UNIVERSITY OF TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE

International Doctorate School
in Information and Communication Technologies

PhD Dissertation

INFORMATION EXTRACTION FROM DATA

Paolo Sottovia

Advisor:
Prof. Yannis Velegrakis
University of Trento

academic year 2018-2019

*

Abstract

Data analysis is the process of inspecting, cleaning, extract, and modeling data with the intention of extracting useful information in order to support users in their decisions. With the advent of Big Data, data analysis was becoming more complicated due to the volume and variety of data. This process begins with the acquisition of the data and the selection of the data that is useful for the desiderata analysis. With such amount of data, also expert users are not able to inspect the data and understand if a dataset is suitable or not for their purposes.

In this dissertation, we focus on five problems in the broad data analysis process to help users find insights from the data when they do not have enough knowledge about its data. First, we analyze the data description problem, where the user is looking for a description of the input dataset. We introduce data descriptions: a compact, readable and insightful formula of boolean predicates that represents a set of data records. Finding the best description for a dataset is computationally expensive and task-specific; we, therefore, introduce a set of metrics and heuristics for generating meaningful descriptions at an interactive performance. Secondly, we look at the problem of order dependency discovery, which discovers another kind of metadata that may help the user in the understanding of characteristics of a dataset. Our approach leverages the observation that discovering order dependencies can be guided by the discovery of a more specific form of dependencies called order compatibility dependencies. Thirdly, textual data encodes much hidden information. To allow this data to reach its full potential, there has been an increasing interest in extracting structural information from it. In this regard, we propose a novel approach for extracting events that are based on temporal co-reference among entities. We consider an event to be a set of entities that collectively experience relationships between them in a specific period of time. We

developed a distributed strategy that is able to scale with the largest on-line encyclopedia available, Wikipedia. Then, we deal with the evolving nature of the data by focusing on the problem of finding synonymous attributes in evolving Wikipedia Infoboxes. Over time, several attributes have been used to indicate the same characteristic of an entity. This provides several issues when we are trying to analyze the content of different time periods. To solve it, we propose a clustering strategy that combines two contrasting distance metrics. We developed an approximate solution that we assess over 13 years of Wikipedia history by proving its flexibility and accuracy. Finally, we tackle the problem of identifying movements of attributes in evolving datasets. In an evolving environment, entities not only change their characteristics, but they sometimes exchange them over time. We proposed a strategy where we are able to discover those cases, and we also test our strategy on real datasets.

We formally present the five problems that we validate both in terms of theoretical results and experimental evaluation, and we demonstrate that the proposed approaches efficiently scale with a large amount of data.

Keywords

[Database explanation, dependency discovery, event extraction]

Acknowledgements

To Whom It May Concern.

Contents

Chapter 1

Introduction

We are living in the Big data era, in a very broad amount of fields data is being collected at an unimaginable scale. Such a revolution had a significant impact on the whole flow of the analysis of the data. Previously such analysis was made and based on the what happened in the real world, now decisions could be taken on the data itself. The big data analysis now control nearly every aspect of our lives, including mobile services, advertisements, social network, financial market, and others.

Obviously, the major challenge is the size of the data. However, the volume is not the only challenge involved in the data analysis; also the variety of data is a crucial challenge [pr11]. Traditionally these challenges were part of the four V's of Big Data: Volume, Velocity, Veracity, and Variety. By Volume, they usually mean the scale of the data, while by Velocity, they mean both the rate at which data arrive and the time in which this data must be processed. By Variety, they usually mean heterogeneity of data types, representation, and semantic interpretation. Finally, by Veracity, they mean its accuracy.

The analysis of Big Data involves multiple distinct phases as shown in Figure 1.1, each of which gives a fundamental contribution to the analysis of the data [LJ12]. The first step is data acquisition, that is the process of ingesting data from the input sources. It is followed by the information extraction and cleaning phase that aims to prepare the needed information for further steps in the right format for the analysis. Given the heterogeneity of the data, it is not sufficient to record it and store it into the data repository because in the majority of case data need to be integrated, aggregated and represented

Figure 1.1: The Big Data Analysis pipeline.

correctly. Finally, different techniques could be applied on the cleaned and integrated datasets in order to extract information from them. Only the ability to analyze the data does not guarantee the user can interpret the results, hence interpretation of the results by the user is a fundamental step.

Many people, unfortunately, focus just on the analysis or modeling phase. On the one hand, this phase represents the most important step of the whole pipeline, on the other hand, this is not effective without the contribution of the other steps.

When users start some analysis, the problems begin directly when they have to acquire their data, since in many cases the considerable amount of data that is ingested required the user to make decisions, typically done by ad hoc strategies, about what data to keep or discard, and to store it with the right metadata.

1.1 Motivation

Consider, for example, a computer researcher that is performing some experiment over some scientific datasets. In many cases, these "bunches of data sets" are dumps of databases, raw data files, or document folders. Most often they do not include structural information, documentation, structural or semantic constraints, or any other kind of metadata. This problem is also related to companies, where datasets are accumulated without any knowledge of their content, structure, and usefulness in repositories called data lakes [DFA+17, AGNP18, LJ12].

This problem mainly affects the data acquisition phase that is one of the fundamental steps in the data analysis. This phase aims at preparing data for the ingestion and analytics and its purpose is to clean, shape, structure, integrate and maintain the data through the entire data analysis process. The importance of this phase is also emphasized by a recent article that reports that the data preparation task takes about 79% of a data scientist's time, and 78% of data scientists declare them to be the least enjoyable part of data science [Pre16].

Recently, since the phenomenon of Big Data is growing year after year a great body of research has been devoted to the study of techniques to examine and generate metadata. Metadata is a very broad concept that may be simple statistics, data types, frequent patterns or important subsets of the input datasets, or its data values. Moreover, metadata could be even dependencies among the different columns or rows of the data. Unfortunately, existing approaches cover only a few characteristics of the datasets and they are limited to an individual complex task and their results are pretty hard to interpret. Moreover, the volume and the variety of the provide additional challenges in the extraction of such relevant information that is of paramount importance in the data analysis phase.

Contributions. In this work we focus on the task of information extraction, mainly focusing on different type of data and metadata, extracting from different types of data, and we address the issue of enabling users to retrieve insights from those datasets. This is a fundamental activity for expert and novice users, that gives them the ability to understand the content or the relationships between rows and columns in their data. As a result, we provide five different contributions that are initially introduced in this chapter, and then extensively presented with data model formalization (definitions, theorems, and proofs), algorithms, and experiments in the rest of the rest of this dissertation.

In particular, to enable experts and novel users to fully exploit the content of a dataset, (1) we provide solutions for an effective explanation system that assist the user in the dataset explanation process by proving a compact and readable data description (Section 1.2 and Chapter 3). We study a novel techniques for extracting order dependencies from datasets; this enhances the users in discovering complex ordering relationships between the datasets columns (Section 1.3 and Chapter 4). Then, we study a new event extraction technique (3) that enables the extraction of historical events from document collections and highlight passages that describe events (Section 1.4 and Chapter 5). Consequently, we study the evolving nature of the data by providing two different contributions. First, we study a new extraction technique (4) that enables the discovery of synonymous attributes in evolving Wikipedia Infoboxes (Section 1.5 and Chapter 6). Finally, we study how we can identify attributes that are moving across entities in an evolving dataset such as Wikipedia (Section 1.6 and Chapter 7).

1.2 Explaining Datasets

Industry and research communities have always had the prerogative to develop easy-to-use and interactive mechanisms for data management and analysis. This problem is nowadays exacerbated to accommodate the many emerging, and not always database-savvy, professionals such as data scientist, data journalists, and data enthusiasts in general, who perform data-related activities. There is, indeed, a large plethora of approaches that facilitate data management such as data debuggers [DG15, IST+15, KBS12], data explanation systems [ROS15, WM13], exploratory search systems [BMQ+15, RWD+08], outlier detection systems [YZH+15] and so on.

This work aims to integrate and complement these approaches by introducing a mechanism for creating *data descriptions* (descriptions in short), a form of explanation that aims at making, from a human perspective, a (large) set of data more understandable at a glance. A description is a compact, readable and insightful structure formed by *predicates* that apply to the target dataset. Predicates are more informative than single tuples and, for this reason, are used extensively within explanation systems [ROS15, WM13, YNM16]. Descriptions are useful in contexts where users need some form of explanation.

While existing data explanation tools and techniques are used to gain knowledge on specific tasks such as unexpected behaviors of systems [YNM16], experiments or query answers [LBL16, ROS15, WM13], data descriptions are generic inasmuch as they are able to concisely represents any arbitrary set of data tuples. While generating a description per se is relatively easy, the problem of generating all descriptions is exponential whereby searching for the "best description" becomes an important and challenging task.

The first challenge is to understand whether a description is relevant to a user: we solve this challenge by introducing a series of metrics for assessing the *qualitative aspects* of descriptions. Interestingly, these aspects are somehow in contrast with each other, e.g., a short description is understandable at the expenses of losing a significant chunk of information. We, therefore, introduce a comprehensive relevance function that balances all metrics and allows users to specify their preferences. The second challenge is related to the large number of descriptions we can choose from: since users tend to use descriptions for exploring a dataset, we have to generate good

descriptions at interactive speed. Clearly, a naive solution that exhaustively evaluates all possible descriptions compromises the usability of the approach. We use a set of heuristic rules to generate a subset of relevant descriptions only, in concert with a mechanism for ranking the top-k descriptions based on statistics and user specifications. The third and final challenge is the objective evaluation of the quality of generated descriptions. In our experiments we assess the approach both in a quantitative and in a qualitative way, taking into account both standard metrics and the user feedbacks. In Chapter 3, we propose an approach that drives the user in the creation of descriptions.

List of publications. The problem of data descriptions, where we provide an insightful and efficient description of a dataset is presented in an article that has been submitted for publication to a journal [PSM+19a]. We also developed a system prototype that shows the performance and effectiveness of our solution, this has appeared in [PSM+19b].

