusion (D)	Fractional shortening (E)	Epss (F)	Lvdd (G)
36	0	55	1	0.210	4.2	4.160
1	1	65	0	0.150	?	5.050
1	1	52	1	0.170	17.200	5.320
3	1	?	0	?	12	?
27	0	47	0	0.400	5.120	3.100
35	0	63	0	?	10	?
26	0	61	0	0.610	13.100	4.070
16	0	63	1	?	?	5.310
1	1	65	0	0.060	23.600	?
19	0	68	0	0.510	?	3.880

Table 5.11: Snippet of the *echocardiogram* dataset.

ing to Property 2, it is necessary to add $B \xrightarrow{\Psi \leq \varepsilon} ACD$ and $D \xrightarrow{\Psi \leq \varepsilon} BG$ as new RFD$_e$ candidates. Moreover, Figure 5.22a shows what happens when an RFD$_e$ is invalidated at time $\tau + 1$. In particular, it shows that $D \rightarrow C$ is invalidated at time $\tau + 1$, and the new RFD$_e$ candidates $AD \xrightarrow{\Psi \leq \varepsilon} C$, $BD \xrightarrow{\Psi \leq \varepsilon} C$, $DE \xrightarrow{\Psi \leq \varepsilon} C$, $DF \xrightarrow{\Psi \leq \varepsilon} C$, $DG \xrightarrow{\Psi \leq \varepsilon} C$ are generated. Then, after the RFD$_e$ discovery algorithm performs the validation of such new RFD$_e$ candidates, only the RFD$_e$ candidate $BD \xrightarrow{\Psi \leq \varepsilon} C$ holds according to the input threshold [85]. Figure 5.22b shows what happens when an RFD$_e$ is valid at time $\tau + 1$, but it is no longer minimal. In particular, although $AB \xrightarrow{\Psi \leq \varepsilon} F$ is still valid after the insertion of new tuples, it is no longer minimal because $A \xrightarrow{\Psi \leq \varepsilon} F$ and $B \xrightarrow{\Psi \leq \varepsilon} F$ have been validated at time $\tau + 1$. Consequently, after the insertion of the new tuples (Figure 5.22), the RFD$_e$s valid at time $\tau + 1$ are the following ones: $A \xrightarrow{\Psi \leq \varepsilon} BCDEG$, $B \xrightarrow{\Psi \leq \varepsilon} ACDEG$, $C \xrightarrow{\Psi \leq \varepsilon} ABDEFG$, $D \xrightarrow{\Psi \leq \varepsilon} ABCEFG$, $E \xrightarrow{\Psi \leq \varepsilon} ABCDFG$, $F \xrightarrow{\Psi \leq \varepsilon} ABCDEG$, $G \xrightarrow{\Psi \leq \varepsilon} ABCDEF$.

When the process starts, the first candidate RFD$_e$ extracted from the linked map is validated by using the validation method implementing

the g3-error described in equation 2.1. If the RFD$_e$ is valid (Figure 5.21b), then BIRD performs *downward* discovery, since new added tuples could reduce the error of candidate RFD$_e$s that are minimal with respect to the considered one, but were not valid before. However, if an RFD$_e$ is not valid, an *upward* discovery is accomplished (see Figure 5.21c) in order to find the possible new holding RFD$_e$s. In both downward and upward search strategies, BIRD incrementally performs a minimality check on each RFD$_e$. This ensures that all the RFD$_e$s in the linked map that have already been analyzed are minimal. Such a minimality check strategy is much more efficient than checking the minimality after executing the discovery algorithm. At the end of each iteration, BIRD checks if there are other RFD$_e$s to analyze, and if not, it returns the minimal RFD$_e$s holding at time $\tau + 1$.

5.6.1.1 *Data Structures of* BIRD

BIRD uses fast and lightweight data structures, avoiding high memory usage. Similarly to the representation defined for FDs in the algorithm INCREMENTAL-FD (Section 5.3.1), RFD$_e$s have been represented by using a compressed structure based on binary vectors, i.e. $(0|1)^+$, and each of them has been associated a value e corresponding to its g3-error coverage measure. More specifically, let v_X be the vector representing the LHS of an RFD$_e$ φ, and v_A the vector representing its RHS, a binary representation of $\varphi : (v_X, v_A, e)$, has the following properties:

- If $v_X[i] = 1$ (or $v_A[j] = 1$) then the i-th attribute is included in the LHS (or the j-th one is included in the RHS);
- $v_X \wedge v_A = (0)^+$, e.g. only non-trivial RFD$_e$s are considered;
- Each binary RFD$_e$'s side must contain at least one bit equal to 1, i.e. $v_X = (0|1)^*1(0|1)^*$ and $v_A = (0|1)^*1(0|1)^*$;
- Each binary RFD$_e$ has associated the value e representing its associated $g3$-error;
- LHS and RHS vectors have a dynamic size adapted to the number of dataset's attributes.

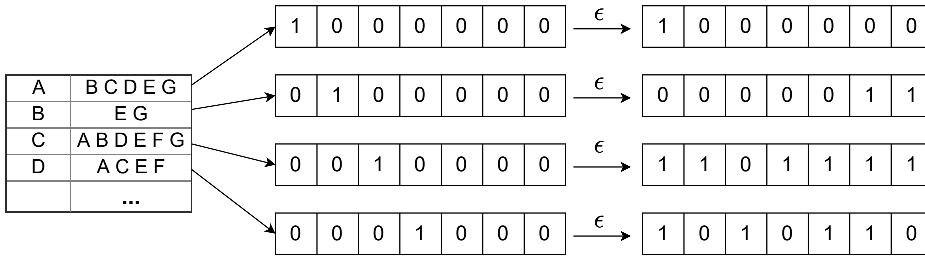


Figure 5.23: Binary representation of candidate RFD_es.

Figure 5.23 shows an example of the representation of the candidate RFD_es from the example shown in Figure 5.22.

Moreover, let Σ_τ be the set of RFD_es holding on a relation instance r, and assuming the RFD_e representation in terms of binary vectors described above, then the *compressed linked ordered map* has the following properties:

- Each RFD_e is inserted in the linked map according to an ordering criterion based on the number of attributes on X, i.e., by considering the number of 1s in the binary representation v_X of X.
- For each RFD_e there is a link to the next RFD_e, except for the last one;
- In order to guarantee the quick insertion of each RFD_e in the linked map, a support vector contains the references of the last inserted RFD_e for each v_X sharing the same number of 1s;
- For each pair of RFD_es, $\varphi : (v_X, v_A, e)$ and $\varphi' : (v'_X, v'_A, e') \in \Sigma$, then $v_X \neq v'_X$.

Figure 5.24 shows the *linked map* for the RFD_es described in the example shown in Figure 5.22. As we can see, all RFD_es have been mapped by using binary vectors with seven bits, which represent the number of attributes in R. The proper size of binary vectors is one of the most important properties that allows to reduce memory usage. The left-side of Figure 5.24 contains the support vector linking to the last inserted RFD_e among the RFD_es having the same number of attributes on the LHS. The right-side of the figure shows the value of each item in the map, representing a pointer to the next RFD_e. The discovery process starts from

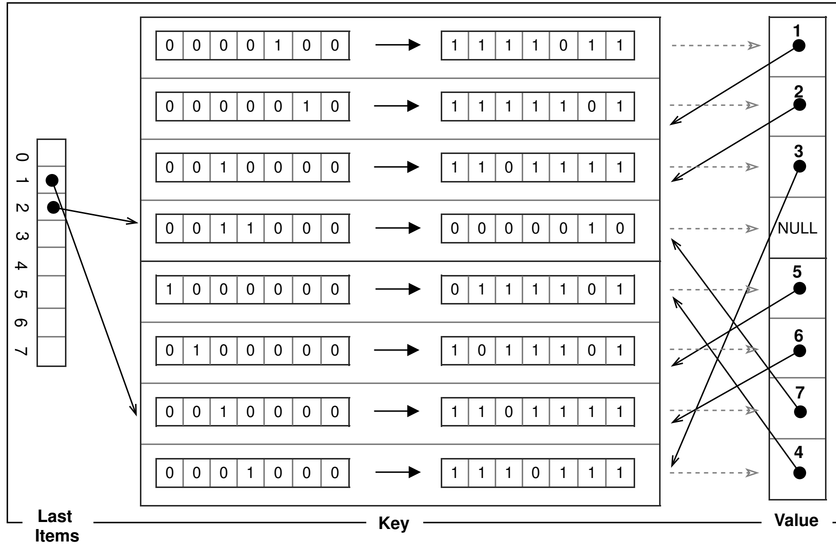


Figure 5.24: The compressed *linked map* related to the Example in Figure 5.22.

the first RFD_e, i.e. that with pointer *1*, and proceeds until it reaches the *null* pointer.

5.6.2 *The* BIRD *Algorithm*

The RFD_e candidate generation process is one of the main steps of the RFD_e discovery process. Given a candidate RFD_e $X \xrightarrow{\Psi \leq \varepsilon} A$, the generation candidate follows the properties defined in Section 5.1.3. Therefore, we consider two main cases in which the generation of new candidates is necessary, namely when the *invalidation of* RFD_es and/or the *refutation of minimality* occur. According to the first one, Algorithm 14 permits to generate candidate RFD_es by evaluating not valid RFD_es. In particular, it considers a candidate RFD_e not valid at time $\tau + 1$, together with the binary representation of its LHS, and it uses bit-wise operations to generate candidate RFD_es for the next lattice level. Algorithm 14 first checks whether it is possible to search candidate RFD_es at the next higher level (line 2), then it processes them an attribute at a time among those

Algorithm 14 NEXTCANDIDATES

INPUT: $X \xrightarrow{\Psi \leq \varepsilon} A \rightarrow$ An RFD$_e$ not valid at time $\tau + 1$, with v_X binary representation of the LHS
OUTPUT: $\Sigma_{\text{next}} \rightarrow$ The set of new candidate RFD$_e$s at time $\tau + 1$

1: $\Sigma_{\text{next}} \leftarrow \emptyset$
2: **if** $|X| < (|attr(R)| - 1)$ **then**
3:   **while** $B_i \in \text{attr}(R)$ **with** $i \leq |\text{attr}(R)|$ **do**
4:     $v_{X_{\text{next}}} \leftarrow v_X$
5:     **if** $B_i \neq A$ **then**
6:       **if** $v_{X_{\text{next}}}[i] \neq 1$ **then**
7:         $v_{X_{\text{next}}}[i] \leftarrow 1$
8:         $\Sigma_{\text{next}} \leftarrow \Sigma_{\text{next}} \cup \{X_{\text{next}} \xrightarrow{\Psi \leq \varepsilon} A\}$
9: **return** Σ_{next}

not already in their LHS or RHS. This avoids the inclusion of trivial RFD$_e$ in the linked map of candidates (lines 3-6). If these conditions are verified, then the algorithm generates a new candidate RFD$_e$, composed of the new LHS X_{next} and the same RHS A. In particular, this is accomplished through an bit-wise OR operation between v_X and $v_{X_{\text{next}}}$ (lines 7-8). Finally, it returns a new set of generated candidate RFD$_e$s (line 9).

Example 3. Figure 5.22a shows an example of this operation. The RFD$_e$ $D \xrightarrow{\Psi \leq \varepsilon} C$ is not valid at time $\tau + 1$. Using this strategy the set of candidates will contain the following RFD$_e$s: $AD \xrightarrow{\Psi \leq \varepsilon} C, BD \xrightarrow{\Psi \leq \varepsilon} C, ED \xrightarrow{\Psi \leq \varepsilon} C, FD \xrightarrow{\Psi \leq \varepsilon} C, GD \xrightarrow{\Psi \leq \varepsilon} C$, which are all non trivial ones.

Concerning the verification of the minimality property, as said above, Algorithm 15 computes possible candidate RFD$_e$s on the previous lattice level. Similarly to the Algorithm 14, given an RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$, together with the binary representation of its LHS v_X, it first checks whether it is possible to search candidate RFD$_e$s at the previous lower level (line 2), and if so, it considers each $B_i \in X$ at a time, removing it from the LHS (lines 5-6). Next, the algorithm adds the new candidate RFD$_e$s to the set of candidate RFD$_e$s (line 7).

Algorithm 15 PREVIOUSCANDIDATES

INPUT: $X \xrightarrow{\Psi \leq \varepsilon} A \rightarrow$ An RFD$_e$ not valid at time $\tau + 1$, with v_X binary representation of the LHS
OUTPUT: $\Sigma_{\text{prev}} \rightarrow$ The set of new candidate RFD$_e$s at time $\tau + 1$

1: $\Sigma_{\text{prev}} \leftarrow \emptyset$
2: **if** $|X| > 1$ **then**
3: **while** $B_i \in \text{attr}(R)$ **with** $i \leq |attr(R)|$ **do**
4: $v_{X_{\text{prev}}} \leftarrow v_X$
5: **if** $v_{X_{\text{prev}}}[i] = 1$ **then**
6: $v_{X_{\text{prev}}}[i] \leftarrow 0$
7: $\Sigma_{\text{prev}} \leftarrow \Sigma_{\text{prev}} \cup \{X_{\text{prev}} \xrightarrow{\Psi \leq \varepsilon} A\}$
8: **return** Σ_{prev}

Example 4. Figure 5.22b shows an example of this operation. The RFD$_e$ $AB \xrightarrow{\Psi \leq \varepsilon} F$ is still valid a time $\tau + 1$. However, following the proposed discovery strategy, we check if there are RFD$_e$s in the previous lattice level with associated g3-error $e \leq \varepsilon$. Thus, the following new candidate RFD$_e$s need to be considered: $A \xrightarrow{\Psi \leq \varepsilon} F, B \xrightarrow{\Psi \leq \varepsilon} F$.

5.6.2.1 *Inference*

During the discovery process, as new candidate RFD$_e$s are validated, it is necessary to check whether they are minimal. Thus, in the proposed search process the minimality check (that we call *Inference*) is accomplished by exploiting a new compressed linked map.

More formally, given a relation instance r of a relation schema R and $X, Z \subseteq attr(R)$, such that $Z \subset X$, then for each RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$ validated at time $\tau + 1$ it is necessary to verify that there exists no RFD$_e$ $Z \xrightarrow{\Psi \leq \varepsilon} A$, among those already validated at time $\tau + 1$.

The inference process requires complex analysis steps to be performed on validated RFD$_e$s. For this reason, we use a linked map to group RFD$_e$s

Algorithm 16 IS_INFERRED

INPUT:
$\varphi : X \xrightarrow{\Psi \leq \varepsilon} A \rightarrow$ An RFD$_e$ holding at time $\tau + 1$
$\Sigma_\tau \rightarrow$ A set of candidate RFD$_e$s
OUTPUT:
True $\rightarrow$ If the φ is not minimal
False $\rightarrow$ Otherwise

```
1: if Σ_τ is empty then
2:     return false
3: else
4:     for each Z --Ψ≤ε--> A ∈ Σ_τ do
5:         if |Z| ≤ |X| then
6:             v_(X∧Z) ← v_X ∧ v_Z
7:             if v_(X∧Z) = v_Z then
8:                 return false
9: return true
```

sharing the same RHS and already validated at time $\tau + 1$. This allows us to reduce the number of RFD$_e$s to be checked.

Algorithm 16 shows the proposed inference process. It considers an RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$ holding at time $\tau + 1$, and a linked map containing all already validated minimal RFD$_e$s and grouped by each possible RHS, and checks if it is minimal. Algorithm 16 first checks if there exists at least one RFD$_e$ holding at time $\tau + 1$ (lines 1-3), and if so, then for each candidate RFD$_e$ $Z \xrightarrow{\Psi \leq \varepsilon} A$ checks if the cardinality of Z is lower than the one of X (lines 4-5), and it performs a bit-wise AND operation between the binary vector representations of X and Z (lines 6-7). If the result is equal to the binary vector of Z, then this means that $Z \subset X$, i.e., $X \xrightarrow{\Psi \leq \varepsilon} A$ is not minimal (line 8).

5.6.2.2 *Main procedure of* BIRD *algorithm*

Algorithm 17 presents the proposed BIRD discovery algorithm. Given Σ_τ the set of all RFD$_e$s holding at time τ and of all RFD$_e$s generated as starting

Algorithm 17 BIRD Algorithm

INPUT: $\Sigma_\tau \to$ A set of candidate RFD$_e$s as starting points from time τ

OUTPUT: $\Sigma_{\tau+1} \to$ A set of new valid and minimal FDs at time $\tau+1$

```
1: for each X --Ψ≤ε--> A ∈ Σ_τ in ascending ordering by LHSs do
2:   if VALIDATION(X --Ψ≤ε--> A) is not valid then
3:     L_{l+1} ← NEXTCANDIDATES(X --Ψ≤ε--> A)
4:     for each W --Ψ≤ε--> A ∈ L_{l+1} do
5:       if IS_MINIMAL(W --Ψ≤ε--> A) then
6:         Σ_τ ← Σ_τ ∪ {W --Ψ≤ε--> A}
7:   else
8:     L_{l-1} ← PREVIOUSCANDIDATES(X --Ψ≤ε--> A)
9:     for each Z --Ψ≤ε--> A ∈ L_{l-1} do
10:      if VALIDATION(Z --Ψ≤ε--> A) is valid then
11:        L_{l-1} ← L_{l-1} ∪ PREVIOUSCANDIDATES(Z --Ψ≤ε--> A)
12:      else
13:        L_{l-1} ← L_{l-1} \ {Z --Ψ≤ε--> A}
14:    if |L_{l-1}| > 0 then
15:      Σ_τ ← Σ_τ \ {X --Ψ≤ε--> A}
16:      for each Z --Ψ≤ε--> A ∈ L_{l-1} do
17:        if IS_MINIMAL(Z --Ψ≤ε--> A) then
18:          Σ_τ ← Σ_τ ∪ Z --Ψ≤ε--> A
19: Σ_{τ+1} ← Σ_τ
```

points according to the Properties 1, 2 and 3 (see Section 5.1.3), BIRD starts by analyzing the candidate RFD$_e$s with lower cardinality (line 1). In particular, we use the compressed linked map to facilitate a level-wise discovery, so as to avoid the sorting of all RFD$_e$s. Thus, for each RFD$_e$ holding at time τ, BIRD uses the VALIDATION process (line 2) to verify whether $X \xrightarrow{\Psi \leq \varepsilon} A$ still holds at time $\tau+1$. This process is performed according to the g3-error computation described in Equation (2.1). Thus,

if the analyzed RFD$_e$ is not valid, BIRD generates new candidate RFD$_e$s at the next lattice level (Algorithm 14), by discarding those that can be inferred from other RFD$_e$s, according to Algorithm 16 (lines 3-6). Notice that the ordering criterion of the RFD$_e$s permits to avoid the re-validation of some candidate RFD$_e$s.

Example 5. As mentioned in the example shown in Figure 5.22, if we consider the RFD$_e$ $D \xrightarrow{\Psi \leq \varepsilon} C$, it is not valid at time $\tau + 1$. Thus, the RFD$_e$s $AD \xrightarrow{\Psi \leq \varepsilon} C, BD \xrightarrow{\Psi \leq \varepsilon} C, DE \xrightarrow{\Psi \leq \varepsilon} C, FD \xrightarrow{\Psi \leq \varepsilon} C, GD \xrightarrow{\Psi \leq \varepsilon} C$ become new candidate RFD$_e$s at time $\tau + 1$. Moreover, let suppose that there exists another RFD$_e$ $E \xrightarrow{\Psi \leq \varepsilon} C$ that is not valid a time $\tau + 1$, then we should also consider all the RFD$_e$s $AE \xrightarrow{\Psi \leq \varepsilon} C, BE \xrightarrow{\Psi \leq \varepsilon} C, DE \xrightarrow{\Psi \leq \varepsilon} C, EF \xrightarrow{\Psi \leq \varepsilon} C, EG \xrightarrow{\Psi \leq \varepsilon} C$ as new candidates. However, the RFD$_e$ $DE \xrightarrow{\Psi \leq \varepsilon} C$ appears twice, but it will be validated only once, thanks to the defined compressed linked map. However, if $X \xrightarrow{\Psi \leq \varepsilon} A$ is a valid RFD$_e$, the algorithm checks

if other RFD$_e$s in the previous lattice level have been already validated at time $\tau + 1$ (lines 8-18). For each RFD$_e$ φ holding at time $\tau + 1$ the process tries to validate its neighbours on the previous lattice level, and φ is not removed from $\Sigma_{\tau+1}$ if and only if none of its neighbours are valid. Moreover, in this last case the iterations on previous candidate RFD$_e$s are stopped due to the analyzed RFD$_e$. Notice that the minimality of candidate RFD$_e$s on previous lattice levels is always locally checked (lines 16-18).

It is worth to notice that, while the implementation of BIRD considers several code optimizations and takes advantage of the defined data structures, for sake of clarity, the pseudo-codes described above do not show such optimizations. They mainly guarantee that each possible candidate RFD$_e$ is validated at most once. Thus, in the worst case, the exploration of the search space is equivalent to the one of non-incremental RFD$_e$ discovery algorithms.

5.6.2.3 *Parallelism in* BIRD

As mentioned above, the RFD$_e$ discovery problem is a complex one, especially with datasets having high cardinality and dimensionality. To

Algorithm 18 BIRD Parallel Version

INPUT: $\Sigma_\tau \rightarrow$ A set of candidate RFD$_e$s as starting points from time τ

OUTPUT: $\Sigma_{\tau+1} \rightarrow$ A set of new valid and minimal FDs at time $\tau + 1$

```
1:  n_CPU ← number of available CPUs
2:  Σ_{τ+1} ← ∅
3:  l ← 0
4:  while l ≤ |attr(R)| do
5:      Γ ← CREATEPOOL(n_CPU)
6:      L_l ← GETRFDS(l)
7:      for all γ ∈ Γ do
8:          φ ← L_l.NEXTDEPENDENCY()
9:          γ.execute(WORKERDISCOVERY(φ,Σ_τ, Σ_{τ+1}))
10:     WAITWORKERS()
11:     l ← l + 1
12: return Σ_{τ+1}
```

tackle this problem, we defined a parallel version of BIRD (named BIRD^p), which differs from the sequential version in that the generation and the validation of candidate RFD$_e$s are accomplished level-wise, i.e. one level at a time, by also using thread-safe collections. Thus, the linked map has been defined as a thread-safe data structure. Moreover, BIRD^p splits the space of candidate RFD$_e$s, assigning them to several workers. More specifically, a pool of thread-workers is allocated for each level in order to continuously monitor its status, enabling the execution of each work only when necessary. In particular, let l be the selected lattice level, then L_l is the set of all candidate RFD$_e$s at time $\tau + 1$ having LHS cardinality equal to l. More specifically, the workload, i.e. the number of candidate RFD$_e$s to be validated, is distributed among all workers, assigning a validation task to each worker whenever it becomes available. The results of each worker are merged into a thread-safe set, which is updated by each thread after performing the assigned task.