1.3 Order Dependency Discovery

In the big data era, the volume and complexity of available datasets has grown so much that data engineers are having a hard time interpreting the information contained in them. In such a reality, the ability to discover hidden dependencies in some automatic way is fundamental. Dependencies across different parts of the data play a significant role in query optimization, since redundant information may be ignored making the query evaluation faster. Furthermore, parts of the data may be replaced with others that are easier to manipulate, without affecting the final result. Data profiling may help with data quality since it highlights constraints that may exist in the data but are not fully satisfied and have not been enforced when designing the database. Dependency discovery is not a new challenge. Functional and inclusion dependencies are the most common type of dependencies and have extensively been studied [LLLC12]. A *functional dependency* states that if two different data elements sharing a common structure have the same part A, then some other part B should also have the same value. An *inclusion dependency* states that the values of the data elements in some part A must be a subset of the values in a subpart B of some other portion of the dataset.

The knowledge encoded by order dependencies can be applied to various tasks during the entire data life-cycle [ALNZ13]: in the design phase, order dependencies can be exploited to assist schema design [SGG12b] or for selecting indexes [DH82]; if data are extracted from unstructured sources, order dependencies can aid knowledge discovery, to find hidden properties of the data; in the context of data profiling [LN16], data integration and cleansing [CIP13], order dependencies can be used to describe a dataset; for data quality [FG12], order dependencies can be used as requirements or constraints [AGN15].

The concept of order dependency in the context of database systems first appeared under the name of *point-wise order* [GH81, GH83, GH86]. A point-wise ordering specifies that a *set* of columns orders another *set* of columns.

A new definition for order dependency was later introduced [SGG12b] to represent an order-preserving mapping between *lists* of attributes. In contrast to point-wise ordering, the new definition was distinguishing tuples with attributes in different order, thus having lists of attributes instead of sets.

Dependencies are typically derived from design specifications, from the context of queries or from other known dependencies using inference rules. Discovering dependencies by analyzing the data is a process known as *dependency discovery* [LLLC12]. It conceptually requires to check for all potential dependencies if they hold in the database instance under examination, which may be time consuming. Thus, there is interest in developing strategies that limit the number of combinations to be checked. The task becomes even more challenging in the case of order dependencies, where the order of attributes matters, leading to a search space much larger than that of functional dependencies.

In this dissertation, we study ways for efficiently discovering order dependencies. We follow a bottom-up approach in which we start by checking short lists of columns and progressively check longer and longer lists. In this process, once an order dependency between two lists of attributes is found not to hold, we prune the search space by ignoring larger lists that include them. In this way, many of the combinations that would have normally been checked are avoided. We advocate that this whole process can be significantly improved by framing the discovery of order dependencies in the context of order compatibility dependencies. This is based on a recently introduced theorem [SGG12b] that established that an order dependency holds if and only if a functional and an order compatibility dependency hold between the two attribute lists of

the order dependency. We illustrate in details how the order compatibility dependencies can be exploited to find order dependencies and propose a new algorithm for finding them.

Recently, two algorithms to automatically detect order dependencies in relational data have been proposed: ORDER, proposed by Langer and Naumann [LN16], and FASTOD proposed by Szlichta et al. [SGG+17]. ORDER explores a lattice of order dependency candidates, in a level-wise fashion reminiscent of the TANE algorithm [HKPT99]. After building a dependency candidates, ORDER checks its validity against the data and then it applies pruning rules to reduce the search space over the lattice. ORDER has been shown to be incomplete [SGG+17], i.e. it does not find the complete set of order dependencies. In particular, this approach is unable to discover dependencies with repeated attributes. Dependencies of this form, however, may not be inferred from other dependencies and are useful in the case of queries that involve ordering with multi-column indexes.

FASTOD [SGG+17] is based on a different axiomatization of order dependencies that allows to map dependencies between lists of attributes to dependencies between sets of attributes written in a canonical form. In this way, several order dependencies are mapped to same set-based canonical form. FASTOD explores the space of order dependencies of this set-based canonical form, still retaining the ability to find a complete set of dependencies. While we have reproduced the results presented in the original work, we have found that an implementation error of the original work produces wrong results over simple datasets, this vitiates the validity of their results and the comparison with our approach.

The approach we present in this dissertation is able to provide a complete set of dependencies that is based on the idea that whole process of order dependency discovery could be done with the discovery of *order compatibility dependencies*. While our approach has a higher complexity worst-case complexity than FASTOD, it outperforms the other state of the art approaches [LN16, SGG+17] when tested over real and synthetic datasets.

List of publications. The efficient discovery of order dependency has appeared in [CSMV19].

1.4 Event Extraction

Most of the world's knowledge is currently found in text documents. The textual form is hard to understand, query, manage and in general exploit. Thus, for the contained knowledge to reach its full potential, it is of paramount importance to be able to identify, extract, and convert it into some structured form.

One of the largest textual resources is Wikipedia, that encodes the collective knowledge of millions of users, and the increasing interest in extracting its contained knowledge into some structured knowledge base comes with no surprise. YAGO [MBS15] and DBpedia [ABK+07a] are the two most popular knowledge bases that have been created in this way.

In turning Wikipedia text into structured knowledge a fundamental first task is entity extraction, i.e., the recognition of references to real-world entities [EIV07], and the extraction from the text of information about them, such as characteristic properties and relationships [WM08, SKAZ15]. Another useful information that can be extracted from the text is about events. *Event identification* [SG16, SJY11, HL12] deals with the discovery of "unique things that happened at some point in time" [ACD+98], but so far has not received the deserved attention.

There are different approaches to event identification. One is to decide the structure or type of an event in advance and then analyze the text for finding descriptions that match that predefined structure [AB15, SG16, KW12]. This of course means that the structure of an event is known in advance and that any description in the text complies with that structure, which is not always the case. To overcome this limitation, other approaches have looked for irregular changes in some monitored quantity, to identify that an event is occurring. For instance, search engines may be monitoring for irregular popularity's (like peaks) of a set of terms and use this as an indication that an event characterized by these terms has occurred [LLL+12]. This online monitoring is effective for events that are occurring in the present, i.e., during the monitoring time, but cannot be used to discover historical events for which there was no monitoring operation in place.

Wikipedia is a set of pages, each about some real-world entity. Although each page contains some structured information (referred to as infobox), most of the information

is actually contained in its textual part. The latter may contain references to other pages when the entity represented is mentioned in the text.

We introduce a novel approach to identify historical and contemporary events from Wikipedia content, based on their co-reference relationships in specific time periods. In Wikipedia, every page is about a specific entity, with the text of the page describing properties and characteristics about that entity. When an entity is associated to another, its text contains a reference (link) to the respective page of that other entity. Associations may be mutual. If an entity references another, the latter may also reference the first. We refer to this case as a *co-reference*. Associations have a temporal validity. In Wikipedia, this temporal validity is often stated in the text. For instance, a part of Barack Obama's page states that "... In June 1989, Obama met Michelle Robinson ..." meaning that the association "met" of Michelle Robinson to Barack Obama has the temporal validity of June 1989. Note that the temporal validity of an association is the one of the real world, and not the time that the link was actually introduced in the text of the page. Our position is that an event can be considered the situation in which a set of entities in a specific period of time have a dense set of established associations between them. This offers us the flexibility to detect events independently of the kind of events that they are.

Identifying events in the Wikipedia content through co-reference, poses a number of challenges. First, a set of entities with a number of temporally restricted co-references among them may not always correspond to some real-world event. For this, it is important to increase the likelihood that such a set constitutes an event. Second, there may be multiple co-references among certain pairs of entities, and it is important to decide which of them to consider. Last but not least, it is important to be able to scale to the level of co-references across the millions of pages of which the current Wikipedia consists.

To cope with these challenges we devise a technique that employs the context of the co-references as an additional criterion for improving the precision of our results and avoids combining unrelated co-references.

List of publications. This novel technique that extracts events from a document collections, helping in the discovery of essential passages from a document collection is under submission [SGNV19].

1.5 Finding Synonymous Attributes in Evolving Wikipedia Infoboxes

Wikipedia, with its more than 5.8 million entries[1] is one of the largest human curated sources of knowledge. A Wikipedia entry provides information about some real world entity, of a specific type, like an event, a person, an organization, a product, etc. It consists of two parts: the unstructured part, which is free text, and the structured part, that is known as the *infobox* and is a set of attribute-value pairs. These pairs describe the main characteristics of the entity that the entry describes. The importance of the Infoboxes is significant. They may contain information that is also found in the text of the of Wikipedia entry, yet, they are highly more structured. This means that the semantics of the information they contain is much easier to interpret, queried, analyzed, combined and explored in general. The attributes (i.e., the names of the attributes) to be present in an Infobox of an entity depend on the type of the entity and are dictated by the Wikipedia policies.

The Wikipedia entries are highly dynamic data. Real world entities evolve over time, and so does the knowledge that we have about them. This real world evolution is reflected into the Wikipedia entries. Users are continuously updating the Wikipedia entries in order to always contain in the best possible way the knowledge we have about an entity. This means that by studying the evolution of the Wikipedia pages, it is possible to understand the evolution of the entities through time. To do so, a fundamental task is to be able to identify and link, across different versions in time of the same Wikipedia page, the parts that model the same kind of information. This is typically done using the schema information, i.e., the attribute names.

Unfortunately, the evolution of the Wikipedia entries is not only on the content but also on the attribute names, making the required linking a challenging task. Attribute names are often changed to more accurately or completely represent the semantics of the attribute values in the Infobox. As a result, different attribute names in different versions of the same Wikipedia entry may be used to represent the same semantic information, and the same attribute name in two different versions may be used to model semantically different pieces of information.

[1] https://en.wikipedia.org/wiki/Wikipedia:Statistics updated on 24 March 2019

We deal with the problem of attribute name alignment in Wikipedia pages across time. We want to analyse and identify sets of attribute names that, across the evolution history of the pages of a specific type in Wikipedia, have been used to represent the same semantic concept. At the same time, we want to identify cases in which the same name has been used in attributes that model semantically different information.

Attribute alignment is a well-known problem that has been studied extensively in the past, mainly in the case of schema matching and in ontology alignment. The straightforward approach would be to look for synonym words in attribute names. This is the approach that has been followed in the area of Natural Language Processing. The techniques that have been developed there can be classified in two main categories. The first are those that exploit dictionaries, i.e., being based on the semantics of the attribute names. They are however, limited.

Their limitation lies mainly in the number of synonyms that can be encoded in a knowledge base. Furthermore, they are context-independence, i.e., are not able to differentiate synonyms according to the context in which they are used. Another approach that has been studied in the context of identifying correspondences in Web form schema matching [HCCT16] is one that exploits correlations. The idea behind these techniques is that the search for synonyms is implemented by discovering attributes that correlate negatively or that do not co-occur. Unfortunately, this concept cannot be directly applied in the context of the Wikipedia Infobox attributes. This, because the Wikipedia content is so rich that using only co-occurrence information results in high false positives and also false negatives.

In an effort to overcome the limitations of the co-occurrence approach we have developed an extension of it that exploits the attribute values. In particular, the occurrence of same values between two different attributes in Infoboxes of different versions of the same type is treated as a positive indication that the two attribute names model the same real world concept. Furthermore, we treat co-occurrences as a negative indication. In particular, high degree of correlation (co-occurrence) in Infoboxes between two different attributes is an indication that these two attribute are referring to different concepts, thus, the high correlation is a negative indication. We turn the above two indicators into two different metrics, and create a network of attribute names where the values of the different metrics are used as a distance function. Then we apply clustering [DEFI06] to identify those

sets that form mutually close names. Such sets are those we consider as semantically related.

List of publications. This novel technique that extracts synonymous attributes in evolving Wikipedia Infoboxes is appeared in [SPGV19a].

1.6 Finding Movement of Values in Evolving Wikipedia Infoboxes

Nowadays, the amount of data recorded is growing enormously. This data changes all the time, reflecting what happens in the real world. To capture real-world happening datasets are organized with entities. One of the main examples is Wikipedia, that is one of the largest sources of knowledge that is freely available in the World Wide Web. Wikipedia provides information about real-world entities such as singer, companies, and other types of entities. The information related to these entities are stored in the Infoboxes. These infoboxes represent entities, and they are composed by attributes that model their characteristics.

Entity models actors of the real-world, and their action, and changes also affects what the data. Entities are the conceptual models that represent real-world objects, and they are composed by some characteristics called attributes such as name = "John", surname = "Doe", or location = "Rome". Attributes are composed by attribute name (e.g., name) and value (e.g., "John"). Both of them can change over time. These attributes represent the characteristics of the entities. We advocate that entities are not monolithic containers of information, but they are interconnected, entities can share common attributes as characteristics, or they can have more complex interactions over time. The evolving nature of the real-world entities is translated into changes in the data that reveals rich insights that cannot be derived from static data. In this scenario, we focus our attention on the discovery of a particular kind of interaction between entities that we call movements.

A movement takes place when a value disappears from an entity and then appears in another one. Obviously, not all the values that disappears or appear would generate a movement. The challenge is to identify changes that are generating a move. More

specifically, we are interested in finding a characteristic of an entity that is moved from one entity into another one. A movement represents an event that may have different meanings. For example, an entity that represents a person that disappears from one company and appears in a different one may represent a job change, or an entity that represent the company as a property of another is transfer in a different one may represent an acquisition.

Many recent works have been proposed in order to analyze and explore evolving datasets. An entire line of research has been recently introduced to extract insights from changes in the data. Their limitations lie mainly in the fact that they consider changes independently without reasoning about possible interactions and exchange of values between them [BBK+19, BBJ+18, BBK+18].

To identify movements in evolving Wikipedia Infoboxes, we proposed a two-step approach, where we first filter values that are less probable to generate a move, and then we score them in order to select the most probable movements. Since movements are generated from changes, we identify two types of changes of values that are not responsible for the movements. These two types of changes are systematic changes and changes that are happened in a close domain attribute. The first type of change identifies changes that are imposed from the outside, i.e., change of denomination of a characteristic or new conventions, while the second identifies values that are always used in the same domain, e.g., the type of a company (private, public or subsidiary). After the exclusion of values that are involved in this type of changes, we generate generate from the remaining values all candidate movements. We score them and finally filter out the ones that are not greater than a certain threshold.

The identification of moving values in evolving dataset create opportunities for the creation of different types of exploratory applications. For example:

- We can measure the similarity of entities base on the exchanges of values that they had during their lifespan.
- We can classify values that move with a similar pattern between the entities and cluster values or entities with similar behavior.
- Extract frequent movement patterns.

- Reconstruct values history: the values could also be entities, and in this case, moves of entity e across other entities in the database D describes the history of that entity. For example consider a person that takes a leading position in different companies during her/him lifespan, looking at the movements of that specific value (e.g., person) we are able to reconstruct all the job history of this person.

1.7 Other contributions

This Ph.D. was instrumental in studying other topics, which are not included in this dissertation. In particular, I was involved in a European project named: "Re-Search Alps" that aims to gather, consolidate, harmonize and make available to different target groups (public and private bodies) data about laboratories and research and innovation centers active in the seven countries that constitute the Alpine Area. In the context of this project, we realize several contributions:

- An innovative rule based framework for entity matching [SPGV19b];
- We develop a prototype system that is able to fine-tune entity matching rules [PSGV19];
- We proposed a parallel and scalable method for the discovery of Full Disjunctions [PBGS19];

1.8 Applications

Data Exploration. The first three and the last methods described in this dissertation can be used in the context of data exploration techniques [IPC15, FAK+18]. Exploratory data analysis aims at finding patterns, regularities or common properties that help users in understating the if some information can be extracted from the analyzed data. For example, consider the scenario of the so-called data lakes, a set of accumulated datasets arrived in some organization. Experts need a basic understating of their context. In this context, our techniques are suitable for this task. If a user needs an overview of the content of a database our data explanation could be used to guide the user in the discovery of common properties of the data or contrary in the discovery of outliers.

Instead, our second contribution could be used to profile the input dataset, providing a set of dependency. In the case of textual data, we may help the user exploring the important events encoded.

Data Cleaning. The first two contributions could serve the aim of data cleaning. Data cleaning aims at detecting potential records in the data that violate rules that could be extracted from an error detection system or from other sources [GKS11]. In this context, data explanation could really help the user in analyzing anomalies in the data and detect possible mistakes in the data. Instead, the metadata provided by order dependencies could provide other information regarding columns dependencies of the database. Different strategies could be applied over the database in order to remove or correct the violations [IC15].

Query optimization and Indexing Space. Our second contribution, order dependency discovery could be used in the context of query optimization [SSM96]. Query optimizers are gathering statistics and other metadata in order to optimize the selectivity of the different operators and other optimization steps [KHM+06]. Order dependencies could be used by a query optimizer in order to simplify ORDER BY clauses. Moreover, order dependencies are extremely important resources to reduce indexing space, since indexes over some columns may be reused for other columns if a dependency holds [DH82].

Search engines. The third contribution, the historical event discovery technique, could find a natural application for search engines, where the extracted events could improve the results of temporal queries by proposing documents related to the more connected entities in a certain period of time. The event sets extracted from our techniques could be used to propose a diversify view of documents with the aim of covering the entire period of an entity or concept, i.e. with an application that is showing a timeline of significant events of an entity [ADM+15a].

1.9 Outline

This dissertation is organized as follows. Chapter 2 provides a complete overview of the prior work in data explanation, dependency discovery, event discovery, schema matching and evolving dataset analysis. Chapter 3 presents our framework for the data explanation problem. We present a principled approach that provides a set of metrics that helps the user in finding the more interesting data descriptions. We also introduce algorithmic solution and some heuristics that we evaluate over qualitative and quantitative aspects, and user satisfaction.

In Chapter 4, we present our study on order dependency discovery, introducing a new solution to this problem by decomposing their discovery in the discovery of two other types of dependencies: order compatibility and functional dependencies. We introduce a new definition of minimality for this kind of dependencies, and we provide an approach that discovers the complete set of dependencies while outperforms the existing techniques.

Chapter 5 presents our third contribution, a new method for historical event detection that is able to extract important events from annotated document collections. We show the effectiveness of our approach with the largest textual encyclopedia on the web, Wikipedia. We provide an extensive experimental evaluation, where we evaluate our work with existing approaches and events collections, and therefore we validate our extraction method via user study.

In Chapters 6, we present our approach for finding synonymous attributes in evolving that identify clusters of attribute names of Wikipedia Infoboxes that represents the same characteristic of an entity. We present our data model and the methodology used to extract clusters of synonymous attributes. We show the effectiveness of our approach over a set of Infoboxes of the last decades.

Chapter 7 presents our last contribution: the identification of movement attributes in evolving datasets. We provide a formal definition of a movement and we provide an exhaustive description of the extraction method. We also provide an experimental evaluation that shows the potential of our analysis.

Chapter 8 concludes this dissertation by providing the main contributions and overviewing the most promising future directions that follow this dissertation.

Chapter 2

State of the art

In this chapter we provide an analysis of the different aspects that are related to work on data explanation, order dependencies discovery, and event extraction. First, we survey the current studies on explanation systems and techniques devoted to explaining data and data transformation (Section 2.1). On the topic of order dependencies discovery, we present a description of topic covering all the different aspects: from the theoretical perspective, its applications and the existing approaches demanded at the discovery of order dependencies (Section 2.2). Consequently, we present and overview of methods that have been successfully employed in different task related to event identification (Section 2.3). Finally, we survey an overview of methods and applications related to evolving datasets, with particular attention to the identification of schema mapping and exploratory analysis over evolving datasets (Section 2.4).

With this survey, firstly we aim to motivate the importance of the study on information extraction, with particular attention on the discovery of the explanation of data, dependencies discovery and event discovery. Secondly, we identify the gaps in the literature that pushed for the need for new techniques in these connected areas of the knowledge and information discovery .

2.1 Data Explanation

To better capture the differences with our approach, we split the discussion of our related work into different families of systems devoted to the explanation of data. Explanation systems are those that help users in gaining knowledge on unexpected behavior of systems, experiments or query answers [MRS14].