Algorithms 18 and 19 implement BIRD^p. In particular, Algorithm 18 shows the initialization phase of the thread pool. Let l be the LHS

Algorithm 19 BIRD Worker

```
1:  function WORKERDISCOVERY(X --Ψ≤ε--> A, Σ_τ, Σ_{τ+1})
2:  | if VALIDATION(X --Ψ≤ε--> A) is not valid then
3:  | | L_{l+1} ← NEXTCANDIDATES(X --Ψ≤ε--> A)
4:  | | for each W --Ψ≤ε--> A ∈ L_{l+1} do
5:  | | | if IS_MINIMAL(W --Ψ≤ε--> A then)
6:  | | | | Σ_τ ← Σ_τ ∪ {W --Ψ≤ε--> A}
7:  | else
8:  | | L_{l−1} ← PREVIOUSCANDIDATES(X --Ψ≤ε--> A)
9:  | | for each Z --Ψ≤ε--> A ∈ L_{l−1} do
10: | | | if VALIDATION(Z --Ψ≤ε--> A) is valid then
11: | | | | L_{l−1} ← L_{l−1} ∪ PREVIOUSCANDIDATES(Z --Ψ≤ε--> A)
12: | | | else
13: | | | | L_{l−1} ← L_{l−1} \ {Z --Ψ≤ε--> A}
14: | | if |L_{l−1}| > 0 then
15: | | | Σ_τ ← Σ_τ \ {X --Ψ≤ε--> A}
16: | | | for each Z --Ψ≤ε--> A ∈ L_{l−1} do
17: | | | | if IS_MINIMAL(Z --Ψ≤ε--> A) then
18: | | | | | Σ_τ ← Σ_τ ∪ Z --Ψ≤ε--> A
19: | Σ_{τ+1} ← Σ_{τ+1} ∪ Σ_τ
```

cardinality of the RFD$_e$s to be analyzed at level l, then the algorithm selects only the RFD$_e$s to be checked by using a custom filter (line 6). The latter is a stream filter based on lambda expressions that selects the RFD$_e$s from the map having a specific LHS cardinality defined by l (line 3). Next, for each candidate RFD$_e$, it assigns an asynchronous task to a worker so that the discovery phase can start through the procedure WORKERDISCOVERY (lines 7-9). Moreover, for each iteration, it waits for the termination of each thread before moving to the next level.

Finally, each worker stores the RFD$_e$s discovered at time $\tau + 1$ in a shared set $\Sigma_{\tau+1}$ that is returned after the complete execution of BIRD^p.

Instead, Algorithm 19 receives a candidate RFD_e as input and tries to validate it by also checking the minimality property. Notice that, this procedure follows in part the same discovery and validation process of the sequential version (Algorithm 17) described above.

5.6.3 *Theoretical Evaluation*

From a theoretical point of view, it is necessary to guarantee that BIRD is able to find all and only the minimal RFD_es holding on a given relation instance r. Thus, it is necessary to prove the correctness of discovered RFD_es. To this end, the correctness of RFD_es discovered with BIRD can be assessed through well-known methods and properties proposed in the literature. In particular, BIRD implements the one proposed in [85]. Notice that, all the proofs provided below refer to the basic version of BIRD (Algorithm 17), even though they can be easily generalized to the parallel one.

Minimality. One of the evaluation dimensions of a RFD_e discovery algorithm is minimality, which guarantees that the discovered RFD_es no longer hold upon removing an LHS attribute.

Theorem 1. Each RFD_e discovered by BIRD is minimal according to the minimality property defined in Section 5.1.3.

Proof. BIRD starts with the set of all candidate RFD_es Σ_τ, and updates it according to validation results in ascending order by LHS cardinality (Line 1). Thus, if the first candidate RFD_e $\varphi_0 \in \Sigma_\tau$ is valid, then it is also minimal, since no `PREVIOUSCANDIDATES` exist. Now, assuming that all minimal RFD_es with LHS cardinality k have been added to Σ_τ, then we prove by induction that if a generic RFD_e $\varphi_{k+1} \in \Sigma_\tau$, having LHS cardinality $k+1$, is valid, then it is also minimal.

In Algorithm 17, when φ_{k+1} is validated, then BIRD executes Lines $8-18$. Thus, `PREVIOUSCANDIDATES` are collected and explored (Lines 8-9), and if there exists at least one valid RFD_e in the previous lattice level, then it is necessary to explore the previous levels in order to assess the minimality property more in depth. However, if there are no valid

RFD$_e$s in a previous level, then BIRD stops the exploration process, since according to the *refinement property* [85], no candidate from the previous levels can be valid. During the exploration of previous levels, if there exists at least one RFD$_e$ that is minimal with respect to φ_{k+1}, then the latter is removed (Line 15). As a consequence, all valid RFD$_e$s discovered by `PREVIOUSCANDIDATES` in previous levels will be added to Σ after assessing their minimality (Line 17-18). Thus, if a valid RFD$_e$ φ_{k+1} with cardinality $k+1$ is maintained into Σ_τ after the exploration by `PREVIOUSCANDIDATES` on those with cardinality k, then it is also minimal. □

Completeness. Another important evaluation dimension of a RFD$_e$ discovery algorithm is completeness, which guarantees that the algorithm discovers *all* minimal RFD$_e$s.

Theorem 2. BIRD discovers all minimal RFD$_e$s.

Proof. At each time instant, BIRD proceeds incrementally by considering an initial set Σ_τ of candidate RFD$_e$s. Let us first consider the candidate RFD$_e$s appearing in Σ_τ at the beginning of the discovery process, at a given time $\tau+1$, and then in order:

- all the RFD$_e$s holding at the previous time instant (τ);
- all the candidate RFD$_e$s $B \xrightarrow{\Psi \leq \varepsilon} A$ such that there is no minimal RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$ holding at time τ, for any attribute $B \neq A$ (Property 1);
- all the candidate RFD$_e$s $B \xrightarrow{\Psi \leq \varepsilon} A$ such that there is no minimal RFD$_e$ $X \xrightarrow{\Psi \leq \varepsilon} A$ holding at time τ, with $B \in X$ (Property 2); and
- all the candidate RFD$_e$s $X \xrightarrow{\Psi \leq \varepsilon} A$ such that for each $B \in X$, B belongs to the LHS of some RFD$_e$s holding at time τ, but at time τ there exists neither a minimal RFD$_e$ $X' \xrightarrow{\Psi \leq \varepsilon} A$, with $X' \subseteq X$ and $B \in X'$, nor a minimal RFD$_e$ $X'' \xrightarrow{\Psi \leq \varepsilon} A$, such that $X \subseteq X''$ and $B \in X''$ (Property 3).

Notice that Property 1 guarantees the fact that at the first time instant ($\tau = 0$), for each attribute A in $attr(R)$, any candidate $B \xrightarrow{\Psi \leq \varepsilon} A$, with $B \neq A$, is added to Σ_τ.

Let us now prove by contradiction the completeness of the proposed search strategy. Let us assume that BIRD misses a minimal RFD$_e$ φ :

$X \xrightarrow{\Psi \leq \varepsilon} A$ holding on an input instance r. Then, 1) X cannot be superset of any candidate RFD$_e$s $X' \xrightarrow{\Psi \leq \varepsilon} A$, with $X' \subset X$, since if the latter has already been validated (Algorithm 17 processes candidates in increasing order of their LHS cardinality), φ would not be minimal; instead, if $X' \xrightarrow{\Psi \leq \varepsilon} A$ does not hold on r, then φ is analyzed by BIRD (Line 3), unless some $X'' \xrightarrow{\Psi \leq \varepsilon} A$, with $X' \subset X'' \subset X$, has already been validated; 2) X cannot be subset of any RFD$_e$ $X' \xrightarrow{\Psi \leq \varepsilon} A$ validated during the discovery process (Line 8-18), according to the aforesaid minimality proof.

Consequently, since all RFD$_e$s holding at time τ represent candidate RFD$_e$s holding at time $\tau + 1$, then points 1) and 2) prove that for each $X' \xrightarrow{\Psi \leq \varepsilon} A$ holding at time τ, $X \neq X \cup X' \neq X'$, that is, neither $X \subseteq X'$ nor $X' \subseteq X$. However, according to Property 3, for each $B \in X$, if there exists at least an RFD$_e$ $X' \xrightarrow{\Psi \leq \varepsilon} A$, such that $B \in X'$, then all possible candidate RFD$_e$s at the lowest possible level are added to the set of candidate RFD$_e$s. This means that for all $B \in X$ no RFD$_e$ valid at time τ can contain B in its LHS. Thus, according to Property 2, B is not included in S_τ and $B \xrightarrow{\Psi \leq \varepsilon} A$ would be added as a candidate RFD$_e$ at time $\tau + 1$. We can deduce that no minimal RFD$_e$ $X' \xrightarrow{\Psi \leq \varepsilon} A$ holds at time τ for any possible attribute set $X' \subset attr(R)$, i.e. attribute A is never determined at time τ. Moreover, according to Property 1, A would never be included into Z_τ, and then for each $B \neq A$ an RFD$_e$ $B \xrightarrow{\Psi \leq \varepsilon} A$ is included as candidate at time $\tau + 1$. Consequently, $X \xrightarrow{\Psi \leq \varepsilon} A$ is always considered as candidate during the discovery process, unless some $X' \xrightarrow{\Psi \leq \varepsilon} A$, with $X' \subseteq X$, has already been validated. In this case, $X \xrightarrow{\Psi \leq \varepsilon} A$ cannot be considered as minimal, which contradicts the original assumption. □

5.6.4 *Experimental Evaluation*

In what follows, we present the experimental evaluation of BIRD on several public datasets, comparing results with those of the TANE algo-

Statistics						
ID	**Dataset**	**Columns [#]**	**Rows [#]**	**Size [KB]**	**fds [#]**	**rfd$_e$s [#]**
1	Iris	5	150	5	4	13
2	Balance-scale	5	625	7	1	5
3	Chess	7	1999	519	1	7026
4	Abalone	9	4176	187	137	88
5	Nursery	9	12960	1024	1	4457
6	Breast-cancer-wisconsin	11	699	20	46	95
7	Bridges	13	108	6	142	340
8	Echocardiogram	13	132	6	538	172
9	Tsa-claims	13	145143	25608	28	355
10	Adult	14	32561	3528	78	2289
11	Ncvoter	19	1001	151	758	3191
12	Hepatitis	20	155	8	8250	14973

Table 5.12: Statistics of the considered public datasets [12].

rithm[8]. In particular, for BIRD we split a given dataset into two portions, where the first one is used to obtain the RFD$_e$s holding at time τ, and the second portion to run BIRD. This permits to simulate the insertion of new tuples in an incremental scenario. Instead, for TANE, we ran its discovery process on the entire dataset. The comparative analysis aims to show the advantages of an incremental discovery algorithm with respect to a complete re-execution from scratch. Among the different RFD$_e$ discovery algorithm proposed in literature, we chose TANE for the comparative evaluation because BIRD relies on its validation strategy.

Implementation details. BIRD has been developed in Java 11. Moreover, to avoid the re-computation of partitions, which are widely used for the validation of candidate RFD$_e$s, we introduced a strategy to store them in cache memory. Finally, as said above, we implemented two versions of BIRD, the sequential and parallel versions (named BIRD^p). The latter exploits functional programming techniques to properly manage parallel executions.

8 We used the implementation available at: https://github.com/HPI-Information-Systems/metanome-algorithms

Hardware and Datasets. All experiments have been executed on a Mac with an Intel Xeon W 3.2GHz 32-core CPU with 128GB of RAM, running MacOS Mojave and OpenJDK 64-Bit 12.0.2 as Java environment. We evaluated BIRD on twelve public datasets, whose details on the considered datasets are shown in Table 5.12.

Evaluation process. For evaluating BIRD we simulated the tuple insertion by splitting each original dataset r in two portions, based on a given percentage value. The first portion, indicated as r_τ, represents the instance at time τ, whereas the second one, indicated as $r_{\tau+1}$, refers to the instance resulting from the insertion of tuples from time τ to time $\tau + 1$. In particular, BIRD uses the RFD_es extracted by TANE on r_τ as starting points. Moreover, we analyzed the time performances of BIRD on $r_{\tau+1}$ by varying *i*) the percentage of tuples inserted from time τ to time $\tau + 1$, denoted as $P_{\tau+1}$, *ii*) the g3-error threshold, and *iii*) the number of execution threads.

5.6.4.1 *Comparative evaluation*

We performed a comparative evaluation of BIRD versus the TANE algorithm. In particular, BIRD and BIRD^p have been executed with $P_{\tau+1} = 10\%$, and all three algorithms used a g3-error threshold $\varepsilon = 0.3$. More specifically, we analyze the average times for BIRD and BIRD^p, since for each dataset we performed 10 executions by considering different cut points.

Figure 5.25 reports the execution times achieved by the different algorithms. They highlight that BIRD and BIRD^p outperform TANE on all datasets with one or more order of magnitude, despite the variability of the dataset in terms of the number of rows/columns. In particular, the execution times of BIRD are always almost an order of magnitude lower than those of TANE. The lowest performance gap is obtained with the datasets *Chess*, *Tsa-claims*, *Ncvoter*, and *Hepatitis*, for which BIRD still gets better performances. They represent the most complex datasets due to the high number of holding RFD_es, which require BIRD to manage a high number of starting points. Moreover, as expected, BIRD^p improves the execution times of BIRD.

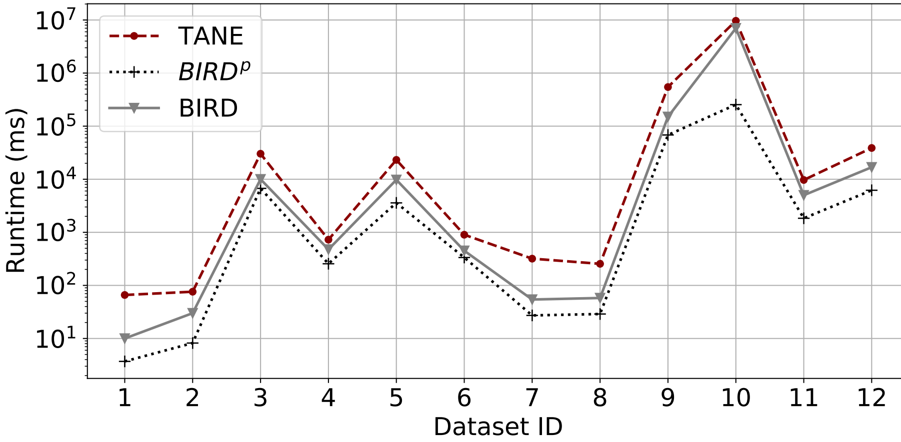


Figure 5.25: Time performances of TANE, BIRD, and BIRD^p.

Another experimental session aimed to analyze the time efficiency of BIRD^p, by using a configuration with $P_{\tau+1} = 60\%$, and by varying the g3-error thresholds in the range $[0.1, 1]$ with step 0.1. The average execution times of BIRD^p are shown based on 10 executions, in which we used random cut-points to split the data into two portions according to $P_{\tau+1}$. Moreover, results are always compared with a complete execution from scratch with TANE (red bars). Figure 5.26 highlights the execution times achieved by BIRD^p (blue bars). We can notice that the execution times of BIRD^p have a decreasing trend when the $g3$-error threshold increases. This is due to the fact that the number of holding RFD$_e$s decreases as the $g3$-error threshold increases, and they typically have small LHS cardinalities, which enable the application of pruning strategies to discard lattice paths yielding non-minimal RFD$_e$s. Non-monotonic and quite constant trends are followed by the execution times on *Abalone*, *Tsa-claims*, and the *Iris* datasets. From the analysis of the discovered RFD$_e$s we observed that on these datasets BIRD^p validates almost always the set of RFD$_e$s received as starting points. Results of *Breast-cancer-wisconsin* represent the only particular case, in which we can notice a high variability on execution

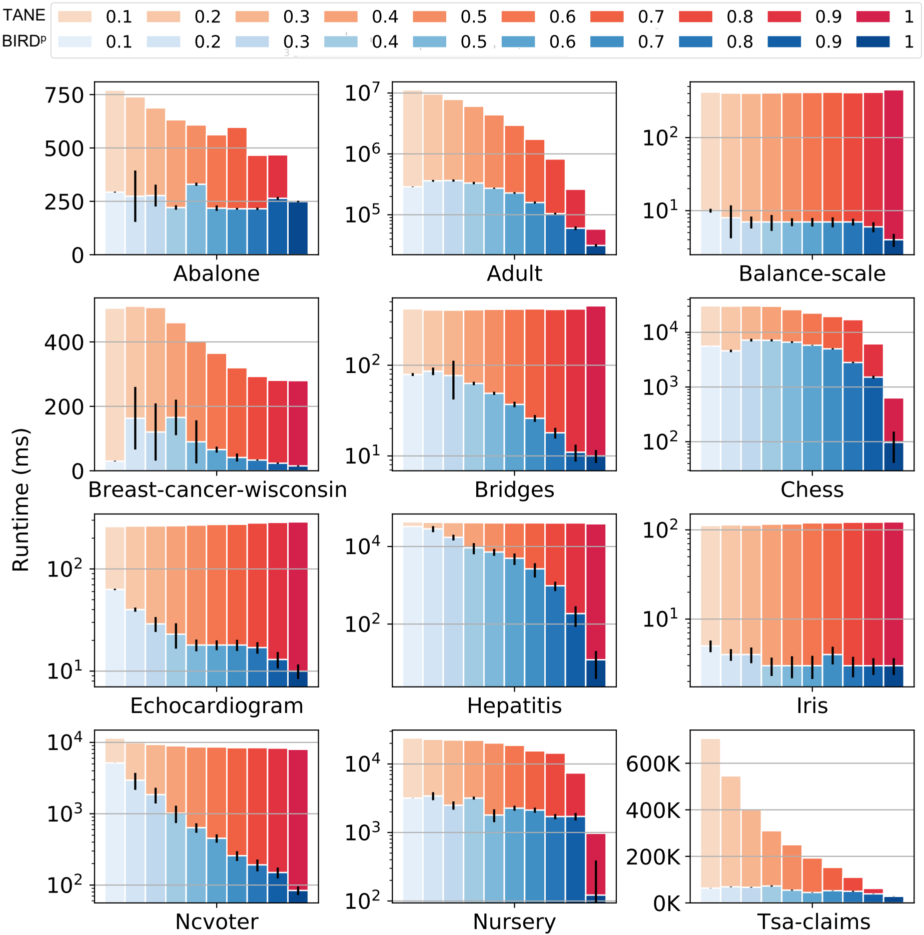


Figure 5.26: Time performances with $P_{\tau+1} = 60\%$ and $g3$-error threshold in the range $[0.1, 1]$.

times when considering smaller thresholds, whereas the trend became decreasing starting from $\varepsilon \geq 0.4$.

Figure 5.26 also shows the error bars computed on the different executions performed by considering random cut-points on data. It can be notice that, in most cases, the variation in times is tiny. The error bars show a slightly bigger variability when average execution times present a

significant gap with respect to those obtained with the previous smaller threshold. Only few exceptions are highlighted into results of *Abalone* and *Bridges*.

With respect to the comparison of results, it is clear that BIRD^p outperforms TANE. As opposed to BIRD^p, the latter often presents quite constant and extremely high execution times. In general, the gap among the performances between the two algorithms is big, mainly with higher thresholds. However, the only cases in which similar time performances are observed for the two algorithms occur with $\varepsilon = 1$ in *Abalone*, and $\varepsilon \leq 0.2$ in *Hepatitis*.

We also evaluated BIRD^p performances with a fixed $g3$-error threshold, and by varying the $P_{\tau+1}$ values. We varied $P_{\tau+1}$ in the range $[10\%, 90\%]$ with step 10%, and set the $g3$-error threshold to $\varepsilon = 0.2$. Also in this case, we show the average execution times of BIRD^p resulting from 10 executions, in which we used random cut-points to split the data in two portions according to $P_{\tau+1}$. Moreover, results are always compared to those of a complete execution from scratch with TANE.

Figure 5.27 summarizes the obtained results. As opposed to the results achieved by varying the $g3$-error threshold, in this case it is not possible to infer a general trend. In fact, we can notice a non-monotonic trend on the *Abalone*, *Balance-scale*, *Breast-cancer-wisconsin*, *Chess*, and *Iris* datasets, a quite constant trend on *Echocardiogram*, *Nursery*, and *Tsa-claim*, and an increasing trend on *Adult*, *Bridges*, *Hepatitis*, and *Ncvoter*. The latter trend is the one we expected, since when $P_{\tau+1}$ increases, the number of new tuples added from time τ to time $\tau + 1$ is bigger. Thus, BIRD^p could invalidate a high number of RFD$_e$s. Usually, this occurs on datasets with high cardinality, since possible introduced violations could lead to a big variation on holding RFD$_e$s. In general, the achieved results highlight that when $P_{\tau+1}$ increases, the execution times depend on the dataset.

According to the error bars, we can notice that in general there is a small variability in execution times. Exceptions can be found for *Balance-scale*, *Breast-cancer-wisconsin*, and *Iris*. The latter are the datasets with a smaller number of holding RFD$_e$s, and also present a non-monotonic trend. This could be due to the fact that the different cut-points can induce the

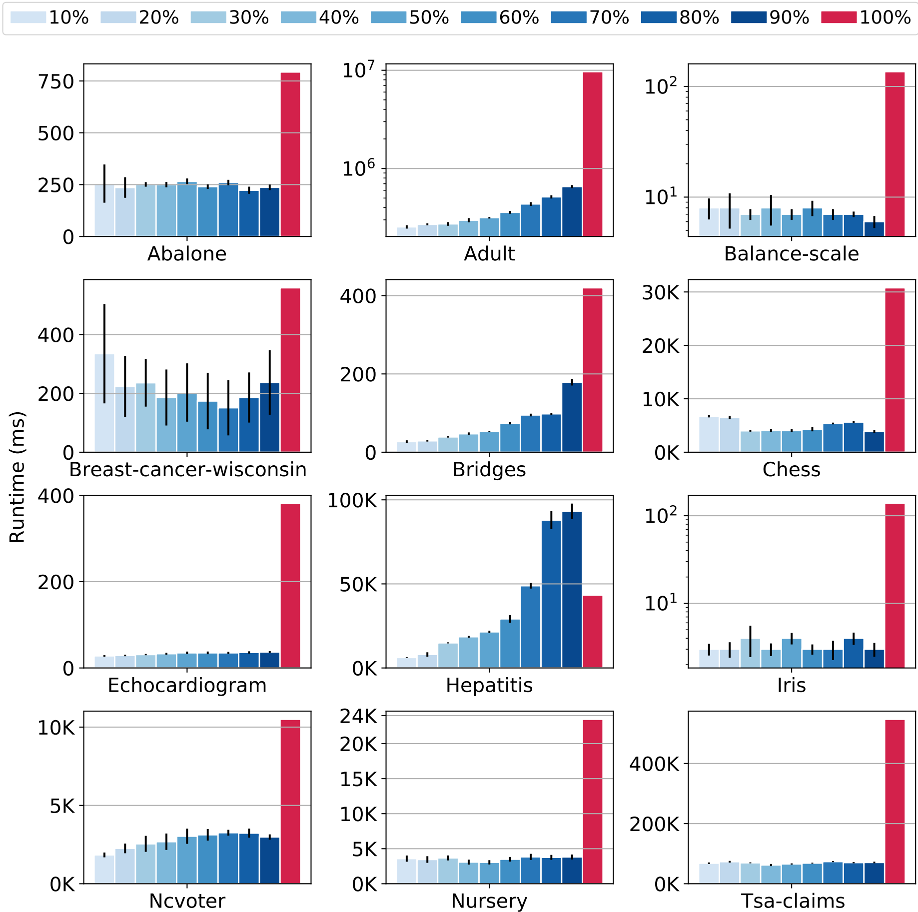


Figure 5.27: Time performances of BIRD with $g3$-error threshold $\varepsilon = 0.2$ and $P_{\tau+1}$ in the range $[10\%, 90\%]$.

invalidation of all or none RFD$_e$s holding at time τ, since RFD$_e$s are few in general.