2.1.1 Provenance-based Explanation Systems

Data lineage, often called data provenance, is a mechanism that tries to explain how query results are related to source data [CW03]. Authors in [BKT01] identifies a different form of lineage – namely *where*, *why* and *how* – under the common term of *data provenance*. While *why*-provenance specifies the input tuples sufficient (but not necessary) to have a given target output tuple, *how* provenance is more informative in the sense that it also encodes how data transformations combine input tuples to generate outputs. In particular, *why*-provenance is usually modeled as tables containing input-output tuples relationship, *how*-provenance can be modeled using annotations representable as semirings of polynomials [GKT07]. Different database and workflow systems exist embracing one type of provenance (*why*-provenance [IST+15, IPW11, LDY13, CLMR16]) instead of another (*how*-provenance [ADD+11, DMT14, ABD+15, WDM15]). In general, we found that both types of systems provide unreadable results is scaled over large datasets: the former because of the size on input-output relationships can grow beyond the gigabyte range; the latter because of its complexity. PROX [ABD+15] uses semiring homomorphisms to generate summaries of provenance annotations in favor of readability and understandability rather than completeness.

Finally, there are many domain-specific systems that enrich their duties with explanations for human users. To mention a few, Fabbri et al. produces explanation templates over access logs to be used in auditing tasks (e.g., to answer questions like "why a record was accessed?") [FL11]. Bender et al. enriches security and monitoring systems over a database with explanation for why an access was accepted or denied [BKG14]. This helps the maintenance of security policies and the reformulation of denied queries.

2.1.2 Intervention-based Explanation Systems

The traditional data lineage presents several limitations when it comes to explain outlier results because users want to know which subsets of the lineage have a "bad" influence on the outlier result. Meliou et al. pioneer this research area by studying causality, as opposed to correlation, in the database area. They identify tuples, seen as potential causes, that are responsible for answers and non-answers to queries [MGMS10]. To pursue this task, they introduce the degree of responsibility to measure how responsible these tuples are. In general, systems of this category delete candidate solutions, i.e., groups of tuples, from the database and evaluate whether the outlier has changed without those tuples. This process is called intervention and it is iteratively repeated in order to find the most influential groups of tuples, i.e. explanations [MGMS10, ROS15, RS14, WM13].

Scorpion [WM13] explains a (possibly outlier) query result by individuating responsible tuples in terms of influence on that result. They additionally introduce the use of predicates for building explanations, which we were inspired of, because more informativeness to users. The intervention over predicates is based on a partitioning and merging process that uses decision trees and relies on certain properties of the aggregative operator and on the influence metric.

Roy et al. mix intervention with causality to overcome the limitations of Scorpion in generating explanations over a single table only [RS14]. Their intervention is propagated on causal paths that connect tuples of different datasets via foreign key constraints. In addition, they make use of data cube operations to support intervention. Generating explanations with intervention is a computationally heavy task since the number of predicates to process are exponential in the number of tuples. To efficient the explanation, there have been extensions of former approaches that prepare offline the database for the online generation of explanations [ROS15].

Analogously to previous approaches, Kanagal et al. produce explanations over probabilistic databases out of the tuples that maximize the probability change of the result when those tuples are set to zero [KLD11].

To overcome the high complexity, online explanations have been proposed [ROS15]. Interpretable decision sets have been proposed in the context of descriptive rules for

decision tree [LBL16]. The final aim of these explanation systems differs from our. While they focus on finding tuples that maximize the influence over a subset of one or more query results (i.e., they do not have a validity constraint), we aim at representing a dataset in a way that is more succinct, readable and understandable.

Recently, a *smart drill-down* has been proposed to select, interactively, the k most interesting rules (according to a certain score function) that describe a portion of the dataset [JGP16]. Unlike our approach, this strategy focuses only on broad (i.e., high coverage) descriptions, and does not allow users identifying information with low coverage or with high entropy.

The final aim of these explanation systems differs from our. While they focus on finding tuples that maximize the influence over a subset of one or more query results (i.e. they do not have a validity constraint), we aim at representing a set of tuples in a way that is more succinct, readable and understandable for human users.

We avoid any form of intervention by proposing a framework that is able to describe a dataset with different granularity thresholds, i.e., from a general description to outlier description, that uses two simple user-specified parameters, it is able to produce a simple and complete description in an online fashion.

2.1.3 Debugging-oriented Explanation Systems

When creating the data processing pipeline for data analysis, not only the explanation on the results is useful but the analyst benefits also from tools that help to understand what has happened during the computation. These tools stand to data analysis as debuggers stand to software engineering and DevOps [GR13, DG15, GIY+16, OR11, DZS13, KBS12, YNM16]. PerfXplain [KBS12] and DBSherlock [YNM16] debug the performances of query processing. PerfXplain explains the performance of MapReduce jobs that take unexpectedly long in a shared-nothing cluster. DBSherlock combines outlier detection, causality and domain knowledge for explaining unexpected performance behaviors. Finally, the works in [GR13] and [DG15] address the problem of debugging SQL queries. BigDebug [GIY+16], Inspector Gadget [OR11] and Arthur [DZS13] explore techniques and primitives for debugging data-parallel, distributed, workflows. All these approaches are different with our but we found that we can integrate well on these kinds

of debuggers so that the analyst can have the full set of explanations, at process level and data-instance level.

2.1.4 Data Reduction and Compression

There are techniques to represent databases in more succinct ways, either lossless or lossy.

Factorised Databases. Factorisation is a lossless representation, called f-representation, for the physical implementation of in-memory relational databases [BOZ12, OZ15]. Because they eliminate the inherent redundancy of the relational model when more tuples share values on one or more attributes, certain classes of query processing can expedite their execution over the factorised database rather than on the traditional counterparts [BOZ12]. In a recent work, d-representations are introduced to further compact f-representations and the size bounds are studied for generating succinct factorized representations [OZ15]. Factorized databases, like us, uses conjunctions and disjunctions to represent data but, since they have to be lossless, their succinctness is focused on query execution and not on understandability.

Summarization and Histograms. Summarization is about creating lossily compressed representations of database schemas or tables [YPS09, YPS11]. They are normally used for user presentation, and query or data access optimization. Yang et al. [YPS09, YPS11] create summaries about the type of information in the database, the most important tables and most important join paths between the tables.

Another kind of summarization can be done with histograms [KM10, MDM+14]. They keep track of the frequency of data values for the attributes and thus help the optimizer in making the right decision. Histograms cannot be used to summarize a portion of the database but it is actually in our future work to adopt them for improving the accuracy of descriptions.

2.1.5 Exploratory and Visual Database Access

Considering that use cases of discovery of the database content with subsequent descriptions, we consider as related work also the approaches that allow exploratory and visual

interaction with the database [RWD+08, BMQ+15]. Faceted search provides an interactive way to refine query results, where the a priori knowledge of the database is not necessary. They group underlying tuples of the database, choosing from both relevant tuples and attributes. In doing so, they focus on the diversification of results and in the mitigation of the information overload, that is to order attributes and values that have to populate the facets. At the same time, they have to consider the minimization of the costs of such groupings and selections in order to guarantee interactivity with the systems [RWD+08]. In a similar way, there is a branch of research that extends these visual approaches to more analytical data manipulation scenarios [BMQ+15]. In the context of data quality, Profilier [KPP+12] aims at discovery data quality issues such as missing, erroneous and duplicate values. These quality issues cost a lot of effort in order to be discovered and corrected. To this end, this framework applies data mining methods to automatically flag problematic data and suggests coordinated summary visualizations for assessing the data in context. This is done with the integration of statistical and visual analysis that supports real-time interaction with huge amount of data. Instead, in the context of data streams, VSOutlier [CWR14] support outlier detection strategies with a large set of interactive interfaces in order to explore outliers in an online fashion.

These approaches solve a problem that is orthogonal to our, but in the future work we can benefits from the diversification of descriptions or for employing a refining mechanism.

2.1.6 Feature Selection.

In literature, we have other approaches that focus on selecting a subset of attributes among the available ones. In particular, Das et al. propose a search algorithm that is able to select (and order) the attributes that the DBMS should select to visualize the results to a query, when they are too many to be displayed altogether [DHKS06, KNPZ16]. Their selection is based on the maximization of the usefulness of the attributes to the user, so that only the most useful are shown.

2.2 Order Dependencies Discovery

Dependencies play fundamental roles in different database areas: database design, data quality management, and knowledge representation. In database design, dependencies

are extracted from requirements, extracted dependencies are used in the database normalization process and they are encoded in the database design process to ensure data quality. Instead, in the context of knowledge discovery, dependencies are extracted from data. This process is called knowledge discovery, and it aims to find from data all dependencies satisfying the input requirements.

2.2.1 Functional Dependencies: Theory, Discovery and Applications

The literature on functional dependencies is vast [Arm74, LLLC12, PEM+15, HKPT99].

Applications of functional dependencies span several fields from query optimization [DD89], to data cleaning [CIP13], to data quality management [FG12]. Given this variety of applications, several algorithms for the discovery of functional dependencies have been developed.

The first algorithm to be proposed was TANE [HKPT99], which has served as inspiration for many subsequent efforts [PEM+15, LLLC12]. Research on better functional dependency discovery algorithms is still ongoing [PBF+15]. Functional dependencies have been extended in several ways: from conditional functional dependencies [BFG+07], to approximate (or partial) functional dependencies [LLLC12].

2.2.2 Order Dependencies Theory and Applications

In the 1980's, Ginsburg and Hull were the first to consider the idea of analyzing orderings between the attributes of a relation as a kind of dependency [GH81, GH83, GH86]. They introduced the concept of point-wise ordering [GH86], that is a relation where a set of attributes orders another set of attributes. Recently, Szlichta et al. introduced the concept of order dependency [SGG12b], which is the one used throughout this dissertation. Order dependencies are defined over lists of attributes, and can be formalized in a similar way to functional dependencies [SGG12b, SGGZ13a]. We focused our attention on dependencies where the attributes are all ordered in the same direction, also called "unidirectional" order dependencies; however order dependencies can be generalized to "polarized" or "bidirectional" ODs where a different direction of the ordering can be specified for each attribute on either side of the dependency [SGG12a]. One of the main

theoretical problems concerning dependencies in relational data is the problem of inference. For order dependencies, this problem is shown to be co-NP-complete [SGGZ13b].

2.2.3 Applications of Order Dependencies

Notwithstanding their recent theoretical formulation, order dependencies have been already used in several applications, such as query optimization.

Sorting is a fundamental database operation. Since the seminal works [GH83], research has focused on developing optimization strategies for dealing with queries with an `ORDER BY` clause [SSM96]. Order dependencies can be used for this purpose, as it has been shown with an implementation of a query optimizer in IBM DB2. Optimizing queries with order dependencies yields significant speedups in execution times over the well-known TPC-DS benchmark and on queries obtain by real-world scenarios [SGG+14]. In this work, the authors describe how order dependencies can be used for query optimization. For example, consider the SQL statement "select name from TaxInfo order by bracket, tax," which queries the names of all people, sorted by bracket and – within equal brackets – by tax. A database management system may initiate a sort operation on bracket and – within equal brackets – on taxes. If, however, the order dependency "tax orders bracket is known to the optimizer, it can rewrite the above order by statement to "order by tax," because the order dependency "tax orders bracket" implies that the ordering of bracket is subsumed by the ordering of tax. By reducing the order by statement, it is, for instance, more likely that the entire sorting operation can be satisfied with the help of an index. Likewise, a query "...order by bracket" can be rewritten to "order by tax," which is beneficial if there is an index on tax, but not on bracket. In the context of indexing, Dong and Hull show how order dependencies can be used to reduce indexing space [DH82] in a DBMS.

Furthermore, Szlichta et al. [SGG+14] show how from order dependencies declared by a database administrator, new order dependencies can be inferred. They also show that order dependencies may originate from the use of algebraic expressions or SQL functions within a query: for example in case of dates, the order dependencies "date orders (date + 30 days)" always holds. This means that dates are "ordered" by month, consequently an order dependency holds for each month.

2.2.4 Discovery Algorithms for Order Dependencies

These applications are driving the need for discovering order dependencies in existing datasets. The first work proposing an algorithm to solve this problem is the one by Langer and Naumann [LN16]. Their approach, called ORDER, follows the path of TANE, computing the potential order dependency candidates from the permutations of attributes and traversing the lattice in a level-wise, bottom-up manner. Pruning rules are applied to reduce the number of candidates to check, with the caveat of eliminating only redundant relations that can be later inferred from the dependencies discovered together with the axioms. However, as shown in this dissertation (Chapter 4), this approach does not consider all the possible order dependency candidates, discarding repeated attributes. While this gives a significant advantage in execution time, reducing the worst-case time complexity of the algorithm to $O(|\boldsymbol{r}|!)$, its major drawback is the possibility of losing completeness. More recently Szlichta et al. [SGG+17] proposed an algorithm called FASTOD that is complete and faster than ORDER. This algorithm exploits a novel polynomial mapping that transforms ODs with lists of attributes into canonical forms of ODs that are established between sets of attributes. FASTOD has exponential worst-case time complexity, $O(2^{|r|})$, in the number of attributes. This last work provides significant advances in the theoretical decomposition of the problem of order dependency discovery. However, the results presented in their work present some erroneous dependencies, and not precise results in their experiments. We tried to analyze where and why sometimes their approach fails in discovery the correct set of order dependencies.

2.3 Event Discovery

Automated extraction of events from textual data plays an important role in knowledge base construction [KW12], entity description [ADM+15b] and question answering tasks [YBRW13]. To this end, several techniques have been proposed that do not only differ in corpora and extraction processes, but also in their underlying representation of extracted events.

2.3.1 Event models

Many event models describe an event as "some unique thing that happens at some point in time" [ACD+98].

This definition can be further extended by including locations [SG16], dynamic relationships [SJY11], a set of actors or entities [SG16, SJY11] or textual descriptions [KVW14, HL12] into the extracted events. While these models are flexible towards entity types and arity, in knowledge bases events are usually represented through strict schemas which often requires triple notations and reification.

A slightly different notion of events is *named events* [KW12, KVW14] which are defined as a concept similar to traditional entities. While this is applicable for well-defined events like "FIFA World Cup 2014", more fine-grained events like "Barack Obama received the Nobel Peace Price" cannot be represented. Other approaches are less restricted to given schemas but do not allow for easy comparison of such events: they employ natural language processing models [ZSW15] to extract very fine-grained relations between two entities that are simply represented as triples consisting of subject, object and a textual relation like "arrives in". In contrast to the aforementioned knowledge base models, these models lack additional information like the time dimension and do not put any emphasis on events of major importance. Instead of using short expressions that connect entities, [AB15, HL12] and [WNRR16] identify whole sentences representing events.

2.3.2 Real-time event detection

In the case of user-created content like Wikipedia or Twitter, events can be detected by finding peaks in activity regarding single entities or sets of them [LLL+12]. Analogously, [SJY11] locates time periods when an entity appears in more documents than usual given a time-stamped document collection. Having mapped news documents covering the same theme onto a time axis, it becomes possible to hierarchically arrange events, e.g., by denoting sub-events [KVW14]. A different approach exploits a graph of entities that is continuously updated with weights extracted from tweets, and dense subgraphs are identified as events [AKSS12]. All approaches focus on the discovery of current events and need extra streams of information for extracting interactions between the entities to finally detect events.

2.3.3 Historical event extraction

In contrast to real-time event detection, which focuses on limited time spans in the present time, historical event detection aims at retrieving events happened before the emergence of the Web. Moreover, temporal perception of events correlates with the actual happening, which is not always guaranteed, e.g., in cases of belated death reports. To overcome these issues, corpora with historical or encyclopedia texts are used to extract their entities, time and textual descriptions: for example, contains pages with lists of historical events that can be extracted [HL12]. The same approach has been applied to the Wikipedia current events portal [TA14], covering events of the current century.

When not relying on a corpus with predefined events and time spans, automated extraction of time expressions in the text becomes necessary which can be done by using tools like SUTime [CM12]. This approach, combined with an appropriate disambiguation of entity mentions in the text, was used to group together sentences representing the same event [AB15]. Another approach automatically induced textual patterns like "⟨entity⟩ was inaugurated as ⟨entity⟩ on ⟨date⟩" within Wikipedia articles in order to extract event triples that are inserted into the YAGO knowledge base [KW12].

Compared to purely text-based approaches using NLP techniques to extract event mentions [ST17], our approach does not rely on explicit mentions of an event within a brief text fragment. Instead, events can be found based on the assumption of temporal co-reference and constructed even from mentions in several sentences from different documents. All these methods suffer limitations based on their event representation since they focus on predefined relations and fixed constraints on the kind of the entities involved in the event.

2.3.4 Temporal relationship mining

Temporal relations between entities and the resulting entity networks play an important role in event extraction. A well-known measure for semantic relatedness between two entities uses a graph extracted from Wikipedia that relies on link counts [WM08]. Other approaches use external tools like search engines [SKAZ15] in order to find textual patterns (relations) between entity pairs. To detect temporal relations, this approach was

extended [SZK+16] by the use of the Internet Archive which supports relation extraction since 1996. A temporal entity relationship graph was built from Wikipedia by combining terms, locations, organizations, actors and dates [SG16]. This joint graph can be queried for relationships between specific types of entities at specific time points. However, all approaches focus their attention on the relationships between entities and they do not lead to the extraction of well-defined events.

2.4 Finding Synonymous Attributes and Movements in Evolving Wikipedia Infoboxes

2.4.1 Importance and usage of Wikipedia infoboxes

Wikipedia infoboxes have been used in a large number of research projects. The most significant works include techniques for building structured knowledge bases [SKW08, ABK+07b], for analyzing the evolution of specific kinds of entities [HSBW13] and for applying structured queries on the Wikipedia content [AS17]. Although several works support the formulation of structured queries, no previous effort has considered the evolutionary nature of Wikipedia. All previous approaches consider only a static snapshot of the infoboxes as input.

2.4.2 Schema matching

Schema matching is one of the most studied topics in the database community. Books [BBR11, ES07] and surveys [RB01, BMR11] introduce the existing approaches in the literature. According to the categorization proposed by [RB01], schema matching approaches can be classified into schema-only matchers and instance-based matchers. Our proposal follows a hybrid approach since we incorporate holistic correspondence refinement that belongs to the category of collective matching approaches [BMR11].

Our approach implements a strategy similar to the one proposed by He et al. [HCCT16] that consider as negative evidence the co-occurence of attribute names in the same schema. Their approach focus on the discovery of synonyms supporting a web search engine and is based on the combined use of web tables and query logs. The intuition is

that users who are looking for the same results provide different synonyms as query terms on a search engine. This is used as positive evidence for attribute synonymy. Instead, attributes within the same web tables are not likely to be synonyms, thus providing a negative evidence. Our approach adapts that idea to work with Wikipedia infoboxes by extracting positive and negative evidence of synonymy analyzing co-occurrences of attributes and values.

2.4.3 Schema matching and alignment over infoboxes

Schema matching techniques have been applied against Wikipedia infoboxes in the context of finding correspondence between schemas in different languages. Adar et al. [ASW09] propose a framework called Ziggurat that creates a supervised classifier based on features that are learned from a set of positive and negative examples extracted from data with heuristics. Bouma et al. [BDI09] match attributes based on the equality of their values. Two values are equal if they have the same cross-language link or exactly the same literals. A different approach [RLN13] exploits value similarity over infobox templates, where first an entity matching process is done, then templates are matched to obtain inter-language mappings between templates and finally attribute matching is done by means of similarity metrics. These approaches rely on similarity metrics that are sensitive to the syntax of the underline languages: they work well if the compared languages have the same root. To overcome this limitation [NMN+11] exploits different evidences for similarity and combines them in a systematic manner.

2.4.4 Exploration of schema and value changes

In the context of data exploration, a new line of research focuses on the exploration of changes over time. The first work [BBJ+18] is a vision paper that introduces innovative concepts related to understanding changes that happen in the data over time. This paper introduces the change cube that models the changes in a dataset. The four dimensions represented are Time, Entity, Property, and Value. Then a change is formally defined as a quadruple in the form $\langle \text{timestamp}, \text{id}, \text{property}, \text{value} \rangle$. This work was further expanded in [BBK+19] and [BBJ+18]. These papers further explore the concept of change cube by introducing a set of exploration primitives that exploit the change cube

to find insights in evolving datasets. In this context also a clustering algorithm based on time series has been proposed [BBK+18].