Finally, with respect to the times observed from the execution of TANE (red bars) on the complete datasets, we notice that in most cases BIRD^p outperforms TANE by a different order of magnitude. The only exception is highlighted for the *Hepatitis* dataset, on which BIRD^p obtains worse

performances with respect to TANE when the number of tuple insertions is greater than 60%. In this case, TANE better exploited the pruning strategies. In particular, the *Hepatitis* dataset is the one containing the highest number of RFD$_e$s. Thus, we expected that with many inserted new tuples BIRD^p has to consider a large variation of holding RFD$_e$s.

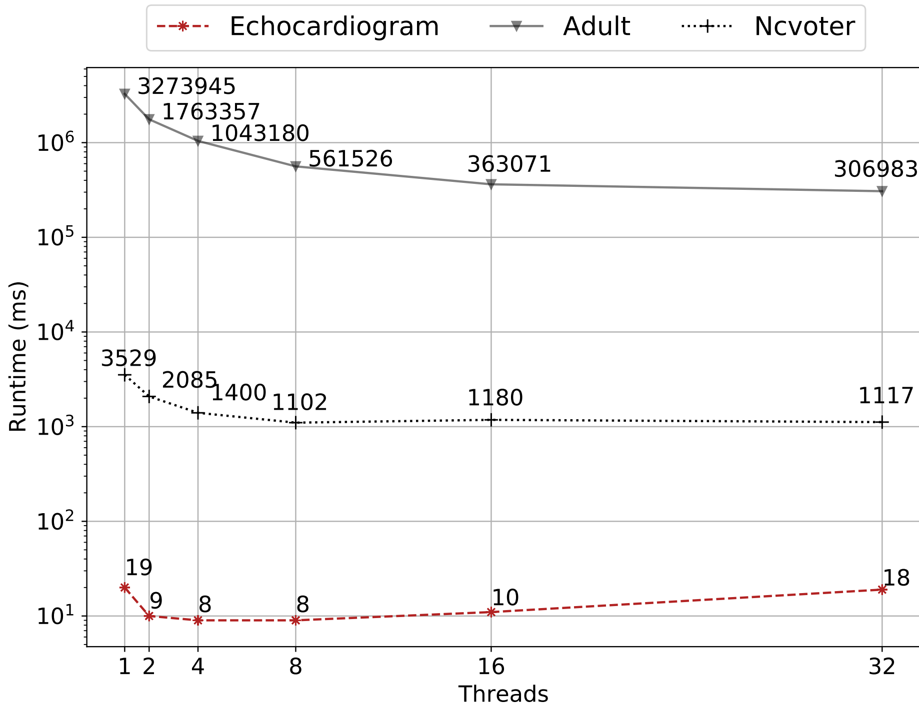


Figure 5.28: Scale-up performances of BIRD^p.

5.6.4.2 *Scale-up evaluation*

We performed a further experimental session to evaluate the communication overhead introduced by the parallelism used by BIRD^p. To this end, we considered a configuration of the algorithm with 2^i available CPUs, varying i in the range $[0, 5]$, $P_{\tau+1} = 50\%$, and $\varepsilon = 0.4$. We analyzed the time performances on *Echocardiogram*, *Adult*, and *Ncvoter* datasets to

understand how datasets with different number of rows and columns impact on the parallelism performances.

Figure 5.28 shows the execution times observed for $BIRD^p$ in this experiment. We can notice a decreasing trend when the number of worker threads increases from 1 to 8, especially for *Adult* and *Ncvoter*. This is due to the fact that worker threads are lock-free and they need to be synchronized at the end of their computation life-cycle, i.e., when the results are included into the final set of RFD_es. This synchronization process represents the biggest $BIRD^p$ overhead, which can degrade time performances when the number of threads exceeds a given value, as depicted in Figure 5.28 for the *Echocardiogram* dataset. For the other two datasets we can notice that the execution times reduced the sequential performances by two-thirds for *Ncvoter* and one-tenth for *Adult*.

Part III

TOOLS FOR VISUALIZING PROFILING METADATA

6 VISUALIZATION AND MONITORING TOOLS FOR INCREMENTAL DISCOVERY ALGORITHMS

As discussed in previous Chapters, the problem of discovering FDs and RFDs from data is an extremely complex one, and has challenged many researchers to investigate efficient algorithms enabling their discovery from "big" data collections in both static and dynamic scenarios. In this chapter, we focus on dynamic scenarios, where the goal is to get holding FDs/RFDs even when the input data dynamically change over time, such as with data streams. In this scenario, a proper analysis of how RFDs change over time cannot be accomplished by looking at a possibly huge number of holding RFDs over millions of time instants. Thus, this problem yields the necessity to provide novel tools and/or visual metaphors to graphically visualize them so as to facilitate their analysis. To this end, in this chapter we present three tools for visualizing the evolution of discovered RFDs during continuous discovery processes. These tools also enable the comparison among RFDs discovered across several executions, by means of visual manipulation operators that permit to dynamically compose and filter results.

6.1 PROBLEM DESCRIPTION

Other than several proposals of algorithms for automatically discovering FD and RFDs from data, some recent algorithms deal with their discovery from dynamic datasets (see Section 5.1). Although the discovery problem is per se an open challenge, it is important to consider that not only the computational complexity represents one of the main issues to tackle, but a proper visualization of results should be addressed [36]. In fact, as discussed in previous Chapters, the discovery of FDs is an extremely complex, especially when the number of attributes increases (see Section

4), and it considerably grows when introducing some relaxation criteria on the canonical definition of FDs, yielding RFDs (Section 5). On the other hand, the problem of visualizing results has to consider their quantity and complexity. In fact, FD and RFD discovery algorithms often output a huge set of dependencies, with many different combinations of thresholds, which makes it difficult for a user to grasp useful insights from them. However, while the research community is focusing on improving the efficiency of FD and RFD discovery algorithms, few efforts have been devoted to devise effective visualization metaphors, capable of summarizing the characteristics of the most relevant dependencies holding on a given dataset. To this end, in this chapter, we focus our discussion on three new tools for monitoring and visualizing metadata extracted from discovery algorithms, and on managing them in real-world application scenarios.

The first tool, namely DEVICE [26], permits to monitor the set of RFDs extracted from static and incremental discovery algorithms through a lattice representation. Moreover, it enables the user to interact with the discovery results by zooming on the search space and/or filtering results according to specific attributes or threshold settings.

The second tool, namely STRADYVAR [21, 22], permits to analyze and compare RFDs dynamically extracted from data streams, enabling users to monitor their evolution over time. In particular, with STRADYVAR users can *i*) visualize the trend related to the number of RFDs holding on a data stream over time, *ii*) visualize an overview of the correlations between attributes included in the discovery results over time, *iii*) compare the RFDs resulting from different executions of a discovery process on the same or different data streams, and *iv*) directly manipulate discovery results by composing those across different executions and/or grouping RFDs according to specific Right-Hand-Side (RHS) attributes.

Finally, the third tool, namely INDITIO [30], permits to monitor and validate in real-time specific profiling metadata upon data insertion operations. It has been conceived as a plugin of the graphical client MySQL Workbench, and it enables the user to intercept data insertion queries, in order to validate specified (or uploaded) metadata before the insertion is committed. It also provides different statistical counters and

visual components enabling users to have an overview of the general impact of data insertions on the considered metadata.

6.2 LITERATURE REVIEW

Although recent algorithms are becoming capable to scale over big data collections, there are only a few solutions in the literature for handling the complexity related to the visualization of a possibly huge number of discovered RFDs. The first proposal for visualizing large sets of RFDs is described in [36]. 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. Moreover, among the recent proposals, two of the most effective platforms for data profiling are the Metanome project [125] and Metacrate [92]. The former embeds several algorithms to automatically discover complex metadata and various result management techniques, such as list-based ranking techniques, and interactive diagrams of discovery results. The latter permits the storage of different metadata and their integrations, enabling users to perform several ad-hoc analysis. However, none of the previous platforms allow users to monitor the results of the discovery algorithms. Moreover, they are mainly designed for domain experts, and not for users who are not familiar with advanced database technologies.

A related research context is represented by visual data mining, for which in the literature there are several approaches and tools aim to improve the understanding of data mining algorithms and their results (see [54] for a survey). Among these, it is worth mentioning Association Rules (ARs) visualization approaches, since the concept of AR is somehow related to that of RFD. For instance, several tools/packages have been designed to visually inspect the set of ARs [75, 142]. Moreover, ad-hoc visualization techniques have been introduced, such as the hierarchical matrix-based [39], or the hybrid matrix- and graph-based [157]. They typically show summarized representations of result sets, and provide mechanisms to filter results. On the other hand, due to the huge quantity

of possible rules, in [40] a visual tool for searching rules has been introduced. It interactively guides users in the definition of the target rule by drawing details of minimum, current, and potential support/confidence measures, based on already or potentially selected attributes. Finally, also in the context of ARs, an enhanced visualization technique has been defined in order to highlight changes among different sets of ARs [122].

Although all of these proposals represent effective tools to visualize and explore properties and metadata after the execution of mining/discovery algorithms, they do not enable users to monitor the discovery processes during their executions. Moreover, even if in the literature several time-related visualization approaches/tools have been proposed [87, 112, 141], they only focus on static scenarios, not meeting the recent need of researchers to explore the evolution of metadata in dynamic scenarios (see Section 2). To this end, the new tools presented in the following sections provide users the possibility to monitor the evolution of RFD over time and to perform result comparisons across different executions.

6.3 DEVICE: A TOOL FOR MONITORING THE EVOLUTION OF RFD DISCOVERY ALGORITHMS

In this section, we present DEVICE (DEpendency VIsualizer on lattiCE) a visual tool that permits monitoring the set of RFDs extracted during the execution of a discovery algorithm through a lattice representation. DEVICE enables the user to interact with the discovery results by zooming on the search space and/or filtering results according to specific attributes. In particular, we first present the system architecture and then provide details on the visual interface and the interactions that a user can perform.

6.3.1 *System Overview*

Monitoring the results of RFD 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 rely on different technologies and frameworks, so requiring the integration of

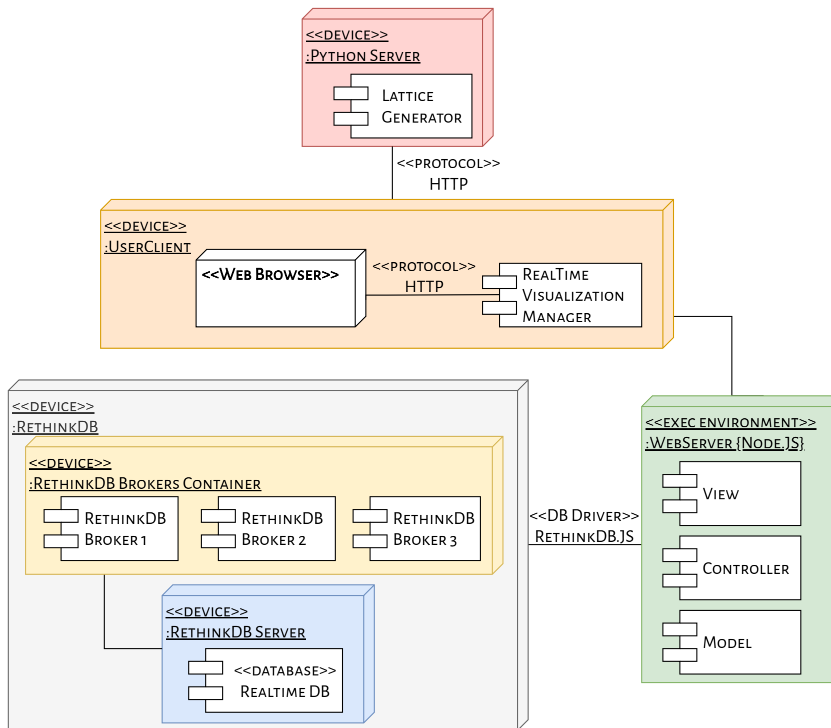


Figure 6.1: The system architecture of DEVICE.

at least one module to adapt the system to the different algorithms; *ii*) the presence of multiple visualization components requires frequent updates in a short time, and *iii*) the number of resulting RFDs can be large at any instant of time. For these reasons, we propose a modular client-server architecture for enabling users to monitor RFD discovery algorithms during their executions, letting them to interact with the results through a responsive visual interface. The architecture of DEVICE is shown in Figure 6.1. In detail, it 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 maintainability, aiming to facilitate the upgrade or replacement of 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. Lattice nodes contain all the possible combinations of attributes, whereas edges contain all the existing links between two nodes of consecutive lattice 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, in order to create a low latency and a 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 produced by each algorithm, and to continuously monitor the results of each execution. To this end, the *Input Driver Connector* receives the dependencies from an 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 to manipulate in the interface. As said before, the proposed tool is also able to handle continuous discovery algorithms [118] and therefore it requires to maximize fluidity and minimize processing times within the visual interface. Thus, all selected technologies for both client- and server-side support real-time update of data.

1 https://rethinkdb.com

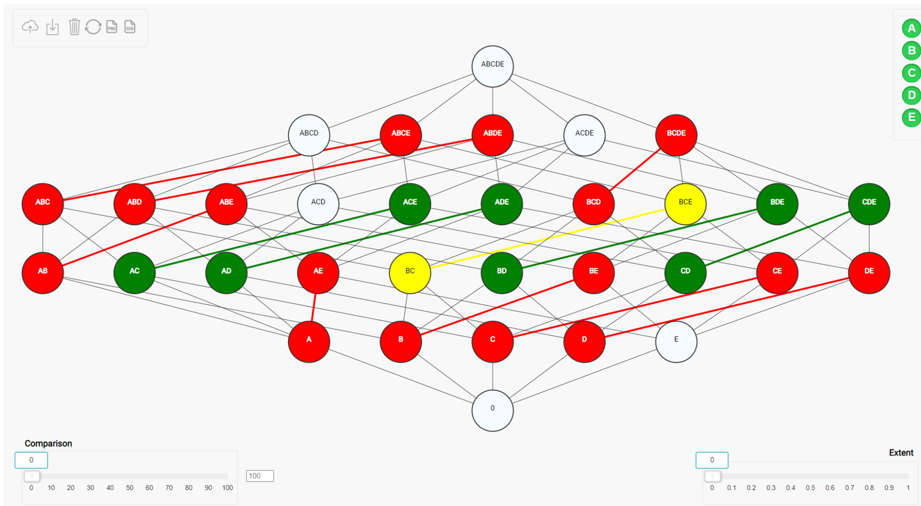


Figure 6.2: The visual interface of DEVICE.

6.3.2 RFD *Visualization*

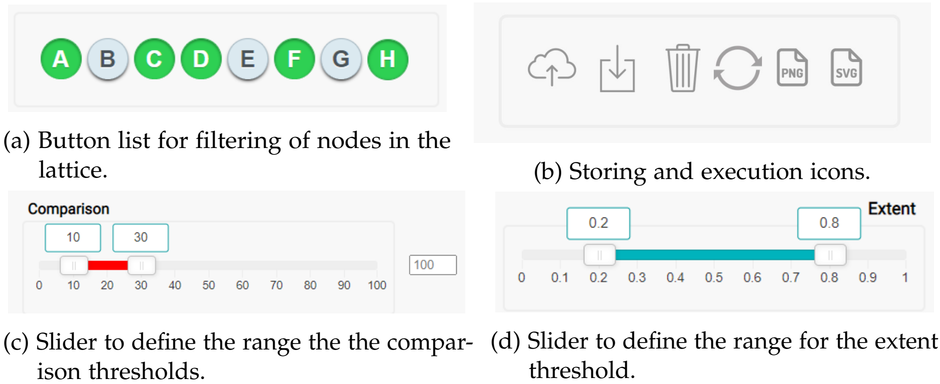
Due to the possible huge number of holding RFDs on a given set of processed data, tools for their visualization should enable users to quickly analyze results, also providing the possibility to directly manipulate them. For this reason, a static representation of discovery results after the execution of algorithms limits the analysis of how dependencies evolve over time, which is particularly interesting in dynamic scenarios, like in the case of data streams.

The dynamic representation of a large amount of metadata requires the application of interactive graphs, capable of highlighting how information evolve over time. To this end, 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 6.2). As said before, the lattice permits to show candidate RFDs through the lattice edges, which connect attribute combinations so to represent the LHS and the RHS of a candidate RFD in a compact way. Moreover, the lattice graph is responsible for displaying

information about the candidate RFDs that have been validated during a discovery process. As shown in Figure 6.2, the lattice graph uses different colors during the execution of a discovery algorithm. 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, two linked nodes assume the same color of the edge connecting them, which represents the last analyzed candidate RFD. 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, they could browse the search space in a different way. Thus, DEVICE enables the comparison among different discovery algorithms and the analysis of possible bottlenecks during their execution on a given dataset. The visual interface of DEVICE also provides different gadgets enabling users to directly manipulate the lattice graph. Moreover, the vectorial representation of the graph also permits zooming on or moving each lattice component without losing the quality of the representation. Details on how users can manipulate with the lattice graph are provided in the following section.

6.3.3 *Interaction in depth*

As introduced in the previous section, users can directly manipulate the lattice graph by simply zooming a specific part of the search space, or by moving its components into the visual interface, aiming to highlight the discovery results on a specific part of the search space. To this end, the user can place the mouse pointer in correspondence with a lattice node and drag it to 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 visual representation of the search space, by using the button list on the top-right corner of the visual interface (also shown in Figure 6.3a). In particular, each button represents an attribute



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

(b) Storing and execution icons.

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

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

Figure 6.3: DEVICE gadgets to interact with the lattice graph.

of the analyzed dataset, 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 6.4 shows the representation of a lattice graph with 5 attributes after the exclusion of attribute A.

Concerning the RFD settings, DEVICE permits to visualize discovered RFDs by filtering results according to specific relaxation parameters. Figure 6.3d highlights a slider that can be used for defining a specific range for the coverage measure threshold. In this way, the colors of the lattice components 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 distance constraints for the relaxation on the attribute comparison (see Figure 6.3c). 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, for sake of simplicity, a lattice edge is colored when at least one candidate RFD satisfies difference thresholds bounded by the range, without showing all possible dispositions of thresholds.

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,

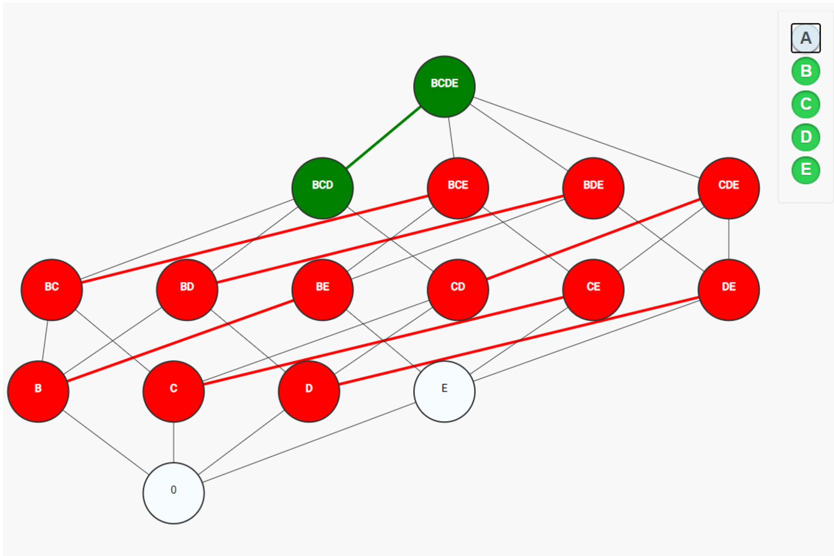


Figure 6.4: The visual interface of DEVICE after filtering out the attribute A.

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, sliders are set to $[0, 0]$ and cannot be modified. 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 to which a discovery algorithm is devoted to.

Finally, the icons in Figure 6.3b enable the interaction with the monitored execution and the download 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 results of the monitored discovery process. More specifically, the third icon permits to refresh the monitored results, by cleaning the lattice representation, whereas 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.

6.3.4 *Case Studies*

In what follows, we analyze the effectiveness of DEVICE in monitoring different algorithms, aiming to analyze how metadata evolve over time. In particular, we performed two different case studies on real-world datasets and real sensor-based streams, respectively. A demonstration video of DEVICE[2] allows us to show how the users can interact with the tool during the monitoring of results produced by RFD discovery processes.

6.3.4.1 *Case study on a real-world dataset*

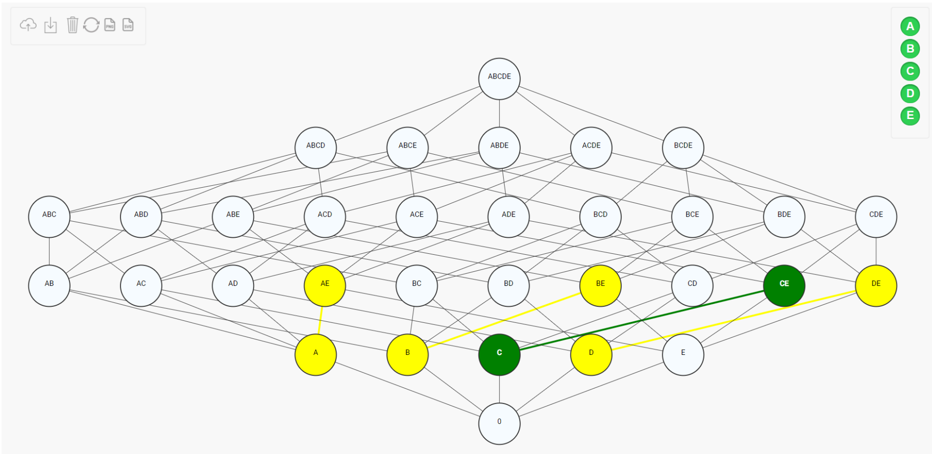
In order to analyze the effectiveness of DEVICE on a real-world dataset, we selected two different discovery algorithms to evaluate how their discovery strategy browses the search space.

The first algorithm involved in our evaluation is REDEVO, the genetic algorithms presented in Section 4.3. Instead, the second one is the incremental discovery algorithm COD3, which was presented in Section 5.3.