Chapter 3

Explaining Datasets with Descriptions

We now focus on the data explanation problem and, more specifically, on the study of a generic and easy to use method for generating data descriptions. Descriptions are useful in contexts where users need some form of explanation. While existing data explanation tools and techniques are used to gain knowledge on specific tasks such as unexpected behaviors of systems, experiments or query answers, data descriptions are generic inasmuch as they are able to concisely represents any arbitrary set of data tuples.

In this chapter, we present a principled framework for the data description problem, that is able to satisfy many user's preferences. We demonstrate that the problem of finding dataset descriptions is hard and we proposed an approximation based on some heuristics that are able to produce high quality descriptions in a very short time. We also include experimental results and a user study that demonstrate the quality results of our framework.

3.1 Contributions

The main contributions in this chapter can be summarized as follows.

- We introduce data descriptions as a new form of explanation targeted to help users in understanding the datasets at hand.

- We show that the problem of generating descriptions is NP-hard, we provide a complete proof based on reduction to the weighted set cover problem, that is a well known NP problem.
- We define a principled approach for generating descriptions over sets of data records, we discuss the different aspects that constitute a description. We discuss the principles that guide our work in the construction of our framework.
- We propose quality metrics for data descriptions on the basis of their understandability and expressiveness, these metrics are used by the user to guide the explanation system in the generation of data descriptions.
- We develop a set of heuristics that allow the efficient generation of descriptions while maintaining high quality. These heuristic impact in a very efficient way in the fast creation of descriptions, searching for the most top-k descriptions over the whole possibles.
- We conduct extensive experiments over real datasets to evaluate the efficiency and the effectiveness of the approach. We analyze both qualitative and quantitative aspects in description evaluation. We measure the impact of our heuristic with respect to the time needed to produce the description. We also compare our work with some baselines and we also conduct a user study in order to measure the satisfaction of description with respect to user desires.

3.2 Outline

The chapter is organized as follows. In Section 3.3 we present a motivating example used through the rest of the chapter. Section 3.4 formally defines data descriptions; Section 3.5 introduces the principles that drive our approach and the quality aspects for evaluating descriptions. Section 3.6 defines the framework and heuristics devised to find the top-k descriptions. Finally, Section 3.7 contains the experimental evaluation.

	Time	SensorID	Voltage	Humidity	Temp.
t_1	11AM	1	2.6	0.4	34
t_2	11AM	2	2.6	0.5	35
t_3	11AM	3	2.6	0.4	35
t_4	12AM	1	2.7	0.3	35
t_5	12AM	2	2.7	0.5	35
t_6	12AM	3	2.3	0.4	100
t_7	1PM	1	2.7	0.3	35
t_8	1PM	2	2.7	0.5	35
t_9	1PM	3	2.3	0.5	35

Table 3.1: The SENSOR dataset.

3.3 Motivating Example

Consider the SENSOR dataset of Table 3.1 containing sensor measurements, and taken from [WM13]. Each measurement is a tuple, detected at a certain Time from a sensor device identified with a SensorID, which consists of Voltage, Humidity and Temperature measures. For the sake of brevity, we associate the short labels `Ti`, `S`, `V`, `H` and `Te` to the attributes following the order of appearance in Table 3.1. Assume now a user who wants to analyze SENSOR and need a supporting tool for initiating the task. A description is a suitable support for such type of users. Consider the example in Table 3.1: a framework for data descriptions should be able to choose among the descriptions in Fig. 3.1 as well as every other possible description in order to output the most relevant.

Example 3.1. *A naive description D_0 of* SENSOR *consists of the set of tuples themselves. Each tuple in D_0 is a set of attribute-value pairs. For instance, a description for t_1 is* $\{\texttt{Ti} = 11AM, \texttt{S} = 1, \texttt{V} = 2.6, \texttt{H} = 0.4, \texttt{Te} = 34\}$*, whereby D_0 includes such description and all the analogously formed descriptions for the remaining tuples $t_2, \ldots, t_9$.*

D_0 is verbose and arguably does not help the analyst in understanding the content of SENSOR. Desirably, when a user asks for a description, she wants an outline of the dataset that is both easier to read and understandable despite the loss of information that results from summarizing data.

Example 3.2. *D_1 of Fig. 3.1 is a description conjoining a list of predicates, each characterizing an attribute of the dataset with the related set of domain values. D_1 is more concise than D_0 because it removes value duplicates but it introduces a loss of information since we cannot reconstruct the original tuples from it.*

D_1 : $\texttt{Ti} \in \{11\text{AM}, 12\text{AM}, 1\text{PM}\} \wedge \texttt{S} \in \{1,2,3\} \wedge$
$\texttt{V} \in \{2.3, 2.6, 2.7\} \wedge \texttt{H} \in \{0.3, 0.4, 0.5\} \wedge \texttt{Te} \in \{34, 35, 100\}$

D_2 : $\{11\text{AM} \leq \texttt{Ti} \leq 1\text{PM}\} \wedge \{2.3 \leq \texttt{V} \leq 2.7\} \wedge$
$\{0.3 \leq \texttt{H} \leq 0.5\} \wedge \{35 \leq \texttt{Te} \leq 100\} \wedge \{1 \leq \texttt{S} \leq 3\}$

D_3 : $\{11\text{AM} \leq \texttt{Ti} \leq 1\text{PM}\} \wedge \{0.3 \leq \texttt{H} \leq 0.5\} \wedge \{1 \leq \texttt{S} \leq 3\}$

D_4 : $(\texttt{S} = 1 \wedge \texttt{H} \in \{0.3, 0.4\}) \vee (\texttt{S} = 2 \wedge \texttt{H} \in \{0.5\}) \vee (\texttt{S} = 3 \wedge \texttt{H} \in \{0.4, 0.5\})$

D_5 : $(\texttt{H} = 0.3 \wedge \texttt{Te} = 35) \vee (\texttt{H} = 0.4 \wedge \texttt{Te} \in \{34, 35, 100\}) \vee (\texttt{H} = 0.5 \wedge \texttt{Te} = 35)$

D_6 : $(\texttt{H} = 0.4 \wedge \texttt{Te} \in \{34, 100\}) \vee (\texttt{Te} = 35 \wedge \texttt{V} \in \{2.3, 2.6, 2.7\})$

Figure 3.1: A subset of the possible descriptions for SENSOR.

When we have complex schemas and instances, a description like D_1 is too long and still difficult to read. A shorter description is preferable. In this case, the selection of the features and of the domain values for the predicates is a crucial task.

Example 3.3. *D_1 can be compacted by using other forms of predicates. For instance, D_2 of Fig. 3.1 increases the readability of continuous attributes by using range predicates. D_2 can be additionally shortened by limiting the set of attributes being employed in the predicates composing the descriptions. For instance, D_3 uses `Ti`, `H` and `S` attributes only.*

In other scenarios, users are interested in understanding the values assumed by specified attributes and how they vary with respect to the rest of the dataset. In this case, the attributes of interest constitute the *pivotal elements* of the descriptions.

Example 3.4. *D_4 and D_5 show examples of descriptions partitioned over pivotal elements. D_4 shows the values assumed by the attribute `H` for each sensor. In D_5, the user is interested in the values of `Te` registered by the sensors when `H` changes.*

3.4 Data Descriptions

Let us consider a dataset I composed of a set of tuples over a set of attributes $A = \{a_1, a_2, \ldots, a_n\}$, where each attribute a_i has associated a domain of atomic values. A tuple t is in the form $t = \langle v_1, v_2, \ldots, v_n \rangle$ where each value v_i is contained in the respective domain. Finally, given a set of tuples I, in the following we will use $adom_I(a_i)$ to denote the active domain of attribute a_i for I.

Descriptions use predicates defined as follows.

Definition 1 (Predicate)**.** Given a dataset I over A, a predicate p is an atomic formula over A in one of the following forms:

- Set Predicate $p : a_i \in \{v_1, v_2, \ldots, v_m\}$, where $a_i \in A$ and each $v_i \in adom_I(a_i)$.
- Range Predicate $p : (v_l \leq a_i \leq v_u)$, where $a_i \in A$ and both $v_l, v_u \in adom_I(a_i)$.

When a set predicate has only one value we will refer to it as *atomic* and use the notation $p : (a = v)$ e.g., $\mathtt{S} = 1$ of D_4 in Figure 3.1.

Definition 2 (d-formula)**.** Given a non-empty set of tuples I over A, a d-formula d for I is a conjunction of predicates over A, such that:

1. an attribute in A occurs in d at most once;
2. each predicate $p \in d$ is true in I according to the classic interpretation of atomic formulas in first order logic;
3. (*full coverage*) $\forall t_i \in I$, d is true for t_i.

Descriptions are usually computed over (horizontal) *partitions* of a dataset I. A partition is a non-empty set of tuples J_i such that $I = \bigcup_i J_i$ and $\forall i, j$, $J_i \cap J_j = \varnothing$;

Definition 3 (Description)**.** Let I be a non-empty dataset over a set of attributes A, and organized into ρ partitions, such that for each partition $i \in 1...\rho$ a d-formula d_i over A exists. A description D for I is a non-empty disjunctive-normal formula of ρ d-formulas $\{d_1, \ldots, d_\rho\}$.

The approach we propose is agnostic to the mechanism adopted for partitioning. Nevertheless, in this work we employ two different ways to determine partitions as we will show in the following sections.

Example 3.5. *Consider Figure 3.1: D_7 is a description based on partitions $P_1 = \{t_1, t_6\}$ and $P_2 = \{t_2, t_3, t_4, t_5, t_7, t_8, t_9\}$, while D_5 is over the partitions $P_1 = \{t_4, t_7\}$, $P_2 = \{t_1, t_3, t_6\}$, and $P_3 = \{t_2, t_5, t_8, t_9\}$.*

Finally, given an input dataset I, the goal of our approach is to generate the *best* descriptions representing I. The quality of a description is supported by a *scoring function*.