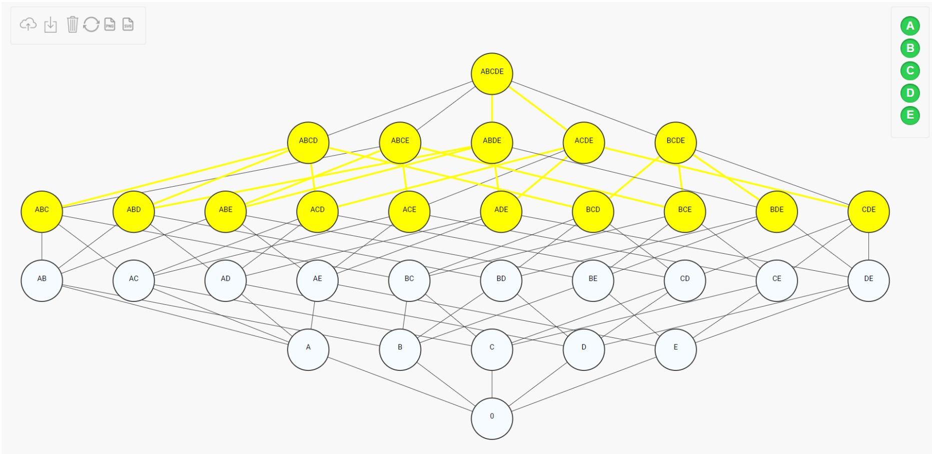
According to the characteristics of the considered algorithms, and to enable a proper comparison of different executions, we set parameters of REDEVO to discover canonical FDs. In particular, we choose the abovementioned types of algorithms since they both use several iterations to get results. Nevertheless, their nature is quite different since REDEVO analyzes new candidate FDs at each iteration by always considering the complete set of tuples; instead, COD3 analyzes new tuples at each iteration by considering the FDs 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 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. In order to perform our evaluation, we ran each algorithm on the

2 https://youtu.be/QC2FjF50A60

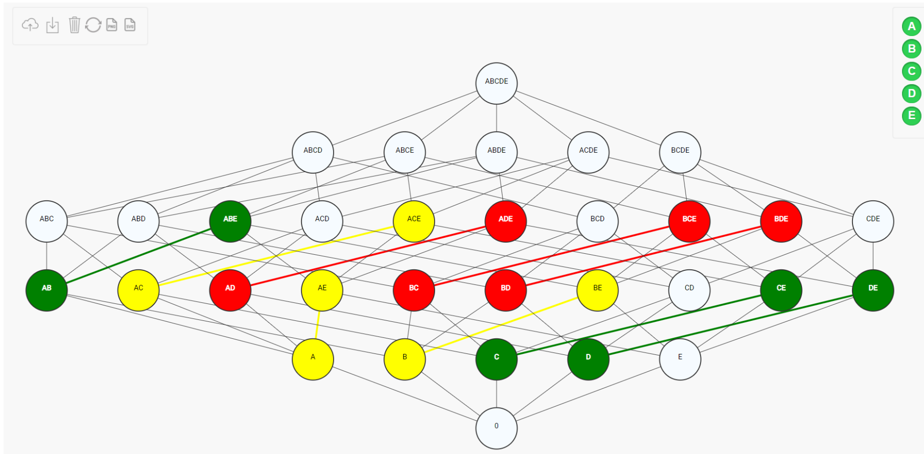


(a) 25% execution of COD3

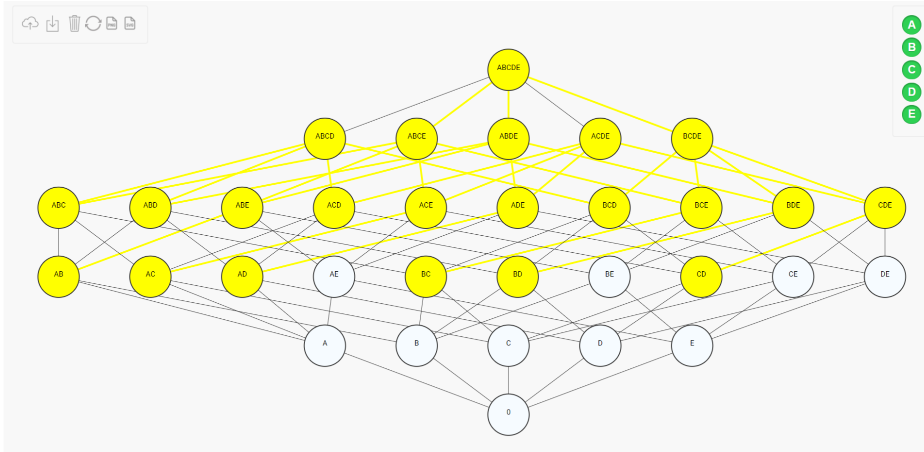


(b) 25% execution of REDEVO

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 executions. For the sake of clarity, we only report the screens at 25%, 50%, 75%, and 100% of their executions (Figure 6.4). More specifically, Figures 6.4a and 6.4b show the evolution of the discovery process for the incremental and genetic algorithm, respectively, at the

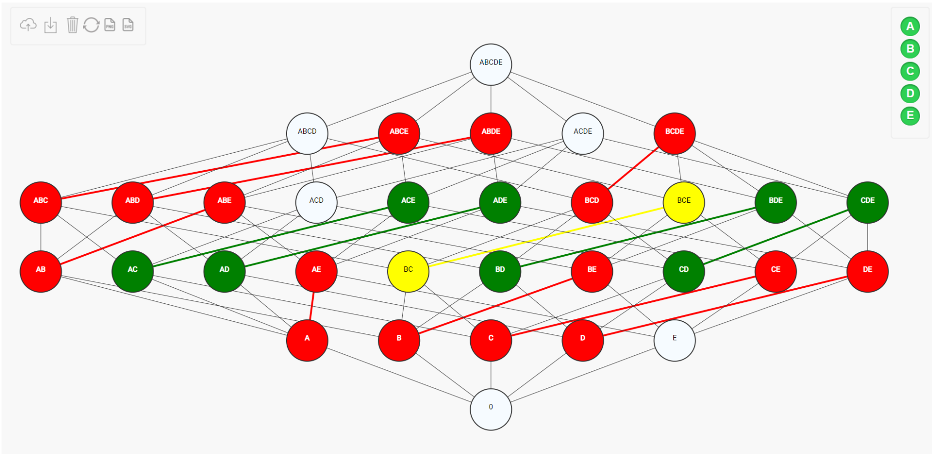


(c) 50% execution of COD3

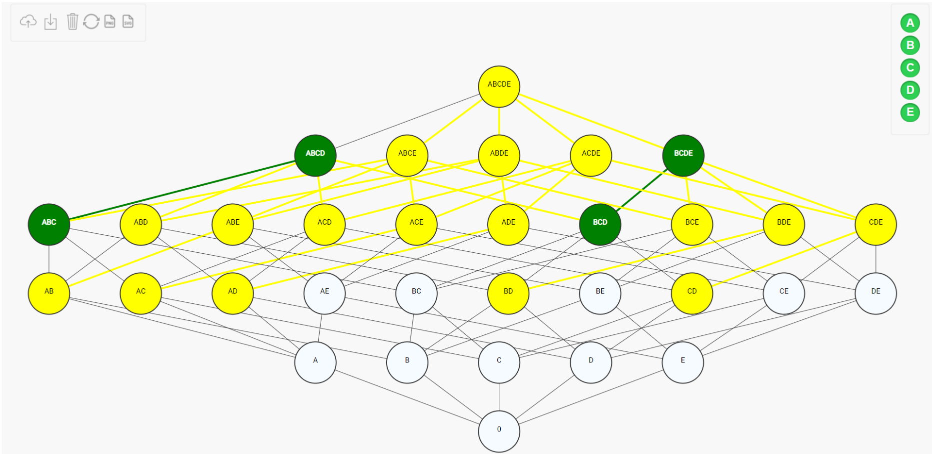


(d) 50% execution of REDEVO

25% of their executions. They highlight the difference between the two discovery strategies. In fact, REDEVO starts by considering candidates in the middle of the lattice, and then performs an evolutionary discovery step throughout the search space. However, COD3 first considers candidates from the lowest lattice level, and then goes up by performing a targeted search. Another relevant difference between the two search

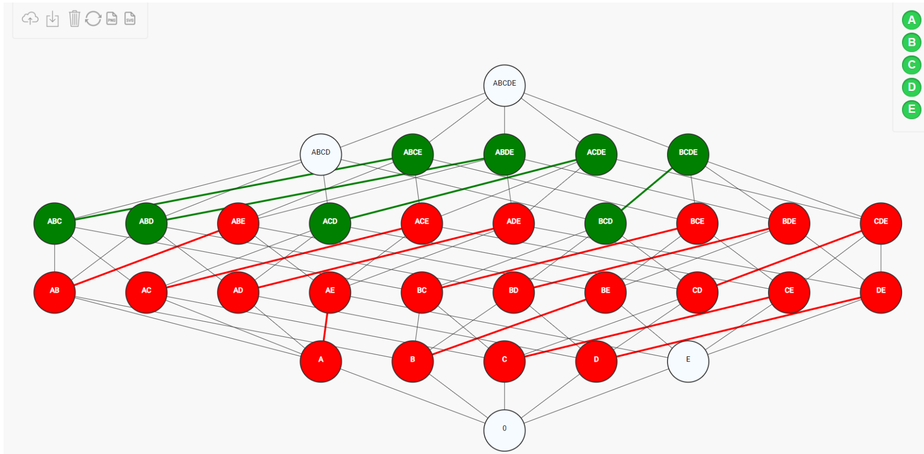


(e) 75% execution of COD3

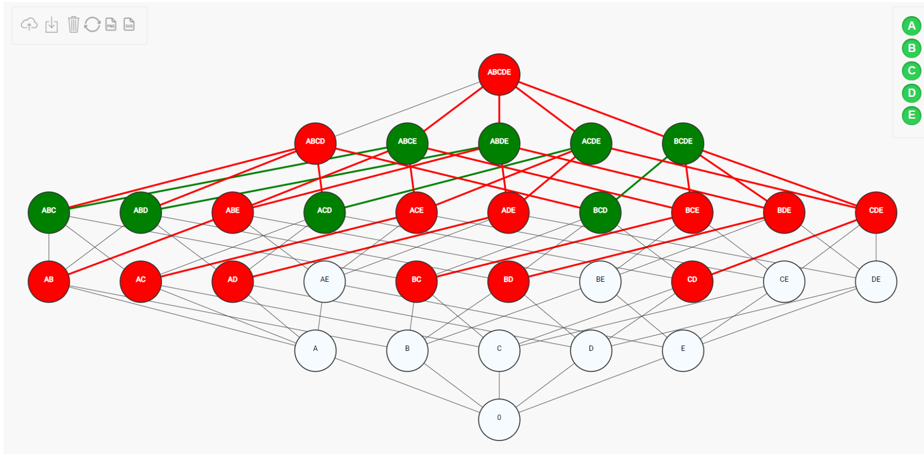


(f) 75% execution of REDEVO

strategies concerns the validation strategy of the candidates. In fact, REDEVO exploits an a posteriori validation strategy of the candidate FDs, which allows to define the first valid and invalid FDs only after 50% of the execution (Figures 6.4d, 6.4f, and 6.4h). On the contrary, COD3 updates the FDs validated at the earliest executions, according to the dynamic change of the dataset, and exploits this information to browse the search



(g) 100% execution of COD3



(h) 100% execution of REDEVO

Figure 6.4: Monitoring COD3 and REDEVO algorithm during its executions.

space (Figures 6.4c, 6.4e, and 6.4g). However, as expected, when algorithms end their execution (Figures 6.4g and 6.4h) they obtain the same set of resulting FDs.

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

and to concretely compare the search strategies of different discovery algorithms.

6.3.4.2 *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 COD3 on data from 1,000 real sensors spread throughout Italy, made available by the *Openweathermap* portal[3], without configuring any sliding window. 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);
- Ground_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 6.5 shows the resulting FDs for each time interval. We can notice that the number of resulting FDs has a

3 https://openweathermap.org/

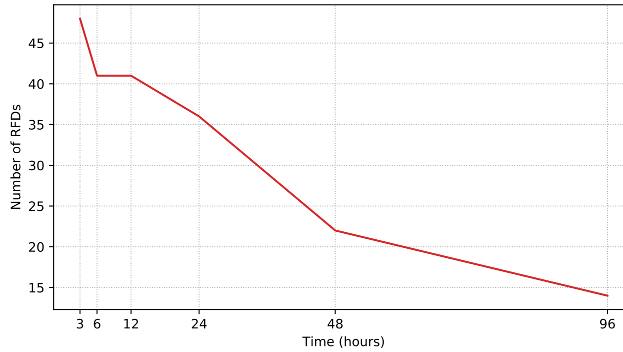
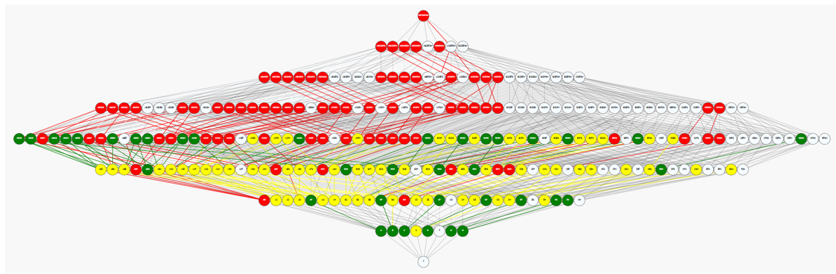


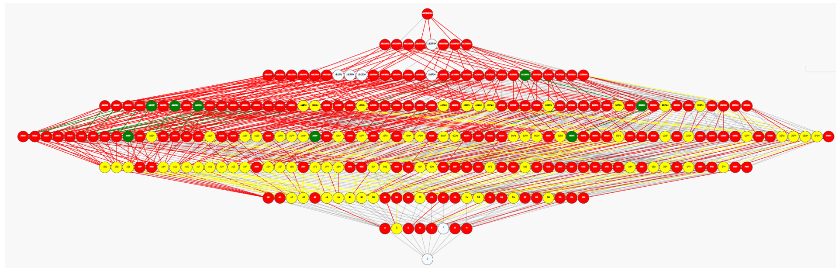
Figure 6.5: Resulting FDs from the executions of COD3 on real streams.

negative trend, since the continuous insertion of new tuples has led to many invalidations. Moreover, since COD3 has been executed without any sliding window, it validates FDs by considering all data previously read from the stream. This means that the initial number of FDs, i.e. FDs involving few attributes, probably evolve when the algorithm considers new tuples. To get some insights on the FD validation trend, DEVICE permits to interact with its interface and explore the search space to concretely analyze how FDs evolve over time.

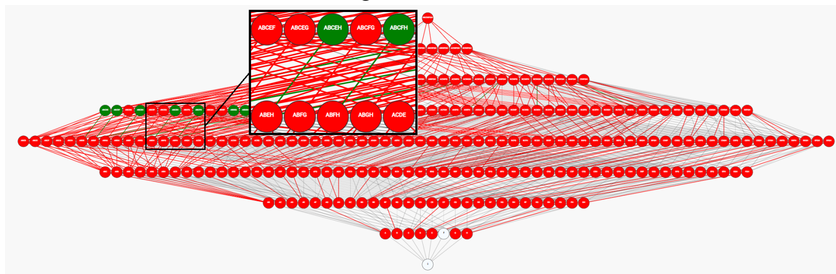
Figure 6.6 shows the details of the discovery process, by considering three different time intervals, 3, 48, and 96 hours, respectively. As expected, DEVICE shows that the algorithm has a large variation in the number of FDs after 3 hours, and a small number of invalid FDs (Figure 6.6a). Moreover, as 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 candidate FDs that can be directly inferred. Figure 6.6b and 6.6c show that many of the FDs validated after 3 hours have been invalidated. Moreover, Figure 6.6c shows that the algorithm also analyzed many candidate FDs in the search space that had not analyzed before. This is due to the invalidation of many FDs on the right side of the search space. In fact, after 96 hours only 14 FDs have been validated.



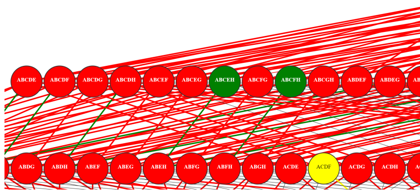
(a) Monitoring interface after 3 hours.



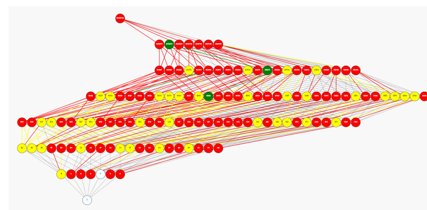
(b) Monitoring interface after 48 hours.



(c) Monitoring interface after 96 hours.



(d) Monitoring interface after zooming.



(e) Monitoring interface after filtering.

Figure 6.6: Monitoring COD3 executions on real data streams.

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 and the evolution of FDs holding on a data stream. In fact, at each time instant, an expert can concretely visualize and evaluate discovery results, and s/he can also monitor the evolution of holding RFDs, by configuring an RFD discovery algorithm working on data streams. Moreover, the different gadgets embedded in DEVICE let users to directly manipulate results during the monitoring process. For instance, the zoom feature (see Figure 6.6d) permits to focus the monitoring only on a specific part of the search space; whereas, the filter feature (see Figure 6.6e) 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 allow users to reduce the complexity of the visualization of discovery results, especially when they have to monitor big datasets and/or data streams.

Notice that there could be several other scenarios in which this kind of visualization tool could be applied, such as user behavior analysis in web browsing, social networks analysis, and anomaly detection. In particular, in the anomaly detection context, we can assume that a set of initial RFDs constitutes the normal functioning of a system. Therefore, by monitoring the evolution of RFDs over time, it could be possible to detect anomalies on the system under analysis when RFD violations are identified.

6.4 STRADYVAR: DEPENDENCY VISUALIZATION IN DATA STREAM PROFILING

In this section, we present STRADYVAR (STReAm DependencY VisuAlizeR), a tool for analyzing and comparing RFDs extracted from dynamic data sources, enabling users to monitor their evolution over time. In particular, we first show an overview of the STRADYVAR tool, also detailing the architecture underlying it; then we detail the functionalities of STRADYVAR and the types of interactions it enables. Finally, we discuss

the results of a user study to assess the effectiveness of STRADYVAR and its functionalities.

6.4.1 *System Overview*

Analyzing dynamic RFD discovery results is an extremely complex problem. In fact, it is necessary to deal with several issues that lead to specific choices for designing the system architecture: *i*) the amount of RFDs processed at each time instant can be huge, *ii*) the presence of several visualization components could require frequent updates in a short time, and *iii*) discovery algorithms rely on different implementation technologies. To this end, STRADYVAR is based on a client-server architecture, and it has been designed to enable users to monitor results during the execution of FD and RFD discovery algorithms through a responsive visual interface. Moreover, the modules of STRADYVAR are standalone and share information with each other by using the JSON standard. This design choice ensures high modularity and maintainability, by enabling the substitution of any back-end component, provided that its output is formatted according to the JSON standard defined for the interaction.

Figure 6.7 shows the architecture of STRADYVAR. The client communicates with the back-end modules through live queries, and it is implemented as a web application consisting of three different interfaces: the first one enables the real-time monitoring of discovery processes; the second one enables users to compare the results of two executions of the same discovery algorithm, or of two different ones; finally, the third interface enables users to visually manipulate results across different executions.

In general, STRADYVAR permits to load either a dataset or a configuration file connecting a data stream, and consequently select a discovery algorithm for datasets or data streams. The back-end of STRADYVAR is based on a microservice architecture, exploiting the power and the flexibility of Docker[4] containers in order to create a highly scalable platform. In particular, the core of STRADYVAR is a long-lived Node.js application,

4 https://www.docker.com/

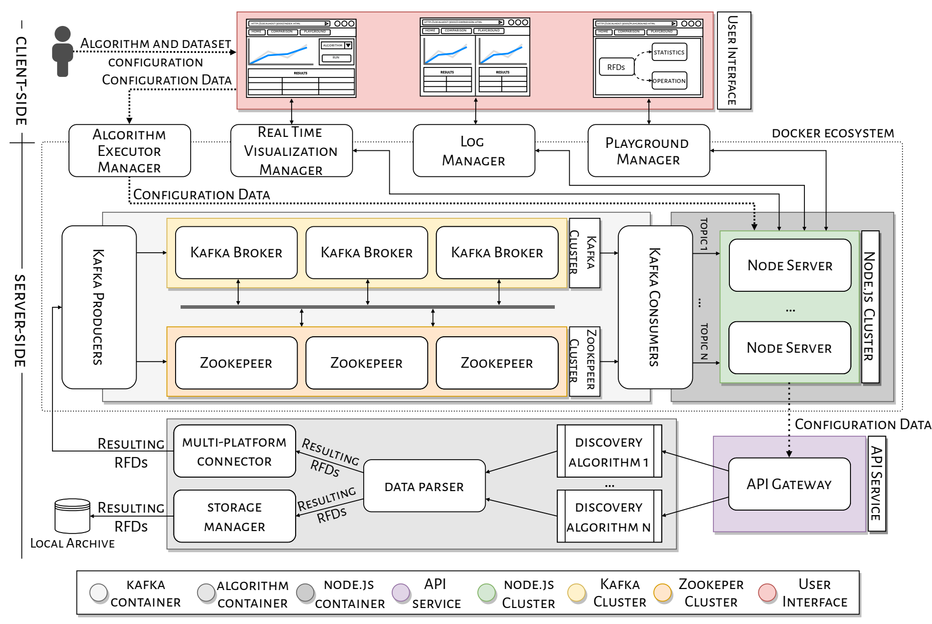


Figure 6.7: STRADYVAR architecture.

running distributed containers, while keeping a live connection with other services. Although the architecture is flexible and scalable, in order to quickly process a large number of messages, we integrated the broker Apache Kafka[5], which is a high-throughput and low-latency platform for handling real-time data feeds. In particular, Kafka temporarily stores key-value messages originating from several processes called *producers*.

Kafka requires the use of Zookeeper[6] servers to coordinate several producers and consumers. In fact, the architecture of STRADYVAR integrates these services into multiple docker containers, which directly communicate with the Node.JS instances through Kafka consumers. Moreover, to make STRADYVAR compatible with most of the FD and RFD discovery algorithms, it is necessary to uniform the syntax of the RFDs, regardless of possible thresholds. In particular, a parser module receives in input an

5 https://kafka.apache.org/
6 https://zookeeper.apache.org/

RFD, manipulates its syntax to extract its JSON version, so as to store it in a local file and in a Kafka topic. All the selected technologies for both client- and server-side support real-time updating of data.

6.4.2 RFD *visualization*

Most of the different solutions for handling the complexity related to the visualization of the discovery results, are not intended for the dynamic processing of data, since thwy focus on a static representation of resulting metadata. However, in the context of continuous data profiling useful insights can be gained by monitoring how FDS and RFDS evolve over time. In general, the dynamic representation of a large portion of data requires the application of interactive graphs, capable of highlighting the arrival of new information, without missing any pre-existing information.