Definition 4 (Scoring function)**.** Given a set of descriptions $\{D_1, \ldots, D_n\}$, a scoring function *Cost* is such that $\forall i$, $\text{Cost}(D_i) \geq 0$, and for each pair D_i, D_j, if $\text{Cost}(D_i) \leq \text{Cost}(D_j)$ then D_i is more relevant than D_j.

In practice, the function *Cost* may take into consideration several factors, such as user preferences, and can be personalized depending on the domain. We will define the full list of user preferences in Section 3.5 and the implementation of our scoring function in Section 3.6.3. Now we show that the general problem of generating the best description for a dataset I is NP-Hard.

Problem 1 (Finding the best description)**.** *Given a dataset I organized into ρ partitions, and a PTIME-computable scoring function Cost, let $\mathcal{D} = \{D_1, D_2, ..., D_m\}$ be the set of all possible descriptions over I. The computation of the most relevant description in $\mathcal{D}$ aims at returning the description $D_i \in \mathcal{D}$ such that $Cost(D_i) \leq Cost(D_j), \forall D_j \in \mathcal{D}$.*

Theorem 5. *Problem 1 is NP-Hard.*

Proof. We show that there exists a polynomial-time reduction from the Weighted Set Cover (WSC) Problem to Problem 1. The WSC Problem, that we recall next, is a well-known NP-Hard problem [CLRS09].

Problem 2 (Weighted set cover problem)**.** *Given a universe of n elements $\mathcal{U} = \{e_1, \ldots, e_n\}$, a set of m non-empty subsets of $\mathcal{U}$, $\mathcal{S} = \{s_1, \ldots, s_m\}$, and a PTIME cost function $c : \mathcal{S} \to \mathbb{R}^+$. The goal is to find the minimum cost subset $x \subseteq \mathcal{S}$ such that:*

1. *all elements are covered by x (i.e. $\forall e_i \in \mathcal{U}, e_i \in \bigcup_{s_j \in x} s_j$), and*
2. *the sum of the costs of the elements in x is minimized.*

Given an instance $(\mathcal{U}, \mathcal{S}, c)$ of Problem 2, let us define a mapping reduction r that creates a dataset $I = \{t_1, t_2, \ldots, t_n\}$ over a set of attributes $A = \{a_1, a_2, \ldots, a_m\}$:

- using a bijective function $g : \mathcal{U} \to I$ that associates each element e_i of the universe to a tuple t_i of I;

- using a bijective function $f : \mathcal{S} \to A$ that associates each s_j to an attribute a_j;
- each value $v_{i,j}$ for the i-th tuple and j-th attribute is such that:

$$v_{i,j} = \begin{cases} \top, & \text{if } g^{-1}(t_i) \in f^{-1}(a_j) \\ \bot, & \text{otherwise.} \end{cases}$$

It follows that each possible description $D_k \in \mathcal{D}$ over I is constructed out of boolean predicates in the form $p : (a_i = \top)$ or $p : (a_i = \bot)$. Additionally, r introduces a scoring function Cost$'$ that sums over the costs of each predicate p_i in a description D_k, i.e., $\text{Cost}'(D_k) = \sum_{p_i \in D_k} \text{Cost-p}(p_i)$, where:

$$\text{Cost-p}(p_i : (a_i = v)) = \begin{cases} c(f^{-1}(a_i)), & \text{if } v = \top \\ \infty, & \text{otherwise.} \end{cases}$$

Finally, let us also introduce a function h that transforms a d-formula with boolean predicates computed over an instance created by r into a candidate solution to Problem 2:

$$h(D_k) = \{s_k : s_k = f^{-1}(a_j), \forall (a_j = \top) \in D_k\}.$$

It is straightforward to see that with r and h we have that $\text{Cost}'(D_k) = c(h(D_k))\ \forall D_k \in \mathcal{D}$, where $\mathcal{D}$ is the set of all possible descriptions over I.

We can now solve Problem 1 over dataset I considering all the tuples grouped into the same partition[1] (i.e., $\rho = 1$). The solution is a description D formed by one d-formula d. By construction all predicates in d are in the form $p : (a_i = \top)$ and $d\ (D)$ is true for each tuple $t_i \in I$ (full coverage). We want to show that if the solution D has a finite cost, then $X = h(D)$ is the optimal solution for Problem 2. Otherwise, there is no solution for Problem 2. This can be proved by considering the two clauses of Problem 2 separately. If a finite solution to Problem 1 exists, then the first clause is true since (1) all the predicates in d are positive (i.e. $a_i = \top$), and (2) $\nexists t_i \in I$ such that D is false for t_i. Consequently, all the tuples have at least one value that is true and it is captured by the description D. By construction of r, it follows that $\nexists e_i = g^{-1}(t_i)$ such that $e_i \in X$.

[1] Note that this is just a simplification: similar arguments hold for the cases where $\rho > 1$.

We now need to show the clause of minimality. Let us assume, by contradiction, that another solution X' is optimal for the instance $(\mathcal{U}, \mathcal{S}, c)$ of Problem 2 (i.e. $c(X') < c(X)$). This means that there must exist a description D' over I such that $X' = h(D')$ and $\text{Cost}'(D') < \text{Cost}'(D)$. Unfortunately, this is impossible because we assumed that D is the minimal solution to Problem 1. Hence X must be minimal. Since (1) Problem 2 is NP-Hard, (2) the reduction r is polynomial (i.e., $\mathcal{O}(|\mathcal{U}| \times |\mathcal{S}|)$), (3) Cost' is in PTIME, and (4) the transformation h for generating D from X has linear time-complexity (i.e., $\mathcal{O}(|\mathcal{S}|)$), we can conclude that generating the best description D is also NP-Hard.

□

3.5 Generating Descriptions: A principled approach

Attributes of a dataset carry different degrees of relevance for users interested in gleaning insights from data. Some attributes have intrinsic value because, for example, they can identify entities in a domain (e.g., the attribute SensorID in the SENSOR dataset). Sometimes, the relevant attributes are indicated by the users due to their domain knowledge or to a specific interest. For example, a user may be interested in understanding when a specific attribute, e.g., the Temperature, assumes certain values in a sensor. For this reason, it is more convenient to see the dataset SENSOR partitioned by SensorID instead of a monolithic entity. *The first driving principle states that it is more explicative to think of a dataset as the composition of different groups of related tuples.* We follow this principle by letting the user specifying how the dataset has to be partitioned, and we associate a d-formula to describe each partition in the dataset. The set of d-formulas is the final description for the dataset.

Data explanation is conducted for different purposes. In some cases, users want an accurate and complete, yet readable, representation of the whole dataset. In other cases, a general profile of the dataset is just enough for the user. The description represents, in these latter cases, an overview that ignores infrequent values. Often, instead, there are users that are interested in finding outliers. *The second driving principle is that we can accommodate the multi-faceted goals of data explanation by relaxing the concept of full coverage in the descriptions (*cf *Definition 2)* [2]. By allowing users to interactively change the coverage of the expected descriptions (intended as the percentage of the total

[2] Note that this relaxation does not invalidate our formalization of Section 2.

number of tuples that are true for the description) they are now capable of extracting proper insights from data. For example, let us imagine a user that wants to separately analyze each sensor in the SENSOR dataset. Initially, she sets a high coverage to get a better picture and she gets that Humidity varies between 0.3 and 0.4 for SensorID 1, it is 0.5 in SensorID 2 and varies between 0.4 and 0.5 for SensorID 3. She then decides to dig deeper by lowering the coverage, and she finds that sensor 3 has Temperature equal to 100, which is an outlier.

We realize that is clearly not possible to define a single and global way to assess how good a description is since the quality is influenced by the subjectivity of the end users. Just as an example, a prolix description can be suitable for an user but, at the same time, be poor for another one. *The third driving principle is to consider user preferences for qualifying descriptions.* We follow this principle by: (1) defining a series of dimensions that characterize the descriptions, (2) we let the users indicate their preferential value for these dimensions, and (3) we take into account the user's preferences in the scoring and ranking of the final descriptions (Section 3.6.3).

Specifically, we have identified three dimensions for characterizing the quality of a description D from a user perspective: *coverage*, *degree*, and *diversity*.

Definition 6 (Coverage)**.** Given a subset X of the dataset I for which a d-formula is true, the coverage of a d-formula is the fraction $\frac{|X|}{|I_\rho|}$ (i.e., the percentage of I over the partition ρ for which the d-formula is true). Given a description D, the coverage of D, i.e. $cov(D)$, is the mean of the coverage of the d-formulas therein.

Basically, broad dataset descriptions have a high coverage; descriptions of datasets' peculiarities (e.g., outliers) have a low coverage. We will use this definition of coverage in our system to validate descriptions instead of the (more strict) one of Definition 2.

Definition 7 (Degree)**.** Given a description D over a dataset I with A attributes and composed by ρ partitions, let $A_{d_i} \subseteq A$ be the set of attributes in a d-formula d_i of D. The degree of a description is the average number of attributes in its constituting d-formulas: $deg(D) = \frac{1}{\rho} \sum_{d_i \in D} \frac{|A_{d_i}|}{|A|}$.

Intuitively, too many attributes makes a long and hard to read description. Conversely, a description is more compact when the number of attributes therein is small.

Definition 8 (Diversity)**.** Given a description D, let $||D||$ be the overall number of predicates in D and $A_D \subseteq A$ be the set of attributes that appear in the predicates of D at least once. The diversity is measured as: $div(D) = \frac{|A_D|}{||D||}$.

The *diversity* measures how often attributes are shared across d-formulas. While repetitions of a set of attributes in different partitions increase readability, they could reduce the fine-grained representation of partitions.

Example 3.6. *With reference to the descriptions reported in Figure 3.1,* $deg(D_1) = 1$, $div(D_1) = 1$; $deg(D_3) = 3/5$, $div(D_3) = 1$; $deg(D_5) = 6/15$, *and* $div(D_5) = 2/6$. *All these descriptions have the maximal coverage* ($cov(D_1) = cov(D_3) = cov(D_5) = 1$) .