Figure 6.8 shows the real-time monitoring interface of STRADYVAR, which permits the execution of an incremental discovery algorithm, and to monitor the trend and the details of validated RFDS in real-time during the execution process. In particular, the top-left corner contains the *line plot*, which is responsible for displaying information about the evolution of valid RFDS discovered over time. It can reveal stability or variability in the number of discovered RFDS, possibly highlighting variability in the correlation among data. Moreover, it becomes particularly useful comparing trends across different executions and/or filtering results according to specific time periods. More specifically, for each time instant the line plot contains a black line, a blue line, and possibly an orange line, showing the number of holding RFDS (black line), invalidated RFDS (blue line), and holding RFDS selected according to searching and/or filtering criteria (orange line). The plot is divided into two sections, the upper one is the main line plot, showing the evolution of the number of valid RFDS on its y-axis and the timestamp on its x-axis; whereas the bottom line plot enables the interaction with the user, allowing him/her to select a time interval through a brush operation, in order to visualize details on how the number of discovered RFDS evolved during the selected period. Both these line plots are continuously updated so that at any time instant

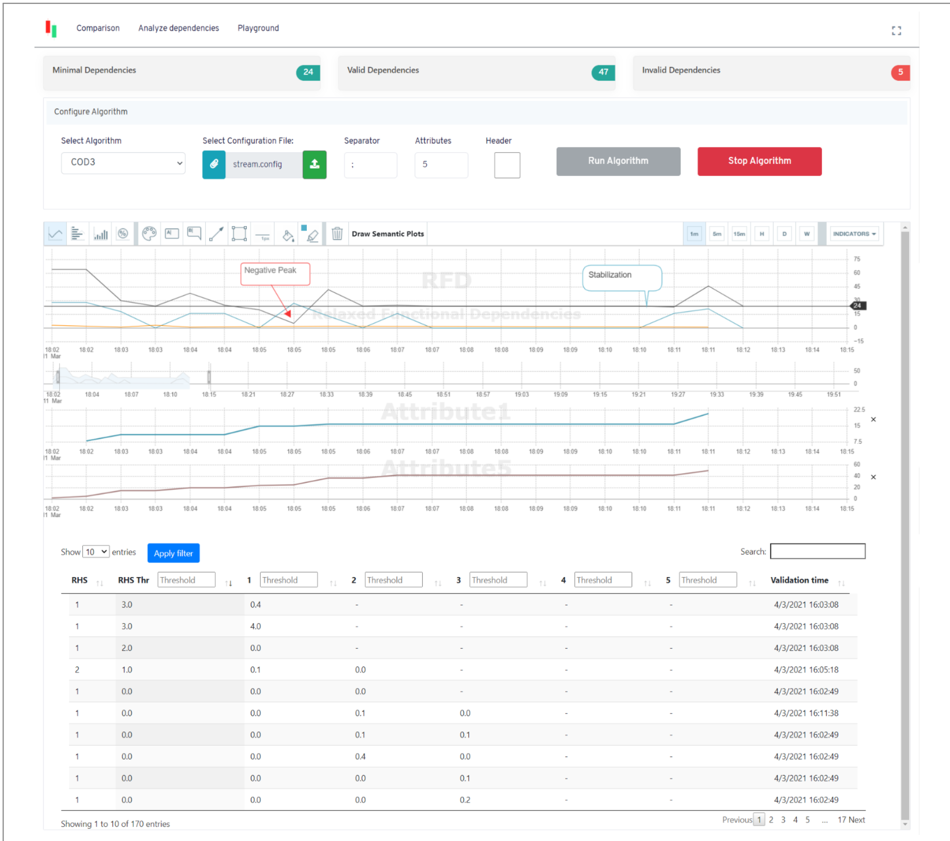


Figure 6.8: Real-time monitoring interface.

the user can see how the trend changes as the RFD discovery process progresses.

The *dependency table* displayed at the bottom of Figure 6.8 details the discovered RFDs. Each row represents an RFD, whereas each column represents an attribute of the dataset. In particular, the column labeled "RHS" contains the indices of the RHS attribute of each RFD. Instead, the other columns display details on all the attributes appearing in an RFD, followed by the time instant in which the RFD was discovered. Moreover, the whole table is continuously updated. In other words, the dependency

table is able to describe the implication property of an RFD and it also shows the difference thresholds, which specifically characterize each RFD.

Finally, the real-time monitoring interface also provides statistical counters (see the top of Figure 6.8) displaying the number of valid, invalid, and minimal RFDs, at a given time instant.

6.4.3 *Interaction in depth*

As mentioned above, the user can interact with the interface through the bottom line plot by selecting time intervals. The selection is highlighted with a light blue rectangle on the bottom line plot. During the monitoring 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 and zoom on the line plot (see Figure 6.8). At any time instant the user can verify whether the discovery process is still in progress, by stopping the brushing process. Moreover, as shown in Figure 6.8, the user has the possibility to annotate the line plot with several visual components. In fact, on the top of the line plot there is a toolbar through which the user can *i*) draw lines, arrows, polygons, and so on, *ii*) add some textual notes, and/or *iii*) modify the color of the plot (see the last two buttons on the toolbar). Finally, the toolbar offers the possibility to select some attributes.

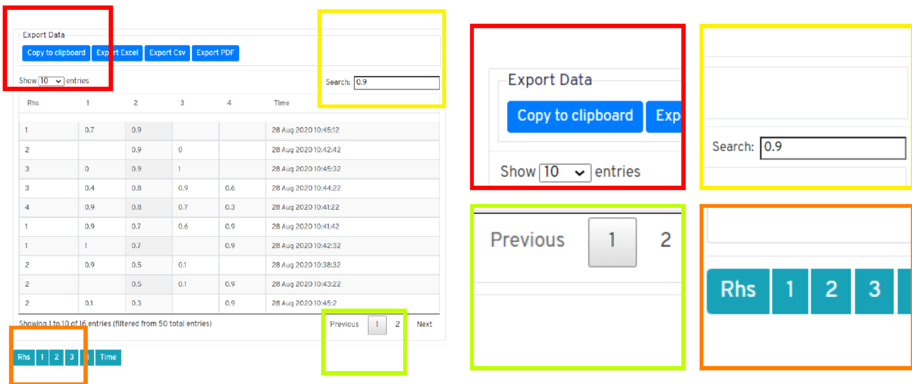


Figure 6.9: Interacting with the dependency table.

When an attribute is selected, a novel line plot is added at the bottom of the general plot, showing the number of holding RFDs that contain the selected attribute as RHS.

Concerning the dependency table, Figure 6.9 highlights all the interaction functionalities offered by STRADYVAR. In order to allow the user to reduce the table content, we added a search text field (Figure 6.9, yellow rectangle) enabling a global search over the attributes of the table, or a data filter, by searching on a specific column of the table. Based on the text the user inserts in the search text field, the table content is adapted, showing only the rows satisfying all enabled selection criteria. Moreover, a new line is added to the line plot, showing the number of holding RFDs for each time instant, also satisfying the selected criteria. Users can also define column-based sorting criteria by clicking on the chosen column and selecting either an ascending or descending order. We also added a paging functionality (Figure 6.9, green rectangle), allowing the user to decide the number of rows to be displayed in a single table page. Moreover, the buttons below the table allow the user to decide which column to hide or show in the table (Figure 6.9, orange rectangle). By pressing one of such buttons, the user can hide the corresponding column and make the associated rectangle turn black. By clicking on it again, it reverses this process.

Finally, at any time instant it is possible to copy and/or export the data contained into the dependency table to an external file (Figure 6.9, red rectangle). The export function supports several formats, such as Excel, CSV, and PDF. The exported file will include all the data that have not been filtered out, according to the user interactions. When interacting with the dependency table also the line plot is updated. In fact, it is possible to visualize a proper line that shows the trend in the number of RFDs, according to searching/filtering criteria specified by the user. It is worth noting that some searching/filtering criteria are specific of RFDs, such as those involving different thresholds.

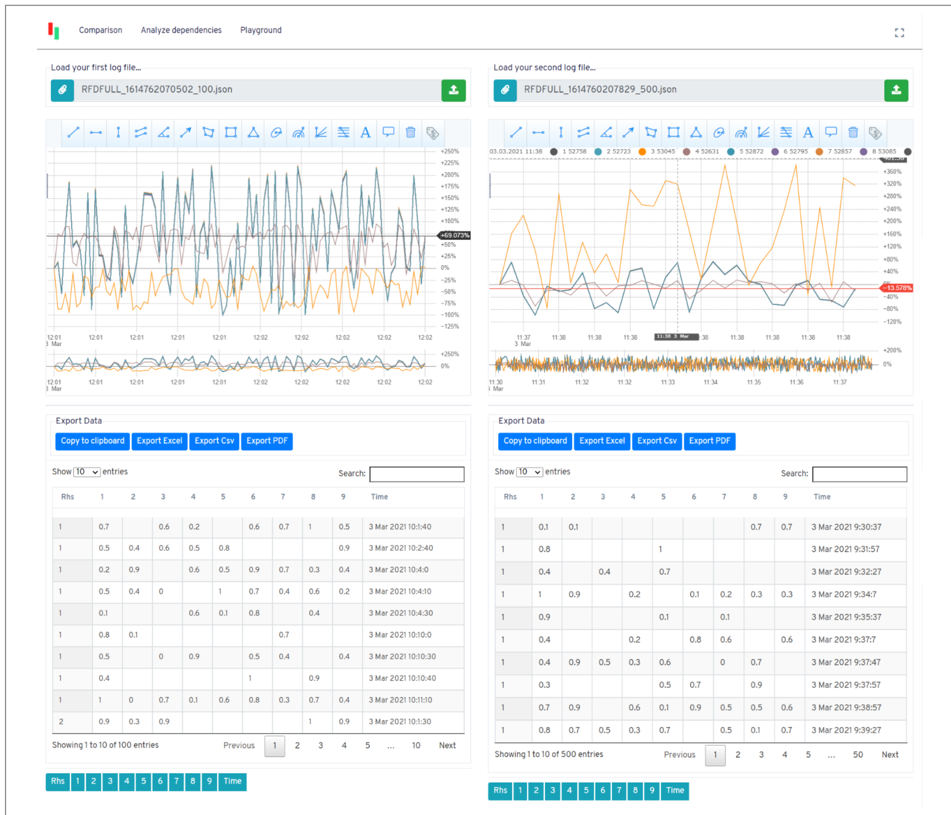


Figure 6.10: Comparing discovery results between two different executions.

6.4.3.1 *Comparing* RFDs

An important feature of STRADYVAR is the possibility to compare RFDs extracted across two different executions. It can involve different data, analyzed in different time periods, or results produced by different incremental discovery algorithms, which is made possible by loading log files from different executions. Figure 6.10 shows the visual interface devoted to the comparison of discovery results. As we can see, both the line plot and the dependency table are duplicated. They contain the evolution of the number of RFDs (line plot), and details on RFDs holding during the execution loaded from a specific log file. In particular, each line

plot shows lines concerning the evolution in the number of holding RFDs, and for each attribute, the evolution of the subset of holding RFDs having it as their RHS. Finally, by interacting with the bottom line plots, the user can select specific time intervals to be analyzed from each execution. This enables the user to verify commonalities across different executions of discovery algorithms, comparing them in terms of both the changes in the number of discovered RFDs and the RFDs validated at a specific time interval of the execution.

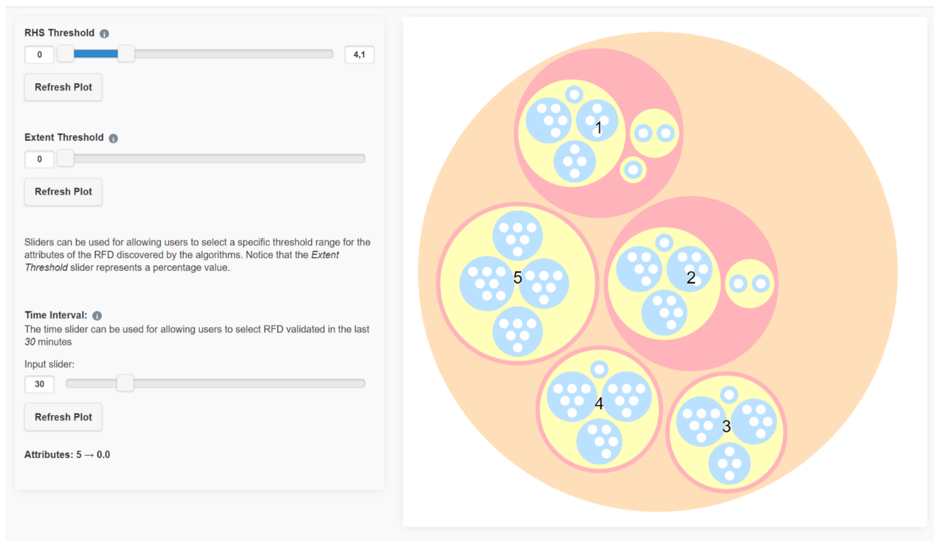
6.4.3.2 *Correlation analysis*

The monitoring interface (Figure 6.8) permits to show holding or invalidated RFDs according to the time variable and user-defined selection criteria. STRADYVAR also provides an additional overview of discovery results in terms of correlations among attributes.

Figure 6.11 shows the visual interface providing a general overview of attribute correlations. It represents each attribute as a circle, which in turn contains circles according to its characteristics and those of holding RFDs. In particular, the hierarchical chain of the circle containment, ranging from the general to the most specific RFD attribute, is defined as: *i*) RHS attribute, *ii*) RHS difference threshold, *iii*) LHS attribute, and *iv*) LHS difference threshold. The correlation plot interactively responds to mouse clicks by zooming the circle the user would like to analyze (see Figure 6.11 at the bottom). Several sliders on the left permit the user to filter out results according to an RHS difference threshold range (i.e., the difference thresholds bounding attributes on the RHS) or a satisfiability degree threshold (i.e., the upper bound of the percentage of admitted errors), which represent specific properties of RFDs. Moreover, the user can move the time interval to change the analyzed resulting set according to the time instant producing it.

6.4.3.3 *Playground*

The real-time monitoring and the possibility to compare discovery results with respect to the execution of possibly different discovery algorithms



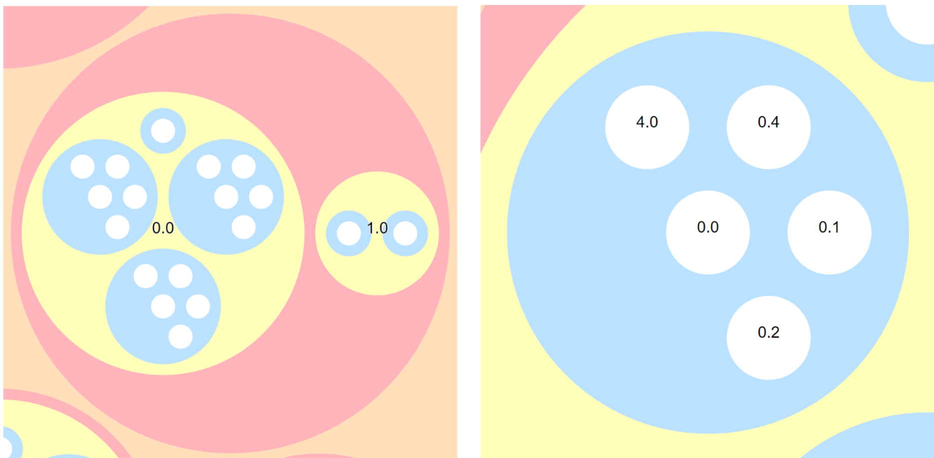


Figure 6.11: Analyzing the correlation among attributes according to holding RFDs.

represent valid means through which users can effectively explore result details and peculiarities. One of the main issues in analyzing results of RFD discovery algorithms is the possibly huge quantity of resulting RFDs. To this end, STRADYVAR provides interactive features enabling

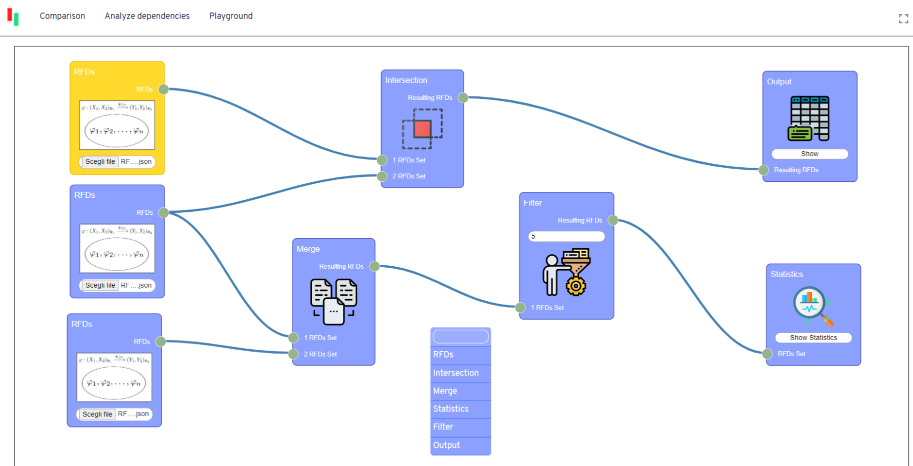


Figure 6.12: Overview of the Playground visual editor.

users to reduce the set of discovered RFDs by removing columns from the dependency table, filtering results, and exporting extracted RFDs. The *playground* module provides additional functionalities to support post-execution analysis of discovered RFDs. This is made possible through a visual editor equipped with several visual components named *blocks*, which enable transformations, and comparative/statistical analysis of RFDs discovered during one or multiple executions.

Figure 6.12 provides an overview of the playground module, whose aim is to enable users to link different blocks to form a sequence of operations, so as to determine a data flow (composed of RFDs) that is gradually updated on the basis of involved operations. Multiple flows can be created at the same time; the only constraint is that each flow has to start with one or more sets of discovered RFDs as input, and it has to terminate in one of the blocks enabling the visualization of some outputs. Blocks show both the number of required inputs (on the left) and the expected outputs (on the right), if any, by means of tiny circles, named *input* and *output* connectors, respectively. The user can require to insert a specific block and connect it to other blocks by simply clicking on the playground. More specifically, the playground integrates different types

of blocks, each representing a specific operation that can be applied to one (or more) set(s) of RFDs. Details on the blocks are provided in the following:

1. The block *RFDs* permits to consider a set of RFDs as input, enabling the selection of the execution results, and giving the possibility to forward the set of holding RFDs.
2. The block *Intersection* permits to forward only the RFDs shared between two sets. Consequently, it contains two inputs and one output connector.
3. The block *Merge* permits to forward all RFDs included in at least one of the sets provided in input, like with a union operation. Consequently, it contains two inputs and one output connector.
4. The block *Filter* permits to specify an attribute used to filter and forward only the RFDs having it on their RHS. Consequently, it contains one input and one output connector.
5. The block *Statistics* permits to visualize the number of RFDs linked to its input connector, grouped by each RHS attribute by means of a bar plot. An example of results that can be obtained by using this block is shown in Figure 6.13.
6. The block *Output* permits to visualize the RFDs linked to its input connector, by means of a dependency table, and on which the user can apply all types of interactions described above. An example of results that can be obtained by using this block is shown in Figure 6.13.

6.4.4 *User Study*

The user study presented in this section aims to show the effectiveness of STRADYVAR in enabling the analysis of discovered RFDs over time. In particular, we evaluated STRADYVAR with ten users (6 computer science students (2 MS students and 4 Ph.D. students) and 4 company's DBAs) who performed different tasks involving the execution of two discovery algorithms [27, 33], on both public datasets, i.e., *Abalone* and *Bridges*[7],

7 https://archive.ics.uci.edu/ml/datasets.php

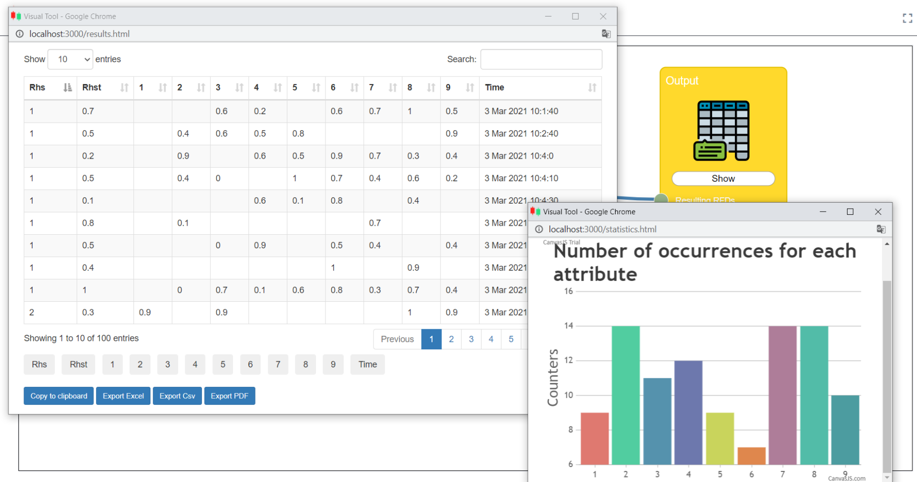


Figure 6.13: An example of the Statistics block usage.

and two sessions over the Twitter data streams lasting 8 hours. Statistics about participants have been collected through a background survey, as shown in Table 6.1(Q1-Q5). In particular, about 70% of recruited people were men, most of which graduated, and they declared a medium level of knowledge on data profiling, FDs, and RFDs by means of a Likert scale (see Figure 6.14, Q3-Q5). Prior the evaluation, they underwent a 30-min tutorial on data profiling and STRADYVAR. After performing a baseline task that required to run several discovery algorithms considering both data stream API configurations and real-world datasets, we requested participants to perform the following tasks:

T1 *Monitoring:* Select a time interval of one hour ending with a negative peak, search for an attribute on the RHS, filter RFDs with threshold 0 on another attribute, and order the set of RFDs by the RHS threshold.

T2 *Comparing:* Visualize RFDs resulting from different executions over Twitter data streams, and highlight (by using annotation visual components) at least two commonalities/differences on both the analyzed trends.

Survey	Alias	Question
Background	**Q1**	Gender
	Q2	Qualification
	Q3	Level of knowledge of Data Profiling and metadata
	Q4	Level of knowledge of FDs
	Q5	Level of knowledge of RFDs
Evaluation	**Q6**	I quickly and easily executed discovery algorithms over several types of datasets
	Q7	I completed the task T1 quickly and easily
	Q8	I completed the task T2 quickly and easily
	Q9	I completed the task T3 quickly and easily
	Q10	I completed the task T4 quickly and easily
	Q11	The tool is simple to use
	Q12	The tool is simple to learn
	Q13	I was able to retrieve back the process, whenever I made a mistake
	Q14	The user interface is pleasant and informative
	Q15	The tool made transparent the execution of the underlying discovery algorithms
	Q16	I understood the RFD syntax
	Q17	By using the tool, I can analyze the correlation among different attributes
	Q18	By using the tool, I can analyze the differences among different discovery results
	Q19	The tool presents all the features I expected
	Q20	I am in general satisfied about the tool
	Q21	In the future, I would like to use the tool
	Q22	What is your general impression about the tool?
	Q23	Do you have any improvements to suggest?
	Q24	Which feature of the tool did you like the least?
	Q25	Which feature of the tool did you like the most?

Table 6.1: Questions proposed to participants.

T3 *Correlating:* Analyze how the correlation between two attributes changes in the last 30 minutes, by also setting an RHS threshold upper bound to 3.

T4 *Exploring:* Consider two sets of discovered RFDs over Twitter data streams, find RFDs with a specific RHS attribute, occurring on both sets, and graphically show common RFDs.

In particular, to rule out learning and tiring effects that otherwise may compound the execution of later tasks, we carried out a random assignment of tasks for each participant. After completing the assigned tasks, we interviewed participants for receiving some feedbacks. In particular,

participants were requested to fill the questionnaire of Table 6.1 (Q6-Q25) to highlight the strengths and weaknesses of STRADYVAR. More specifically, the quantitative questions from Q6 to Q21 have been measured by means of a Likert scale ranging from 1 (Strongly disagree) to 5 (Strongly agree).

6.4.4.1 *Results and Discussion*

Figure 6.14 depicts the box plots derived from the answers of participants. A boxplot shows the median (horizontal lines), the interquartile ranges (boxes), the largest and the smallest observations (whiskers). Concerning the quantitative questions from Q6 to Q21 in the questionnaire, STRADYVAR obtained the general agreement of participants, while evaluating its usability and effectiveness. In particular, according to answers for Q9, the analysis of attribute correlations has been considered the most simple task. Moreover, according to answers to Q18, the capability of analyzing differences among discovery results has been positively evaluated. STRADYVAR has turned out to be pleasant and informative according to answers to Q13 and Q14. However, different opinions have been provided on the algorithm execution transparency (see answers to Q15). In fact, the answers to Q15 appeared to be strongly influenced by the different outcomes participants expected from STRADYVAR, probably due to their working/studying environments.

The open questions in the final questionnaire (see Q21-Q25 in Table 6.1) aimed at highlighting the strengths and weakness of STRADYVAR. In general, they revealed that some participants remarked the necessity to better integrate the different visual interfaces and features, in order to enable the analysis to be performed. Conversely, they welcomed the effectiveness of STRADYVAR in working with discovery results even when they change over time. Moreover, they positively judged the intuitiveness and the interaction capabilities of all the visual components.

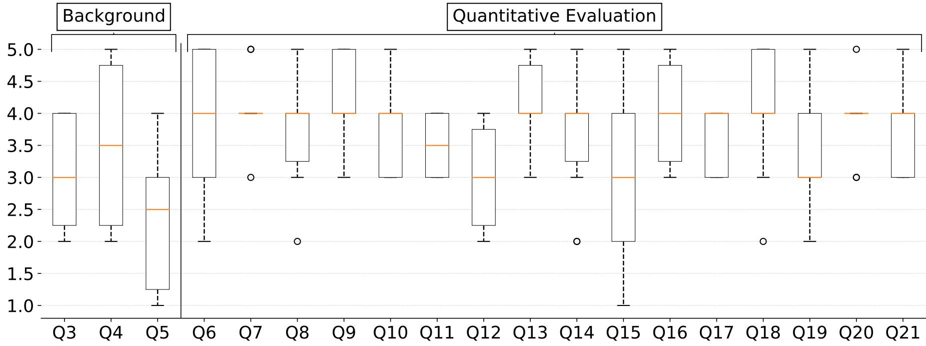


Figure 6.14: Distribution of user answers to quantitative questions.

6.5 INDITIO: REAL-TIME VALIDATION OF PROFILING METADATA IN A DATA MANAGEMENT SYSTEM

Data-intensive processes must deal with the problem of monitoring the quality of data. In this context, metadata can be exploited in order to highlight errors and support the cleaning of data. However, the use of raw approaches through algorithms mainly allows domain experts to use these techniques without considering users and developers that daily work with database technologies. To this end, we present a novel Database Management System (DBMS) plugin, namely INDITIO (INteracting with metaData during data InserTIOns), for validating profiling metadata during data insertions, and assisting users in checking a priori the quality of data being inserted into a database.

In this section, we first provide details about the INDITIO's visual interface, and then we show how users can interact with it. Finally, we discuss the results obtained from a user study aiming to highlight the effectiveness of the proposed plugin, and emphasize its strengths and weaknesses.

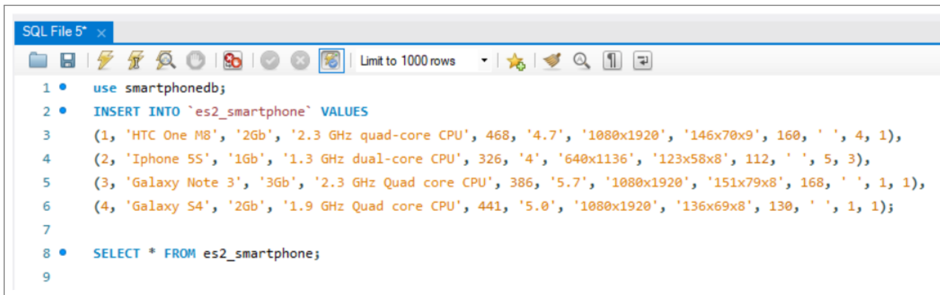


Figure 6.15: The MySQL Workbench SQL Editor.

6.5.1 *System Overview*

INDITIO has been implemented within the MySQL Workbench client[8], and it is able to intercept and verify in real-time whether the data to be inserted into a database instance will produce some violations on specific metadata, such as Unique Column Combinations (UCC) or Functional Dependencies (FDs). In particular, INDITIO is able to intercept data insertion queries provided by users into the SQL Editor (Figure 6.15). Due to the possible big number of errors that can be introduced during the insertion of new tuples, DBMSs should enable users to visualize profiling metadata that could possibly invalidate newly inserted data, by also giving them the possibility to interact with them. For this reason, it is necessary to evaluate such metadata upon data insertion operations. To this end, INDITIO extends MySQL Workbench functionalities by enabling users to validate UCCs and FDs upon the insertion of new tuples.

Figure 6.16 shows the general visual interface of INDITIO. In general, it permits to evaluate the impact of new data on a set of holding metadata. Thus, it enables users to visualize the new tuples being inserted (Figure 6.16(a)), the metadata to validate (Figure 6.16(b-d)), and the results of the metadata validation process (Figure 6.16(e-h)). Moreover, INDITIO also provides several functionalities enabling users to interact with both SQL

8 https://www.mysql.com/it/products/workbench/

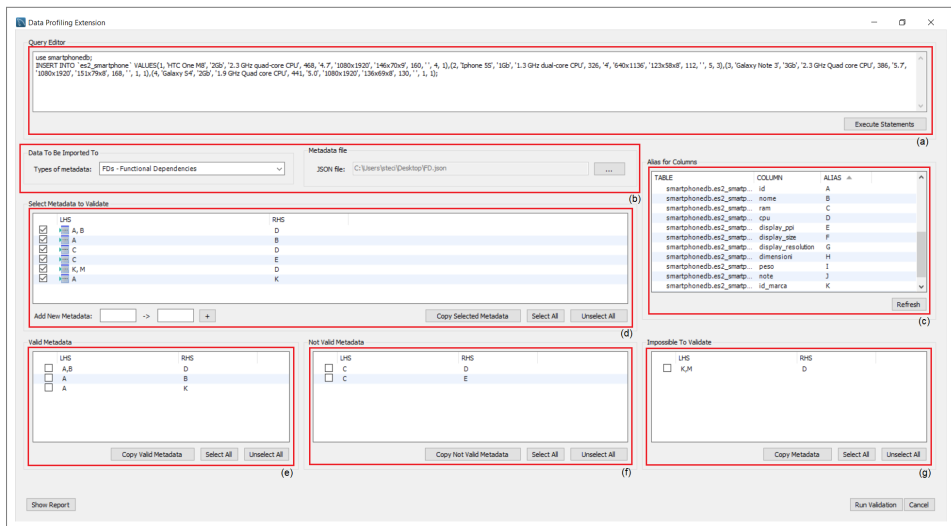


Figure 6.16: The INDITIO visual interface.

statements and metadata, as described below. A demonstration video of INDITIO is available on YouTube[9].

6.5.2 *Interaction in depth*

The main novelty introduced by INDITIO is the possibility to evaluate some metadata directly into the MySQL Workbench. A user can analyze UCCs or FDs by selecting the type of metadata s/he plans to monitor, and uploading a file containing them (Figure 6.16(b)).

The uploaded metadata is shown in the middle form (Figure 6.16(d)), which is customized according to the type of metadata the user selects. For instance, FDs are divided into LHS and RHS, each containing some attributes, in order to graphically visualize the implication property. Instead, the UCC customized form visualizes each metadata by considering a single group of attributes (Figure 6.17). Aside from the metadata uploaded via file, a user can always add new metadata. Moreover, through

9 https://youtu.be/u03Vftge8pA

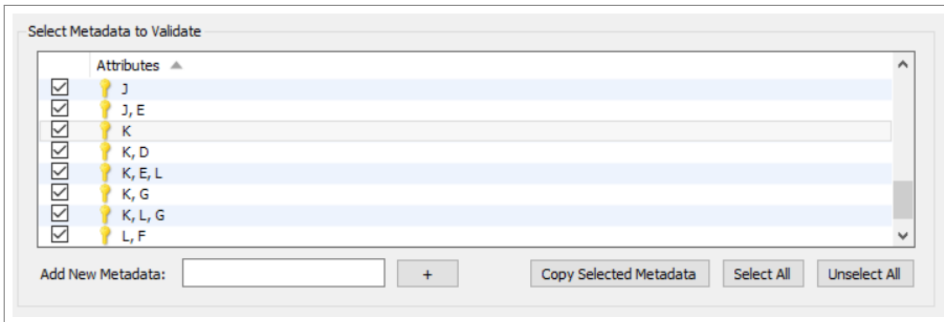


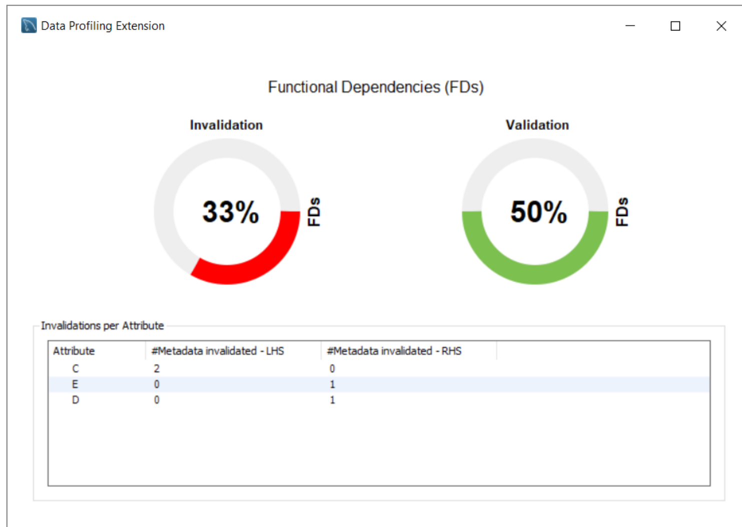
Figure 6.17: Visualization of UCCs.

this form, it is possible to select which metadata must be considered during the validation. Indeed, each metadata can be selected by means of the check box, and/or by using the "Select All" or "Unselect All" buttons. Finally, it is always possible to "Copy Selected Metadata" as text by means of a specific button. Notice that metadata is described through letters or numbers, e.g., alias, in order to identify attributes. This facilitates users in focusing on attributes and/or in defining new metadata. In fact, possible long (or inappropriate) attribute names could confuse the user. However, INDITIO provides a suitable form to show the mapping between attribute names and their associated alias (see Figure 6.16(c)).

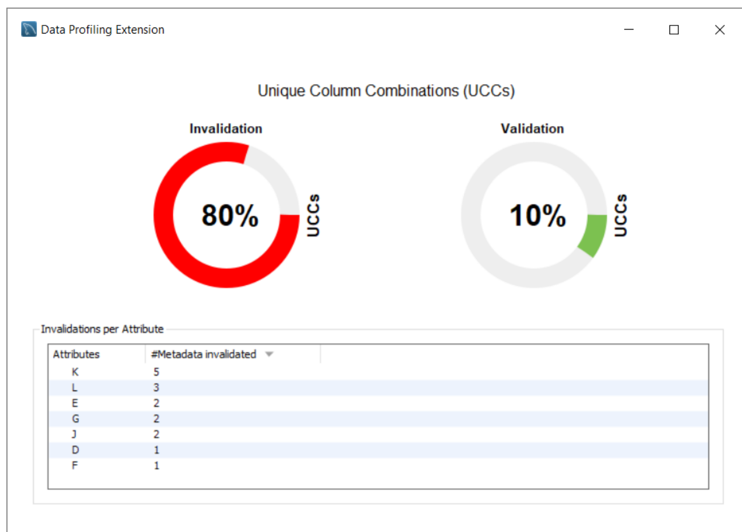
All selected metadata can be validated by clicking on "Run Validation", which triggers the execution of a validation module whose aim is to check if the new tuples violate the selected metadata. According to the validation process, each metadata can be classified in one of the following categories:

- *Valid Metadata*, when the new tuples do not produce any violation;
- *Invalid Metadata*, when the new tuples produce at least one violation; or
- *Impossible to validate*, when the metadata cannot be validated. This occurs when the user introduces errors in the metadata, such as when the attribute names do not exist in the considered database.

Example 1. Let us consider a database storing smartphone characteristics. Figure 6.16(e-g) shows validation results of the considered FDs (Figure



(a) A form showing validation statistics after the FD validation process.



(b) A form showing validation statistics after the UCC validation process.

Figure 6.18: Validation statistics provided by INDITIO.

6.16(d)) according to the new tuples the user is planning to insert (see Figure 6.16(a)). In particular, three out of six FDs are valid, two are invalid, and one cannot be validated. In fact, K,M $\rightarrow$ D includes the attribute M that does not appear in the considered database. Instead, C $\rightarrow$ E, i.e., ram $\rightarrow$ display_ppi, is invalidated if the new tuples are inserted.

In general, the impact of the new tuples on the considered metadata is summarized by INDITIO in a new form, named *report form*, shown in Figure 6.18. This form graphically shows the percentage of validation/invalidation produced on the selected metadata by the tuples that the user is planning to insert. Moreover, the report form ranks database attributes in descending order according to the number of invalidated metadata containing them. More specifically, the form represents this kind of information according to the type of metadata, i.e., by splitting the information about invalidation on LHS and RHS when considering FDs (see Figure 6.18a).

Example 2. Figure 6.18a shows the FD validation report for the validation results represented in Figure 6.16. In particular, the form shows that the impact of invalidations is 33% of the analyzed metadata. Moreover, among the attributes involved in the invalidated metadata, attribute C (e.g., ram) is involved in two invalidated FDs; whereas attributes E (e.g., cpu) and D (e.g., display_ppi) are involved in one invalidated FD. This could suggest to verify the values of attributes ram, cpu, and display_ppi on the new tuples.

6.5.2.1 *Interacting with data insertions*

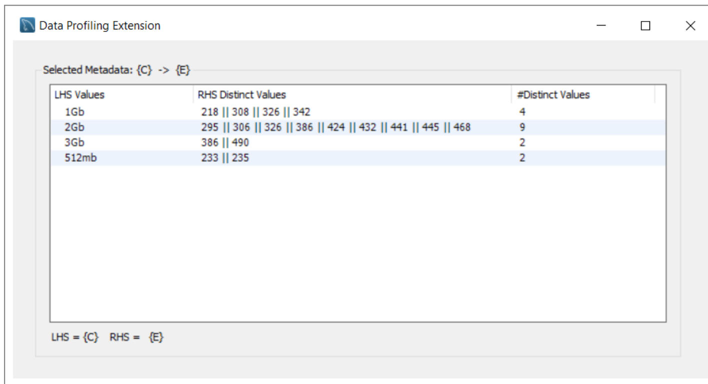
INDITIO not only enables users to visualize the impact of new tuples on holding metadata, but it also permits them to interact with the new data. First of all, INSERT INTO statements can always be modified within the INDITIO interface (see Figure 6.16(a)), triggering subsequent validation processes with modified tuples. As said above, INDITIO freezes the execution of INSERT INTO statements while verifying the possibility to correct values being inserted so as not to invalidate holding metadata. Nevertheless, INDITIO always gives users the possibility to overlook possible

violations of metadata, and to force the execution of data insertion operations by means of the "Execute Statements" button (see Figure 6.16(a)). On the other hand, one of the main goals of INDITIO is to help users in correcting possible errors. To this end, after a validation process (i.e., by clicking "Run Validation"), a user can visualize data yielding violations, by interacting with the "Invalid Metadata" form (see Figure 6.16(f)). More specifically, by clicking on any metadata in such form, INDITIO shows a *violation detail* form, as shown in Figure 6.19. In particular, concerning FDs, apart from the details of the selected metadata, the violation form describes for each LHS value combination involved in a violation: *i*) the value combination of the LHS, *ii*) the corresponding distinct values found on the RHS, and *iii*) their total number of occurrences (see Figure 6.19a). Instead, concerning UCCs, apart from the details of the selected metadata, the violation form describes for each value combination involved in value duplication: *i*) the value combination involved in a duplication, and *ii*) the number of duplications (see Figure 6.19b). The latter should represent the functionality that drives users in accomplishing the best possible correction of errors.

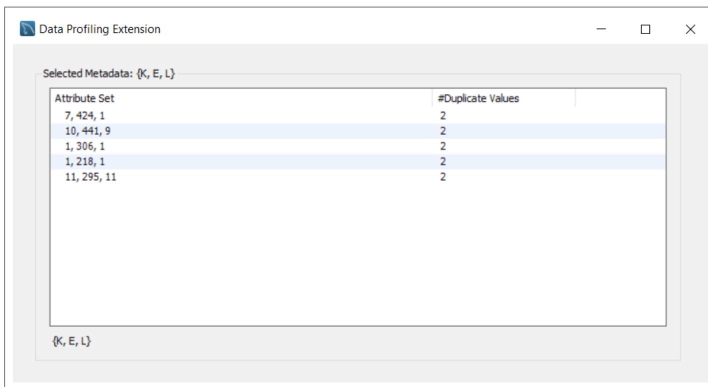
Example 3. Figure 6.19a shows the violation details of the FD $C \rightarrow E$ (e.g. ram $\rightarrow$ display_ppi) according to the validation results represented in Figure 6.16. In particular, the form shows four specific values on attribute C (e.g. ram), i.e. 1Gb, 2Gb, 3Gb, 512mb, each implying different values of attribute E (e.g. display_ppi). Moreover, it is also possible to see that the value 2Gb is the one implying the highest number of distinct values. Instead, Figure 6.19a shows that for the UCC K,E,L (e.g. id_brand, display_ppi, id_os) there are six specific value combinations inducing duplicate values. In general, this form could suggest correcting on the new tuples the values of attributes involved in the considered violated metadata, or to exclude the metadata from the validation process.

6.5.3 *User Study*

The user study presented in this section aims to show that INDITIO makes metadata validation a simple and effective process for improving



(a) A form showing validation statistics after the FD validation process.



(b) A form showing validation statistics after the UCC validation process.

Figure 6.19: Violation details of INDITIO.

data quality. We recruited 86 students majoring in Computer Science who just attended the *Fundamentals of databases* course. We also recruited 3 Ph.D. students and 1 Ph.D., all of which were familiar with the given domain. Statistics about participants have been collected through a background survey, as shown in Table 6.2(Q1-Q4), and whose results are reported in Figure 6.20. In particular, about 85% of recruited people were men, 15% women, and most of them were undergraduate students. Moreover, on average they declared, through a Likert scale, a medium level of knowledge concerning MySQL and MySQL Workbench. Before

the evaluation started, participants underwent a 45-min tutorial on the theoretical foundations of data profiling and INDITIO.

Survey	Alias	Question
Background	Q1	Gender
	Q2	Qualification
	Q3	Level of knowledge of MySQL
	Q4	Level of knowledge of MySQL Workbench
Comparative	Q5	I completed the tasks quickly and easily
	Q6	The instructions for completing the tasks are clear and easy to read
	Q7	The metadata validation process has been simple
	Q8	The values that invalidated the metadata have been easy to find
Final	Q9	The tool is simple to use
	Q10	The tool is simple to learn
	Q11	I was able to retrieve back the process, whenever I made a mistake
	Q12	The tool shows the information very clearly
	Q13	The tool is pleasant to use
	Q14	The user interface is pleasant and informative
	Q15	The tool made transparent the execution of the validation processes
	Q16	It was simple to understand the metadata syntax
	Q17	The tool presents all the features I expected
	Q18	I am in general satisfied about the tool
	Q19	In the future, I would like to use the tool
	Q20	The tool simplified the validation process w.r.t the manual process
	Q21	What is your general impression about the tool?
	Q22	Do you have any improvements to suggest?
	Q23	Which feature of the tool did you like the least?
	Q24	Which feature of the tool did you like the most?

Table 6.2: Questions proposed to participants.

Each of the 90 participants was given a database concerning personal data, three data insertion statements, and two sets of UCCs and FDs metadata, respectively. Moreover, we requested them to check the correctness of data insertion statements according to the provided metadata, and if necessary, to correct statements aiming to guarantee the validity of the provided metadata. More specifically, we conducted a within-subjects

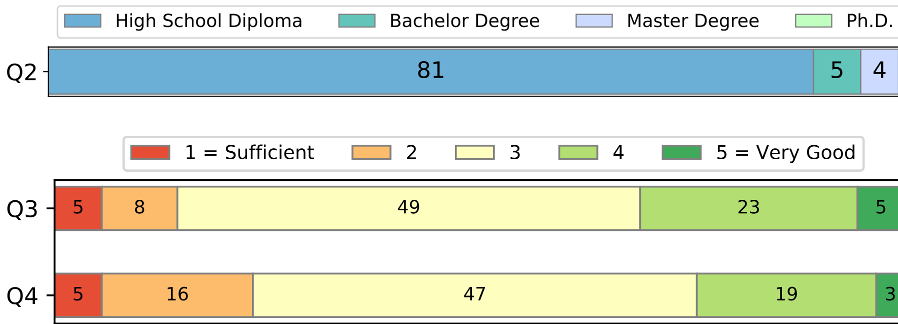


Figure 6.20: Statistics concerning involved participants.

study by considering two scenarios: with and without INDITIO, and requested to accomplish the task in one scenario first, and then with the other one. Half participants considered first the scenario without INDITIO, while the remaining ones used INDITIO first. Notice that, the provided data insertion statements were different but equivalent in complexity. In particular, in the case participants performed the tasks without the tool, they were able to validate the metadata by directly analyzing the data source or by using the SQL language to compose specific queries. To this end, they could interact only with the tools already integrated into MySQL Workbench, both to validate the metadata and to correct the values within the data.

After completing the assigned tasks, participants were requested to fill some questionnaires, aiming to highlight advantages and drawbacks of INDITIO (see Table 6.2 (Q4-Q20)). More specifically, questions from Q4 to Q8 have been filled after participants performed each task (with and without INDITIO, or vice versa), whereas the remaining ones have been included in a final survey. Moreover, questions from Q4 to Q20 are quantitative, and they have been measured through a Likert scale, ranging from 1, mapping "Strongly disagree" response, to 5, mapping "Strongly agree" response. Finally, to further evaluate the effectiveness of both processes (with and without INDITIO) we measured the time required for completing the task and the number of errors.

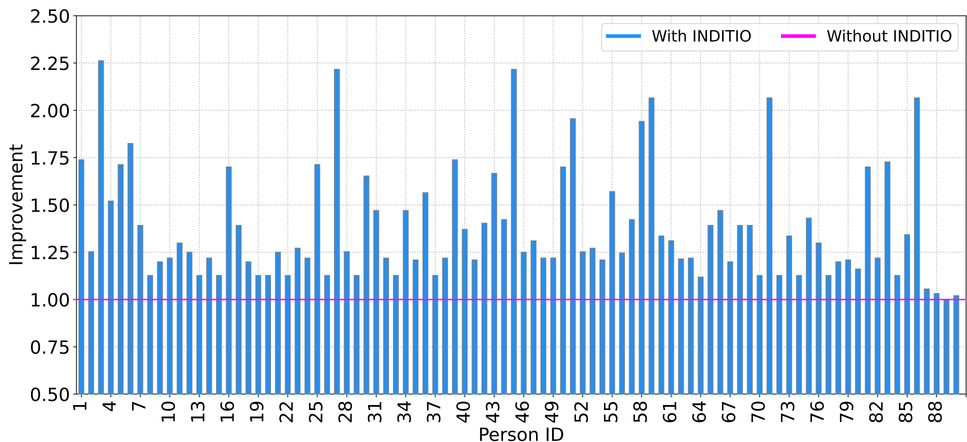


Figure 6.21: Distributing scores achieved by participants for each analyzed scenario (with and without INDITIO).

6.5.3.1 *Results and Discussion*

Figure 6.21 shows the results achieved from each participant while executing the assigned tasks with both the compared scenarios (without and with the tool). The results achieved in the tasks performed without the tool have been considered as the comparative baseline (purple line), while the results obtained with the proposed tool were described by the bars. In particular, the plot highlights the improvement obtained by using INDITIO, e.g., a value of 2 indicates that the results achieved with INDITIO are 2 times better than those achieved without it, whereas a value less than 1 indicates the opposite case. In general, it can be observed that most of the participants performed better with INDITIO, even if satisfactory results have been achieved also without the use of the plugin.

Concerning the time employed to complete the assigned tasks, on average participants took 30 minutes with INDITIO, ranging from 5 to 67 minutes, and 45 minutes without it, ranging from 5 to 84 minutes. In general, we noticed that the times for performing manual validation tasks were particularly long, especially for users with less knowledge of

MySQL. On the contrary, with INDITIO almost all users have reduced the time of the validation processes by more than 50%.

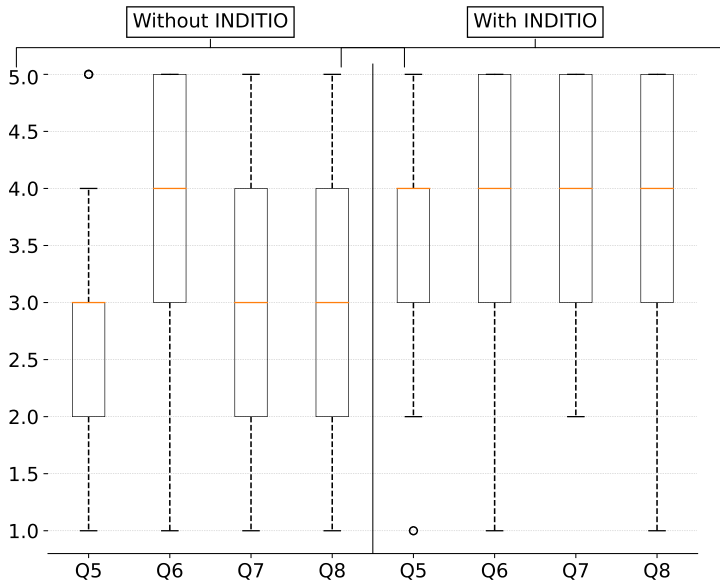


Figure 6.22: Comparative boxplots showing distribution of user answers to the quantitative questionnaire.

Figure 6.22 depicts the box plots derived from the answers (on a Likert scale from 1 to 5) to questions ranging from Q5 to Q8. In particular, users answered the same questions after performing tasks for each considered scenario (with and without the tool). By comparing the results achieved without the plugin (the first four box plots in Figure 6.22) against those with the plugin (the remaining box plots), we can conclude that participants felt more comfortable and effective when working with INDITIO.

Concerning the quantitative questions in the final questionnaire, INDITIO obtained the general agreement of participants while evaluating its usability and effectiveness (see Figure 6.23). In particular, according to answers for Q20, the capability of simplifying the metadata validation process has been widely recognized to INDITIO. The latter has turned

out to be comfortable and useful according to answers to questions Q18 and Q19, simple to learn and pleasant to use accordint to answers to questions Q10 and Q13 (see Figure 6.23). Nevertheless, some work should be made to further improve the general usability of the plugin and the transparency of the validation process, according to answers to questions Q11, Q12, and Q15.

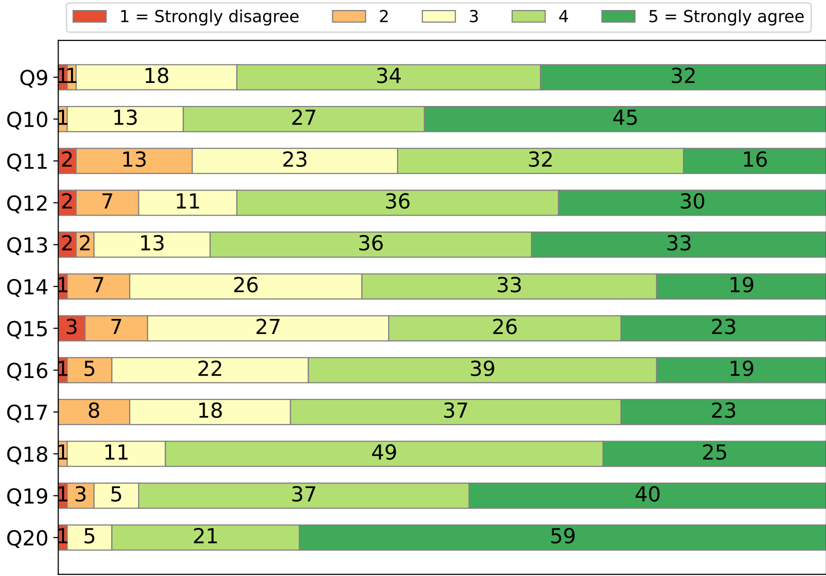


Figure 6.23: Distributing participant answers to the quantitative questions in the final questionnaire.

The open questions in the final questionnaire (Q21-Q24 in Table 6.2) aimed at highlighting the strengths and the weaknesses of INDITIO. In particular, concerning the general impressions about the plugin (see Q21 in Table 6.2), many participants said that INDITIO shows a simple and intuitive interface, in which the components appear well organized in the frame. Another group of them expressed their opinion on the usefulness and efficiency of the plugin, claiming that "*The tool has certainly been successful in its intent, significantly speeding up the time for validating metadata*". Moreover, some of the most interesting comments have been

provided from users that are less familiar with the research context and with the MySQL Workbench. In fact, they report that INDITIO is able to improve the understanding of MySQL Workbench and to show that metadata potentially allows them to extract further knowledge from data, which often is not clearly visible. Only a small group of inexperienced users affirmed that the plugin interface might be initially unclear, claiming that "*Initially the tool seems difficult to understand. Then, once I understand how to use it, it is very useful for checking the validity of FDs and UCCs*". However, by practicing with it, INDITIO allowed them to easily understand its features and to become familiar with its environment.

In addition, concerning the specific features that participants liked the most and the least (see Q22-Q23 in Table 6.2), experienced users have greatly appreciated the functionality of identifying values yielding the invalidation of metadata. In fact, some of them claimed that "*One of the most interesting features is the identification of the values that invalidate the metadata. This functionality could be directly integrated into the MySql Workbench suite.*" Instead, inexperienced users showed interest in the report forms (see Figure 6.19) and in the simplicity through which the plugin could be integrated within the MySQL Workbench suite as a simple plugin. Although most of the comments were positive, we also investigated the features they liked the least. Among them, many users have highlighted that the Query Editor component appears small and does not clearly show statements. However, this is limited by graphics components included in the MySQL Workbench. For these reasons, we allow users to directly interact with the SQL editor of MySQL Workbench, and to import their statements. Other participants suggested adding further reports in the interface, in order to enhance their understanding of how data insertion statements affect metadata. Only a few users have proposed to extend the interface of INDITIO with new graphical components to improve the interaction with both the plugin and the MySQL Workbench.

Finally, we have asked users for some suggestions to enhance INDITIO (see Q24 in Table 6.2). To this end, some users have suggested integrating new metadata, also allowing them to simultaneously validate multiple metadata. Other users have suggested improving the integration with

systems based on the Linux architecture. In fact, it has been found that some of the users using these operating systems tend to view some reports differently from users who use Windows systems. However, this is due to the compatibility problems between the technologies underlying MySQL Workbench and different operating systems. In the future, these compatibility issues might be solved with new software versions.

In summary, the four open questions of the final questionnaire revealed that some participants remarked some limitations of the INDITIO user interface. Moreover, they would like to receive more hints during the statement modification process, according to validation results. Conversely, they positively judged the intuitiveness of the metadata validation and the error detection processes. Moreover, they welcomed the tool and recognized its usefulness.

Part IV

CONCLUSION

7 CONCLUSION AND FUTURE WORK

This section presents the conclusion and the future direction of this thesis.

Thesis Summary. In this thesis we have presented an overview of data profiling tasks and applications, with the aim to highlight the importance of data profiling activities as part of the processes for data quality assessment. In particular, we presented tasks and challenges in the data profiling research area, classifying them and reviewing the state-of-the-art of data profiling systems and techniques. Among such challenges, we first focused on the problem of profiling unstructured data, discussing the necessity to design and develop efficient tools to support companies and researchers in the analysis of unstructured artifacts from the web. Successively, we presented the tool CAIMANS, which extracts metadata from unstructured web data sources, aiming to derive a focused crawler enabling company analysts to make better strategic decisions for improving the productivity and competitiveness of their company. Then, we have focused the discussion on the discovery problem of FDS and RFDS in static and dynamic scenarios, by analyzing its complexities and by introducing several new incremental methodologies and algorithms for discovering FDS and RFDS, aiming to avoid the re-execution of the discovery process from scratch upon update operations on datasets.

The first proposed algorithm, named REDEVO, is a new genetic algorithm for discovering hybrid RFDS, i.e., RFDS relaxing on both the *extent* and *attribute comparison* from data, which is inspired by the process of natural selection belonging to the larger class of evolutionary algorithms. It exploits natural evolution operations of species, such as natural selection, crossover, and mutation, to perform the discovery step and to validate candidate RFDS.

The second proposed algorithm, named INCREMENTAL-FD, is an incremental discovery algorithm of FDS from data. It is a column-based algorithm relying on a lattice representation of the search space to per-

form a level-wise discovery process of FDs. The algorithm considers data as partitions and adopts refinement property to validate each candidate FD. Such a strategy permits to consider previously holding FDs and to efficiently update them upon the insertion of new tuples.

The third proposed algorithm, named REXY, represents an extension of the algorithm INCREMENTAL-FD. It is an incremental discovery algorithm of FDs that introduces a new efficient validation method exploiting regular expressions (RegExs), which permits to validate FDs by focusing only on the subset of data affected by updates. Moreover, REXY adopts a compressed data representation that allows to limit the memory load and optimize the discovery process.

The fourth proposed algorithm, named COD3, is an incremental discovery algorithm optimized to continuously discover FDs from data streams. It adopts a new data structure that permits to efficiently validate FDs and to store data using a lightweight representation. Moreover, COD3 relies on a non-blocking architectural model, which enables continuous processing of data whenever they are read from the stream. To the best of our knowledge, COD3 represents the first algorithm enabling the FD discovery from data streams.

The last proposed algorithm, named BIRD, is an incremental discovery algorithm for RFDs relaxing on the extent (RFD_es). It performs a thorough analysis of the initial candidate RFD_es for incrementally updating the set of holding RFD_es whenever new tuples are inserted into the data. Moreover, BIRD exploits effective data structures and an efficient execution strategy, which permit to split the discovery process into a level-wise parallel execution.

For each algorithm, we have performed several experimental sessions, which have shown that the proposed algorithms are able to efficiently discover FDs and RFDs in both static and dynamic scenarios, achieving good performances with respect to problem complexities. Each algorithm has also been compared with some of the most efficient discovery algorithms by demonstrating the effectiveness of the proposed algorithms. Moreover, for evaluating COD3 performances, we conducted experiments focusing on the discovery of FDs from a real-world data stream.

In the last part of this thesis, we have presented three new tools for monitoring results of incremental discovery algorithms. These tools have been designed to involve users in the analysis of metadata and in the evaluation of discovery processes.

The first proposed tool, named DEVICE, has been designed for monitoring FDs and RFDs extracted during the execution of discovery algorithms through a lattice representation of the search space. It enables users to perform filtering and zooming operations over a graphical lattice representation of the search space, in order to analyze the evolution of discovery results during algorithm executions.

The second proposed tool, named STRADYVAR, visualizes FDs and RFDs discovered from data streams. It enables users to monitor discovery results and their evolution over time, to compare FDs and RFDs discovered across several execution sessions, and to dynamically analyze results by means of visual manipulation operators.

The third proposed tool, named INDITIO, is a MySQL Workbench plugin capable to intercept queries and validate metadata before the execution of insertion operations. It permits to verify in real-time whether the data to be inserted into a database produce some violations on specific metadata, such as unique column combinations (UCCs) and/or functional dependencies (FDs).

For each tool, we have conducted several experimental sessions, by also involving users with different levels of knowledge on the given domain. Experimental results have demonstrated the effectiveness and usefulness of these tools.

Perspectives. Data profiling includes many activities and tasks for analyzing data and extracting insights from them. However, the continuous diffusion of new data sources, leading to the consequent growth of information, requires that existing algorithms and methodologies be extended to support new application domains. For these reasons, starting from the methodologies presented in this thesis, in the future we would like to define new algorithms capable of incremental discovering different types of metadata from data streams, such as unique column combinations and relaxed functional dependencies relaxing on both extent and attribute comparison. Moreover, we plan to define new frameworks for executing

discovery algorithms on non-blocking distributed architectures, aiming to guarantee high performances, also when they analyze large datasets.

Concerning the profiling of unstructured data in the e-procurement domain, we would like to extend the semantic search engine CAIMANS adopting metadata, such as functional dependencies, on web data sources. In this way, it could be possible to improve the matching between the search criteria defined by the users and the artifacts extracted from the web. Consequently, also the discovery algorithms should be extended in order to detect correlations between raw texts within unstructured data.

Another direction that we would like to investigate concerns the involvement of users in the processes of discovery and analysis of metadata. In fact, users could support discovery algorithms by sharing useful information on data, enabling the definition of focused discovery strategies targeted on them. Furthermore, the combination of users and domain experts could lead to the definition of new methodologies for evaluating and ranking metadata, which is still one of the biggest challenges for several data profiling tasks.

Moreover, we would like to further investigate the applicability of data profiling metadata to evaluate the feasibility of machine learning and deep learning models, and to improve their performances with the definition of new advanced techniques for preprocessing data.

Finally, we would like to define new visual tools capable of integrating data profiling algorithms in existing systems that manage relational and non-relational databases, such as Improvado, Postgres, Redis, and so on. Moreover, we aim to extend existing tools by introducing new modules capable of visualizing and analyzing different types of metadata at the same time.

BIBLIOGRAPHY

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. "Profiling relational data: a survey." In: *The VLDB Journal* 24.4 (2015), pp. 557–581.

[2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. "Data profiling: A tutorial." In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1747–1751.

[3] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. "DFD: Efficient Functional Dependency Discovery." In: *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*. CIKM '14. 2014, pp. 949–958.

[4] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. "In-database learning with sparse tensors." In: *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. PODS '18. 2018, pp. 325–340.

[5] Sandigdha Acharya and S Parija. "The process of information extraction through natural language processing." In: *International Journal of Logic and Computation (IJLP)* 1.1 (2010), pp. 40–51.

[6] Lada A Adamic and Eytan Adar. "Friends and neighbors on the web." In: *Social networks* 25.3 (2003), pp. 211–230.

[7] Divyakant Agrawal, Philip Bernstein, Elisa Bertino, Susan Davidson, Umeshwar Dayal, Michael Franklin, Johannes Gehrke, Laura Haas, Alon Halevy, Jiawei Han, et al. "Challenges and opportunities with big data." In: *A community white paper developed by leading researchers across the United States* 5 (2012), pp. 34–43.

[8] Boanerges Aleman-Meza, Farshad Hakimpour, I Budak Arpinar, and Amit P Sheth. "Swetodblp ontology of computer science publications." In: *Journal of Web Semantics* 5.3 (2007), pp. 151–155.

[9] Morteza Alipourlangouri, Adam Mansfield, Fei Chiang, and Yinghui Wu. "Temporal Graph Functional Dependencies–Technical Report." In: *arXiv preprint arXiv:2108.08719* (2021).

[10] Hasitha Indika Arumawadu, RM Rathnayaka, and SK Illangarathne. *K-Means Clustering For Segment Web Search Results*. Kambohwell Publisher Enterprises, 2015.

[11] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. "Models and issues in data stream systems." In: *Proceedings of the 21st ACM Symposium on Principles of Database Systems*. PODS '02. ACM. 2002, pp. 1–16.

[12] K Bache and M Lichman. *UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA (2013)*. 2017.

[13] Jana Bauckmann, Ulf Leser, Felix Naumann, and Véronique Tietz. "Efficiently detecting inclusion dependencies." In: *2007 IEEE 23rd International Conference on Data Engineering*. IEEE. 2007, pp. 1448–1450.

[14] Siegfried Bell. "Discovery and Maintenance of Functional Dependencies by Independencies." In: *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD '95)*. 1995, pp. 27–32.

[15] Martin Berglund, Brink van der Merwe, and Steyn van Litsenborgh. "Regular Expressions with Lookahead." In: *JUCS-Journal of Universal Computer Science* 27 (2021), p. 324.

[16] Ida Bifulco and Stefano Cirillo. "Discovery Multiple Data Structures in Big Data through Global Optimization and Clustering Methods." In: *Proceedinfs of the 22nd International Conference Information Visualisation (IV)*. 2018, pp. 117–121.

[17] Ida Bifulco, Stefano Cirillo, Christian Esposito, Roberta Guadagni, and Giuseppe Polese. "An intelligent system for focused crawling from Big Data sources." In: *Expert Systems with Applications* 184 (2021), p. 115560.

[18] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked data: The story so far." In: *Semantic services, interoperability and web applications: emerging concepts*. IGI global, 2011, pp. 205–227.

[19] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. "Conditional Functional Dependencies for Data Cleaning." In: *Proceedings of the 25th International Conference on Data Engineering*. ICDE '07. 2007, pp. 746–755.

[20] Ingwer Borg and Patrick Groenen. "Modern multidimensional scaling: Theory and applications." In: *Journal of Educational Measurement* 40.3 (2003), pp. 277–280.

[21] Bernardo Breve, Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. "Visualizing Dependencies during Incremental Discovery Processes." In: *EDBT/ICDT Workshops*. 2020.

[22] Bernardo Breve, Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. "Dependency Visualization in Data Stream Profiling." In: *Big Data Research* 25 (2021), p. 100240.

[23] Alan Bundy and Lincoln Wallen. "Breadth-first search." In: *Catalogue of artificial intelligence tools*. 1984, pp. 13–13.

[24] Claudio Carpineto and Giovanni Romano. "A survey of automatic query expansion in information retrieval." In: *ACM Computing Surveys* 44.1 (2012), pp. 1–50.

[25] Loredana Caruccio and Stefano Cirillo. "Incremental discovery of imprecise functional dependencies." In: *Journal of Data and Information Quality (JDIQ)* 12.4 (2020), pp. 1–25.

[26] Loredana Caruccio and Stefano Cirillo. "Monitoring Evolution of Dependency Discovery Results." In: *Journal of Visual Language and Computing (JVLC)* 2 (2020).

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

[28] Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. "Efficient Discovery of Functional Dependencies from Incremental Databases." In: *23rd International Conference on Information Integration and Web Intelligence (iiWAS2021)*. Association for Computing Machinery (ACM). 2021.

[29] Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. "Efficient Validation of Functional Dependencies during Incremental Discovery." In: *Proceedings of the 29th Italian Symposium on Advanced Database Systems*. 2021.

[30] Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. "Real-time visualization of profiling metadata upon data insertions." In: *EDBT/ICDT Workshops*. 2021.

[31] Loredana Caruccio, Vincenzo Deufemia, Felix Naumann, and Giuseppe Polese. "Discovering relaxed functional dependencies based on multi-attribute dominance." In: *IEEE Transactions on Knowledge and Data Engineering* 33.09 (2021), pp. 3212–3228.

[32] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. "Relaxed Functional Dependencies – A Survey of Approaches." In: *IEEE Transactions on Knowledge and Data Engineering* 28.1 (2016), pp. 147–165.

[33] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe 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)*. ACM. 2017, p. 5.

[34] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. "Learning Effective Query Management Strategies from Big Data." In: *16th IEEE International Conference on Machine Learning and Applications*. 2017, pp. 643–648.

[35] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. "Learning effective query management strategies from big data." In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2017, pp. 643–648.

[36] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. "Visualization of (multimedia) dependencies from big data." In: *Multimedia Tools and Applications* 78.23 (2019), pp. 33151–33167.

[37] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. "Mining relaxed functional dependencies from data." In: *Data Mining and Knowledge Discovery* 34.2 (2020), pp. 443–477.

[38] Chuck Cavaness. *Quartz Job Scheduling Framework: Building Open Source Enterprise Applications*. Pearson Education, 2006.

[39] Wei Chen, Cong Xie, Pingping Shang, and Qunsheng Peng. "Visual analysis of user-driven association rule mining." In: *Journal of Visual Language and Computing* 42 (2017), pp. 76–85.

[40] Chih-Wen Cheng, Ying Sha, and May D Wang. "Intervisar: An interactive visualization for association rule search." In: *Proc. of ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. BCB '16. 2016, pp. 175–184.

[41] David W Cheung, Jiawei Han, Vincent T Ng, and CY Wong. "Maintenance of discovered association rules in large databases: An incremental updating technique." In: *Proceedings of the twelfth international conference on data engineering*. IEEE. 1996, pp. 106–114.

[42] Fei Chiang and Renée J. Miller. "Discovering data quality rules." In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 1166–1177.

[43] David Choy, A Brown, E Gur-Esh, R McVeigh, and F Muller. *"Content management interoperability services (CMIS), version 1.0"*. OASIS Standard. Accessed: 2021-01-07. 2010.

[44] X. Chu, I. F. Ilyas, and P. Papotti. "Holistic data cleaning: Putting violations into context." In: *Proceedings of IEEE 29th International Conference on Data Engineering (ICDE '13)*. 2013, pp. 458–469.

[45] Xu Chu, Ihab F Ilyas, and Paolo Papotti. "Holistic data cleaning: Putting violations into context." In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE. 2013, pp. 458–469.

[46] EF Codd. "A Relational Model of Data for Large Shared Data Banks. Comm. of ACM, vol. 13, N 6." In: (1970).

[47] EF Codd. “Further Normalization of the Data Base relational Model. IBM Research.” In: *Journal RJ909* 15857 (1971).

[48] Carlo Combi, Matteo Mantovani, Alberto Sabaini, Pietro Sala, Francesco Amaddeo, Ugo Moretti, and Giuseppe Pozzi. “Mining approximate temporal functional dependencies with pure temporal grouping in clinical databases.” In: *Computers in biology and medicine* 62 (2015), pp. 306–324.

[49] Carlo Combi, Matteo Mantovani, and Pietro Sala. “Discovering quantitative temporal functional dependencies on clinical data.” In: *2017 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE. 2017, pp. 248–257.

[50] Richard Cyganiak, Holger Stenzhorn, Renaud Delbru, Stefan Decker, and Giovanni Tummarello. “Semantic sitemaps: Efficient and flexible access to datasets on the semantic web.” In: *European Semantic Web Conference*. Springer. 2008, pp. 690–704.

[51] Wei Dai, Isaac Wardlaw, Yu Cui, Kashif Mehdi, Yanyan Li, and Jun Long. “Data profiling technology of data governance regarding big data: review and rethinking.” In: *Information Technology: New Generations* (2016), pp. 439–450.

[52] Michael Daum, Frank Lauterwald, Martin Fischer, Mario Kiefer, and Klaus Meyer-Wegener. “Integration of heterogeneous sensor nodes by data stream management.” In: *Wireless Sensor Network Technologies for the Information Explosion Era*. Springer, 2010, pp. 139–172.

[53] Fabien De Marchi, Stéphane Lopes, and Jean-Marc Petit. “Efficient algorithms for mining inclusion dependencies.” In: *International Conference on Extending Database Technology*. Springer. 2002, pp. 464–476.

[54] MC Ferreira De Oliveira and Haim Levkowitz. “From visual data exploration to visual data mining: a survey.” In: *IEEE Transactions on Visualization and Computer Graphics* 9.3 (2003), pp. 378–394.

[55] César Roberto De Souza. "A tutorial on principal component analysis with the accord. net framework." In: *arXiv preprint arXiv:1210.7463* (2012).

[56] Pedro G DeLima and Gary G Yen. "Multiple objective evolutionary algorithm for temporal linguistic rule extraction." In: *ISA transactions* 44.2 (2005), pp. 315–327.

[57] Yajun Du, Wenjun Liu, Xianjing Lv, and Guoli Peng. "An improved focused crawler based on semantic similarity vector space model." In: *Applied Soft Computing* 36 (2015), pp. 392–407.

[58] Jérôme Euzenat, Pavel Shvaiko, et al. *Ontology matching*. Vol. 18. Springer, 2007.

[59] Seyed Mostafa Fakhrahmad, MH Sadreddini, and M Zolghadri Jahromi. "AD-Miner: A new incremental method for discovery of minimal approximate dependencies using logical operations." In: *Intelligent Data Analysis* 12.6 (2008), pp. 607–619.

[60] Wenfei Fan, Philip Bohannon, Floris Geerts, Xibei Jia, and Anastasios Kementsiets. "Conditional functional dependencies for data cleaning." In: *Data Engineering, 2007, IEEE 23rd International Conference on*. 746-755. IEEE. 2007.

[61] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. "Dynamic constraints for record matching." In: *The VLDB Journal* 20.4 (2011), pp. 495–520.

[62] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. "Dynamic constraints for record matching." In: *The VLDB Journal* 20 (2011), pp. 495–520.

[63] Wenfei Fan, Floris Geerts, Laks V. S. Lakshmanan, and Ming Xiong. "Discovering Conditional Functional Dependencies." In: *Proceedings of the 25th International Conference on Data Engineering, ICDE'09*. 2009, pp. 1231–1234.

[64] Peter A. Flach and Iztok Savnik. "Database Dependency Discovery: A Machine Learning Approach." In: *AI Communications* 12.3 (1999), pp. 139–160. ISSN: 0921-7126.

[65] Nicolas Foucault, Gilles Adda, and Sophie Rosset. "Language modeling for document selection in question answering." In: *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*. 2011, pp. 716–720.

[66] Sandra Geisler. "Data stream management systems." In: *Dagstuhl Follow-Ups*. Vol. 5. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2013.

[67] Thanaa M Ghanem, Moustafa A Hammad, Mohamed F Mokbel, Walid G Aref, and Ahmed K Elmagarmid. "Incremental evaluation of sliding-window queries over data streams." In: *IEEE Transactions on Knowledge and Data Engineering* 19.1 (2006), pp. 57–72.

[68] Luca M. Ghiringhelli, Jan Vybiral, Sergey V. Levchenko, Claudia Draxl, and Matthias Scheffler. "Big data of materials science: Critical role of the descriptor." In: *Phys. Rev. Lett.* 114.4105503 (2015).

[69] M Rami Ghorab, Dong Zhou, Alexander O'connor, and Vincent Wade. "Personalised information retrieval: survey and classification." In: *User Modeling and User-Adapted Interaction* 23.4 (2013), pp. 381–443.

[70] Chris Giannella and Edward Robertson. "On approximation measures for functional dependencies." In: *Inform. Syst.* 29.6 (2004), pp. 483–507.

[71] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. "On generating near-optimal tableaux for conditional functional dependencies." In: *PVLDB* 1.1 (2008), pp. 376–390.

[72] Lukasz Golab and M Tamer Özsu. "Issues in data stream management." In: *ACM Sigmod Record* 32.2 (2003), pp. 5–14.

[73] Robert Gove. *Using the elbow method to determine the optimal number of clusters for k-means clustering*. Accessed: 2021-01-07. 2015.

[74] Jonatas S Grosman, Pedro HT Furtado, Ariane MB Rodrigues, Guilherme G Schardong, Simone DJ Barbosa, and Helio CV Lopes. "Eras: Improving the quality control in the annotation process for Natural Language Processing tasks." In: *Information Systems* (2020), p. 101553.

[75] Michael Hahsler. "arulesViz: Interactive Visualization of Association Rules with R." In: *The R Journal* 9.2 (2017), p. 163.

[76] Jean-Luc Hainaut, Jean Henrard, Didier Roland, Jean-Marc Hick, and Vincent Englebert. "Database reverse engineering." In: *Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends*. IGI Global, 2009, pp. 181–189.

[77] Sven Hartmann and Sebastian Link. "More functional dependencies for XML." In: *East European Conference on Advances in Databases and Information Systems*. Springer. 2003, pp. 355–369.

[78] Oktie Hassanzadeh and Mariano P Consens. "Linked movie data base." In: *LDOW*. 2009.

[79] Michael Hausenblas, Wolfgang Halb, Yves Raimond, Lee Feigenbaum, and Danny Ayers. "Scovo: Using statistics on the web of data." In: *European Semantic Web Conference*. Springer. 2009, pp. 708–722.

[80] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. "Scalable discovery of unique column combinations." In: *Proceedings of the VLDB Endowment* 7.4 (2013), pp. 301–312.

[81] Allan Heydon and Marc Najork. "Mercator: A scalable, extensible web crawler." In: *World Wide Web* 2.4 (1999), pp. 219–229.

[82] Guobiao Hu, Shuigeng Zhou, Jihong Guan, and Xiaohua Hu. "Towards effective document clustering: A constrained K-means based approach." In: *Information Processing & Management* 44.4 (2008), pp. 1397–1409.

[83] Anna Huang. "Similarity measures for text document clustering." In: *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC)*. Vol. 4. 2008, pp. 9–56.

[84] Ykä Huhtala, Juha Karkkainen, Pasi Porkka, and Hannu Toivonen. "Efficient discovery of functional and approximate dependencies using partitions." In: *Proceedings 14th International Conference on Data Engineering*. IEEE. 1998, pp. 392–401.

[85] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. "TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies." In: *The Computer Journal* 42.2 (1999), pp. 100–111.

[86] Michael Jones, Brain Campbell, and Chuck Mortimore. *JSON Web Token (JWT), profile for OAuth 2.0 client authentication and authorization Grants*. Accessed: 2021-01-07. 2015.

[87] Daniel A Keim, Jorn Schneidewind, and Mike Sips. "CircleView: A new approach for visualizing time-related multidimensional data sets." In: *Proc. of Working Conference on Advanced Visual Interfaces*. AVI '04. 2004, pp. 179–182.

[88] Hyunjoong Kim, Han Kyul Kim, and Sungzoon Cho. "Improving spherical k-means for document clustering: Fast initialization, sparse centroid projection, and efficient cluster labeling." In: *Expert Systems with Applications* 150 (2020), p. 113288.

[89] R.S. King and J. Oil. "Discovery of Functional and Approximate Functional Dependencies in Relational Databases." In: *J. Applied Math. and Decision Sciences* 7.1 (2003), pp. 49–59.

[90] Jyrki Kivinen and Heikki Mannila. "Approximate inference of functional dependencies from relations." In: *Theor. Comput. Sci.* 149.1 (1995), pp. 129–149.

[91] Anja Klein and Wolfgang Lehner. "Representing data quality in sensor data streaming environments." In: *Journal of Data and Information Quality (JDIQ)* 1.2 (2009), pp. 1–28.

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

[93] Sebastian Kruse and Felix Naumann. "Efficient discovery of approximate dependencies." In: *Proceedings of the VLDB Endowment* 11.7 (2018), pp. 759–772.

[94] Sebastian Kruse, Thorsten Papenbrock, Christian Dullweber, Moritz Finke, Manuel Hegner, Martin Zabel, Christian Zöllner, and Felix Naumann. "Fast approximate discovery of inclusion dependencies." In: *Datenbanksysteme für Business, Technologie und Web (BTW 2017)* (2017).

[95] Manish Kumar, Rajesh Bhatia, and Dhavleesh Rattan. "A survey of Web crawlers for information retrieval." In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7.6 (2017), e1218.

[96] Selasi Kwashie, Jixue Liu, Jiuyong Li, and Feiyue Ye. "Mining Differential Dependencies: A Subspace Clustering Approach." In: *Proceedings of Australasian Database Conference*. ADC '14. 2014, pp. 50–61.

[97] Kenneth Lai and Narciso Cerpa. "Support vs. confidence in association rule algorithms." In: *Proceedings of the OPTIMA Conference, Curico*. 2001, pp. 1–14.

[98] R Lakshmi and S Baskar. "Novel term weighting schemes for document representation based on ranking of terms and fuzzy logic with semantic relationship of terms." In: *Expert Systems with Applications* 137 (2019), pp. 493–503.

[99] Kwok-Wa Lam and Victor CS Lee. "Building decision trees using functional dependencies." In: *Proceedings of International Conference on Information Technology: Coding and Computing*. Vol. 2. ITCC 2004. IEEE. 2004, pp. 470–473.

[100] Rohulla Kosari Langari, Soheila Sardar, Seyed Abdollah Amin Mousavi, and Reza Radfar. "Combined fuzzy clustering and firefly algorithm for privacy preserving in social networks." In: *Expert Systems with Applications* 141 (2020), p. 112968.

[101] Andreas Langegger and Wolfram Woss. "RDFStats-an extensible RDF statistics generator and library." In: *2009 20th International Workshop on Database and Expert Systems Application*. IEEE. 2009, pp. 79–83.

[102] Dominique Laurent and Nicolas Spyratos. "Rewriting aggregate queries using functional dependencies." In: *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*. 2011, pp. 40–47.

[103] Marie Le Guilly, Jean-Marc Petit, and Vasile-Marian Scuturici. "Evaluating Classification Feasibility Using Functional Dependencies." In: *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLIV*. Springer, 2020, pp. 132–159.

[104] Andrew Lensen, Bing Xue, and Mengjie Zhang. "Using particle swarm optimisation and the silhouette metric to estimate the number of clusters, select features, and perform clustering." In: *Proceedings of the European Conference on the Applications of Evolutionary Computation*. 2017, pp. 538–554.

[105] Huiying Li. "Data profiling for semantic web data." In: *International Conference on Web Information Systems and Mining*. Springer. 2012, pp. 472–479.

[106] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. "CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks." In: *Proceedings of 37th IEEE International Conference on Data Engineering, to Appear*. ICDE 2021.

[107] Chien-Liang Liu, Wen-Hoar Hsaio, Chia-Hoang Lee, and Chun-Hsien Chen. "Clustering tagged documents with labeled and unlabeled documents." In: *Information Processing & Management* 49.3 (2013), pp. 596–606.

[108] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. "Discover Dependencies from Data - A Review." In: *IEEE Transactions on Knowledge and Data Engineering* 24.2 (2012), pp. 251–264.

[109] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. "Efficient Discovery of Functional Dependencies and Armstrong Relations." In: *Proceedings of the 7th International Conference on Extending Database Technology*. EDBT '00. 2000, pp. 350–364.

[110] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. "Generic schema matching with cupid." In: *vldb*. Vol. 1. Citeseer. 2001, pp. 49–58.

[111] Heikki Mannila and Kari-Jouko Raiha. "Dependency inference." In: *Proceedings of the 13th International Conference on Very Large Data Bases*. VLDB '87. 1987, pp. 155–158.

[112] Adam Marcus, Michael S Bernstein, Osama Badar, David R Karger, Samuel Madden, and Robert C Miller. "Processing and visualizing the data in tweets." In: *ACM SIGMOD Record* 40.4 (2012), pp. 21–27.

[113] Arkady Maydanchik. *Data quality assessment*. Technics publications, 2007.

[114] Suyash Mishra and Anuranjan Misra. "Structured and Unstructured Big Data Analytics." In: *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*. IEEE. 2017, pp. 740–746.

[115] Mark Lukas Möller, Nicolas Berton, Meike Klettke, Stefanie Scherzinger, and Uta Störl. "jhound: Large-scale profiling of open JSON data." In: *BTW 2019* (2019).

[116] Ullas Nambiar and Subbarao Kambhampati. "Mining approximate functional dependencies and concept similarities to answer imprecise queries." In: *Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004*. 2004, pp. 73–78.

[117] B Nath, DK Bhattacharyya, and A Ghosh. “Incremental association rule mining: a survey.” In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 3.3 (2013), pp. 157–169.

[118] Felix Naumann. “Data profiling revisited.” In: *ACM SIGMOD Record* 42.4 (2014), pp. 40–49.

[119] Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. “Using of Jaccard coefficient for keywords similarity.” In: *Proceedings of the international multiconference of engineers and computer scientists*. Vol. 1. 6. 2013, pp. 380–384.

[120] Levin Noronha and Fei Chiang. “Discovery of Temporal Graph Functional Dependencies.” In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 3348–3352.

[121] Noel Novelli and Rosine Cicchetti. “FUN: An Efficient Algorithm for Mining Functional and Embedded Dependencies.” In: *Proceedings of 8th International Conference Database Theory*. ICDT ’01. 2001, pp. 189–203.

[122] Hian-Huat Ong, Kok-Leong Ong, Wee-Keong Ng, and Ee Peng Lim. “CrystalClear: Active visualization of association rules.” In: *Proc. Workshop on Active Mining*. 2002, pp. 1–6.

[123] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Vol. 2. Springer, 1999.

[124] Tero Paivarinta and Bjørn Erik Munkvold. “Enterprise content management: an integrated perspective on information management.” In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (2005), pp. 96–96.

[125] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. “Data profiling with Metanome.” In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1860–1863.

[126] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. "Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms." In: *Proceedings of the VLDB Endowment* 8.10 (2015), pp. 1082–1093.

[127] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. "Divide & conquer-based inclusion dependency discovery." In: *Proceedings of the VLDB Endowment* 8.7 (2015), pp. 774–785.

[128] Thorsten Papenbrock and Felix Naumann. "A hybrid approach to functional dependency discovery." In: *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD '16)*. ACM. 2016, pp. 821–833.

[129] Kostas Patroumpas and Timos Sellis. "Window specification over data streams." In: *International Conference on Extending Database Technology*. Springer. 2006, pp. 445–464.

[130] Eduardo Pena, Eduardo HM Pena, Erik Falk, Jorge Augusto Meira, and Eduardo Cunha de Almeida. "Mind your dependencies for semantic query optimization." In: *Journal of Information and Data Management* 9.1 (2018), pp. 3–3.

[131] Romain Perriot, Laurent d'Orazio, Dominique Laurent, and Nicolas Spyratos. "Rewriting aggregate queries using functional dependencies within the cloud." In: *International Workshop on Information Search, Integration, and Personalization*. Springer. 2013, pp. 31–42.

[132] J-M Petit, Jacques Kouloumdjian, J-F Boulicaut, and Farouk Toumani. "Using queries to improve database reverse engineering." In: *International Conference on Conceptual Modeling*. Springer. 1994, pp. 369–386.

[133] Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. "Improved histograms for selectivity estimation of range predicates." In: *ACM Sigmod Record* 25.2 (1996), pp. 294–305.

[134] Hema Raghavan, James Allan, and Andrew McCallum. "An exploration of entity models, collective classification and relation description." In: *KDD Workshop on Link Analysis and Group Detection*. Citeseer. 2004, pp. 1–10.

[135] Juan Ramos et al. "Using tf-idf to determine word relevance in document queries." In: *Proceedings of the first instructional conference on machine learning*. Vol. 242. 2003, pp. 133–142.

[136] Colin R Reeves. "Genetic algorithms." In: *Handbook of metaheuristics*. Springer, 2010, pp. 109–139.

[137] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. "HoloClean: Holistic Data Repairs with Probabilistic Inference." In: *Proc. VLDB Endow.* 10.11 (Aug. 2017), pp. 1190–1201. ISSN: 2150-8097.

[138] Octavian Rusu, Ionela Halcu, Oana Grigoriu, Giorgian Neculoiu, Virginia Sandulescu, Mariana Marinescu, and Viorel Marinescu. "Converting unstructured and semi-structured data into knowledge." In: *2013 11th RoEduNet International Conference*. IEEE. 2013, pp. 1–4.

[139] Daniel Sánchez, Jose Maria Serrano, Ignacio Blanco, Maria Jose Martin-Bautista, and Maria-Amparo Vila. "Using association rules to mine for strong approximate dependencies." In: *Data Mining and Knowledge Discovery* 16.3 (2008), pp. 313–348.

[140] Philipp Schirmer, Thorsten Papenbrock, Sebastian Kruse, Dennis Hempfing, Torben Meyer, Daniel Neuschafer-Rube, and Felix 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.

[141] Vinicius Segura and Simone DJ Barbosa. "Historyviewer: Instrumenting a visual analytics application to support revisiting a session of interactive data analysis." In: *Proc. of the ACM on Human-Computer Interaction* (2017), p. 11.

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

[143] Nuhad Shaabani and Christoph Meinel. "Scalable inclusion dependency discovery." In: *International Conference on Database Systems for Advanced Applications*. Springer. 2015, pp. 425–440.

[144] Antonio M Silva and Michael A Melkanoff. "A method for helping discover the dependencies of a relation." In: *Advances in Data Base Theory*. Springer, 1981, pp. 115–133.

[145] Yannis Sismanis, Paul Brown, Peter J Haas, and Berthold Reinwald. "Gordian: efficient and scalable discovery of composite keys." In: *Proceedings of the 32nd international conference on Very large data bases*. 2006, pp. 691–702.

[146] Jieun Son and Seoung Bum Kim. "Content-based filtering for recommendation systems using multiattribute networks." In: *Expert Systems with Applications* 89 (2017), pp. 404–412.

[147] Shaoxu Song and Lei Chen. "Differential dependencies: Reasoning and discovery." In: *ACM Transactions on Database Systems* 36 (3 2011), p. 16.

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

[149] Shaoxu Song, Fei Gao, Ruihong Huang, and Chaokun Wang. "Data Dependencies over Big Data: A Family Tree." In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

[150] Shaoxu Song, Aoqian Zhang, Lei Chen, and Jianmin Wang. "Enriching data imputation with extensive similarity neighbors." In: *Proceedings of the VLDB Endowment* 8.11 (2015), pp. 1286–1297.

[151] Shaoxu Song, Aoqian Zhang, Jianmin Wang, and Philip S Yu. "Screen: Stream data cleaning under speed constraints." In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 2015, pp. 827–841.

[152] Hyontai Sug. "Making Use of Functional Dependencies Based on Data to Find Better Classification Trees." In: *International Journal of Circuits, Systems and Signal Processing* (2021).

[153] Zhen Sun, Ee-Peng Lim, Kuiyu Chang, Teng-Kwee Ong, and Rohan Kumar Gunaratna. "Event-driven document selection for terrorism information extraction." In: *Proceedings of the International conference on intelligence and security informatics*. 2005, pp. 37–48.

[154] A Szczurek, M Maciejewska, et al. "Sensor array data profiling for gas identification." In: *Talanta* 78.3 (2009), pp. 840–845.

[155] Salman Taherian, Marcelo Pias, R Harle, George Coulouris, Simon Hay, Jonathan Cameron, Joan Lasenby, Gregor Kuntze, Ian Bezodis, Gareth Irwin, et al. "Profiling sprints using on-body sensors." In: *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE. 2010, pp. 444–449.

[156] Robert Tarjan. "Depth-first search and linear graph algorithms." In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[157] Kesaraporn Techapichetvanich and Amitava Datta. "VisAR: A new technique for visualizing mined association rules." In: *Proc. of International Conference on Advanced Data Mining and Applications*. ADMA '05. Springer. 2005, pp. 88–95.

[158] Fabian Tschirschnitz, Thorsten Papenbrock, and Felix Naumann. "Detecting inclusion dependencies on very many tables." In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), pp. 1–29.

[159] Ozge Uncu and IB Turksen. "Two step feature selection: approximate functional dependency approach using membership values." In: *Proceedings of IEEE International Conference on Fuzzy Systems*. Vol. 3. IEEE. 2004, pp. 1643–1648.

[160] Courtney Cox Wakefield. "Achieving position 0: Optimising your content to rank in Google's answer box." In: *Journal of Brand Strategy* 7.4 (2019), pp. 326–336.

[161] Shyue-Liang Wang, Ju-Wen Shen, and Tzung-Pei Hong. "Incremental discovery of functional dependencies using partitions." In: *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*. Vol. 3. IEEE. 2001, pp. 1322–1326.

[162] Catharine Wyss, Chris Giannella, and Edward Robertson. "FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances." In: *Proceedings of International Conference on Data Warehousing and Knowl. Disc.* DaWaK '01. 2001, pp. 101–110.

[163] Hong Yao and Howard J Hamilton. "Mining functional dependencies from data." In: *Data Mining and Knowledge Discovery* 16.2 (2008), pp. 197–219.

[164] Hong Yao, Howard J. Hamilton, and Cory J. Butz. "FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences." In: *Proceedings of IEEE International Conference on Data Mining*. ICDM '02. 2002, pp. 729–732.

[165] Li Yujian and Liu Bo. "A normalized Levenshtein distance metric." In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1091–1095.

[166] Lin Zhu, Xu Sun, Zijing Tan, Kejia Yang, Weidong Yang, Xiangdong Zhou, and Yingjie Tian. "Incremental Discovery of Order Dependencies on Tuple Insertions." In: *Proceedings of the 24th International Conference on Database Systems for Advanced Applications (DASFAA '19)*. Springer. 2019, pp. 157–174.