We identified these three dimensions as the minimal set of parameters that allows us to both (1) extract enough information from users for enabling task-specific explorations, and (2) allow the system to return high-quality descriptions. We identified these set of dimensions through experimental evaluation and by reviewing similar approaches (e.g., [JGP16, LBL16]).

3.6 The Approach

The description generation process is interactive: users can specify their preference parameters repeatedly until the explanation need is fulfilled. The preference parameters are in Table 3.2 and include, among the others, the dimensions previously defined.

Parameter	Meaning
COV	The desired coverage. It is a float value between 0.0 and 1.0.
DEG	The desired degree. It is a float value between 0.0 and 1.0, where 1.0 stands for high-degree descriptions.
DIV	The desired diversity. It is a float value between 0.0 and 1.0, where 1.0 stands for high-diverse descriptions.
A_x	In the *user-driven* mode, the attributes of interest for the user, and $\|A_x\|$ the size of the description.
ρ	In the *data-driven* mode, the number of clusters to be generated.
k	The maximum number of descriptions to be returned.

Table 3.2: Users' preferences used as input to the framework.

Users interact with the system in two different modes. The *user-driven* mode is useful when the task is to profile data over some feature of interest; this can be obtained by pinpointing the set of pivotal attributes A_x. For example, D_5 of Figure 3.1 is a user-driven description on attribute Humidity, and therefore it determines the partitions $P_1 = \{t_4, t_7\}$, $P_2 = \{t_1, t_3, t_6\}$, and $P_3 = \{t_2, t_5, t_8, t_9\}$. The *data-driven* mode is instead more suited for cases where a user has little knowledge of the dataset, and the partitions are created in a fully-automated fashion via a clustering algorithm. D_7 in Figure 3.1 is a data-driven description based on partitions $P_1 = \{t_1, t_6\}$ and $P_2 = \{t_2, t_3, t_4, t_5, t_7, t_8, t_9\}$.

The generation of the descriptions is performed over 3 phases, as shown in Algorithm 1. In the first phase (line 1) the input dataset is partitioned. Recall that a partition is a set of tuples to be described together by the same d-formula. Data partitioning is detailed in Section 3.6.1. The generation of the d-formulas happens during the second phase (line 2) [3]. For each partition, the number of possible d-formulas is exponential over the number of attributes' values. Generating all possible d-formulas is, therefore, prohibitively expensive. As explained in Section 3.6.2, we adopt a heuristic procedure that allows us to prune d-formulas that are less relevant for the task at hand.

In the last phase (line 3), the actual descriptions are computed by combining d-formulas of different partitions. Intuitively, we assemble the top-k descriptions that maximize the scoring function from the generated d-formulas. This problem can be solved with a dynamic programming approach: in Section 3.6.3 we will employ a variant of the Viterbi Algorithm called *LVA* [SS94] (a.k.a. *List Viterbi*), although any other algorithm in this class can be used.

Algorithm 1 Generation of Descriptions

Input: The dataset I, A_x, COV, DEG, DIV, k, ρ and CONC.
Output: The top-k descriptions D_k.
1: $P \leftarrow CreatePartitions(I, A_x, \rho)$
2: $DF \leftarrow GetDFormulas(P, \text{COV}, \text{DEG})$
3: $D_k \leftarrow ViterbiTopK(DF, \text{DEG}, \text{DIV}, k)$
4: **return** D_k

[3] Note that when numerical attributes exists in a dataset, a discretization algorithm is applied during the second phase to extract categorical features from numerical attributes (e.g., [GLS+13]). An inverse process is applied to the descriptions before returning them to the user. Descriptions with d-formulas based on categorical attributes generated by discretization, in a post-process phase are transformed by interpolation over range predicates (Definition 1).

3.6.1 Building partitions

The first phase of our technique splits the input dataset into partitions. Since partitions will be described separately, it is desirable to create them such that similar tuples are grouped together. In *user-driven* mode, we create a partition for every distinct value of the projection of I over a non-empty set of user provided attributes $A_x \subseteq A$. In this case, the building procedure applies a SQL's `group-by` operator over the pivotal attributes specified by the A_x parameter.

Example 3.7. *Looking at Figure 3.1, D_4 is a user-driven description in which the user pinpointed* SensorID*; D_5 is another user-driven description but built around* Humidity*.*

In *data-driven* mode, the partitioning logic ensues from the application of a clustering algorithm. In our implementation, we used *k-means* and we therefore let the user specify the number of clusters (i.e. the ρ parameter). However, any clustering algorithm can be used for the task.

Example 3.8. *In Figure 3.1 D_7 is a data-driven description composed of two partitions.*

3.6.2 Building d-formulas

In the second phase, we build all feasible and relevant d-formulas for each partition. The number of candidate predicates depends on the size of the active domain of the attributes in the partition. Therefore, the complexity of this phase is $\mathcal{O}\Big(\prod_{a \in A}(2^{|adom_I(a)|})\Big)$, where A is the set of the attributes and $|adom_I(a)|$ is the aforementioned size. Given the large complexity for generating all possible d-formulas, we follow a heuristic process that generates the most relevant candidate d-formulas only. Currently, we adopt two heuristics, one for pruning prolix d-formulas, and one for pruning selective d-formulas and predicates.

Heuristic 1 - pruning prolix d-formulas

Descriptions are consumed by human users. A description with a large number of predicates is somehow hard to read. Intuitively, when a user asks for descriptions with low degree she is declaring that a d-formula with a small number of predicates is preferable.

Conversely, high degree values are specified by users that prefers descriptions with wide d-formulas. We can therefore push the degree intent of the users into the process of building d-formulas so that we can early-prune d-formulas that do not meet DEG requirements. This limits the number of predicates evaluated and therefore improves the efficiency of the approach.

We implement Heuristic 1 through the notion of *conciseness* (parameterized with CONC): i.e., a threshold specifying the maximum number of possible atomic predicates allowed in a d-formula. More precisely, CONC and DEG are connected as follow:

$$\text{CONC} = e^{\lambda \ * \ \text{DEG}}. \tag{3.1}$$

The exponential function models the perception of the user according to which small variations in the number of predicates of a low-degree description have a significant impact on the legibility of the description. In the following we fix $\lambda = 6$ to limit the number of atomic predicates in a d-formula to 400 when DEG assumes the maximum level.

Heuristic 2 - pruning selective d-formulas and predicates

The desired coverage COV imposes the percentage of tuples the d-formulas should reach to be part of a description. Heuristic 2 transforms the COV discrete value into an interval of admissible values whose width is dependent on the coverage itself. This reflects the needs of users who selecting a small coverage are looking for outliers, thus aiming to retrieve descriptions in a small coverage interval width centered on COV. Conversely, high COV generates a large interval width, since in this case a user is looking for a broad descriptions of a dataset. Heuristics 2 captures this behavior: the width of the interval of admissible coverage for d-formulas is proportional to the coverage itself:

$$\textit{offset}(\text{COV}) = \alpha \ * \text{COV}^2 \tag{3.2}$$

We have experimentally determined that $\alpha = 0.14$ is the value that maximizes the quality metrics (see Section 3.7). The *offset* of Equation 3.2 determines the interval $[\text{COV}_l, \text{COV}_u]$ of desirable coverage, whereby $\text{COV}_l = \text{COV} - \textit{offset}(\text{COV})$ and $\text{COV}_u = \text{COV} + \textit{offset}(\text{COV})$.

In our approach we generalize the above intuition over predicates so that we can early prune predicates (before they compose into d-formulas) if they do not meet the expected coverage. A naive way to early identify admissible predicates is to check if their coverage is within the interval $[0, \text{COV}]$. While this condition has the ability to keep relevant predicates only, it might still not be very effective as pruning filter. It would generate d-formulas composed of predicates with largely different coverage levels, that, for this reason, could not to be interesting for the users. For example, if a user wants descriptions with coverage equal to 0.8, a d-formula composed of a large coverage predicate (e.g., it applies to 79% of tuples) and a number of small coverage predicates is not really meaningful (altough correct). A first idea to solve this issue could be to use the same interval $[\text{COV}_l, \text{COV}_u]$ adopted for filtering d-formulas. Nevertheless, discarding predicates with coverage out of this interval limits the d-formulas computed to the combinations of those predicates with coverage close to the COV value only. This appears to be too selective and some data distributions can generate a low number of d-formulas (or even no d-formula). We therefore enlarge the interval width by using $\text{COV}_p = \frac{\text{COV} - \mathit{offset}(\text{COV})}{\text{CONC}}$ instead of COV_l as left bound. This makes the coverage level depending on the conciseness, so that when CONC is high (wider descriptions are required), the interval width increases.

Algorithm 2 `GetDFormulas`

Input: A list of partitions P, the dataset attributes A, COV and CONC.
Output: The d-formulas $\mathbf{d}$.

1: $\mathbf{d} \leftarrow \varnothing$
2: $\text{CONC} = e^{\lambda \ *\text{DEG}}$
3: $\text{COV}_l \leftarrow \text{COV} - \mathit{offset}(\text{COV})$
4: $\text{COV}_u \leftarrow \text{COV} + \mathit{offset}(\text{COV})$
5: $\text{COV}_p \leftarrow \frac{\text{COV}_l}{\text{CONC}}$
6: **for each** $p \in P$ **do**
7: $\quad \Theta \leftarrow \varnothing$
8: $\quad$ **for each** $a \in A$ **do**
9: $\quad\quad$ **for each** $v \in dom(a)$ **do**
10: $\quad\quad\quad$ **if** $\frac{|\sigma_{a=v}(p)|}{|p|} \in [\text{COV}_p, \text{COV}_u]$ **then**
11: $\quad\quad\quad\quad \Theta \leftarrow \Theta \cup \{a = v\}$
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end for**
14: $\quad$ **end for**
15: $\quad \mathbf{d} \leftarrow \mathbf{d} \cup \texttt{ValidDFormulas}(\Theta, \text{CONC}, \text{COV}_l, \text{COV}_u, p)$
16: **end for**
17: **return** $\mathbf{d}$

Algorithm 2 shows the routine for generating all possible and relevant d-formulas. It

takes in input the list of partitions (P), the list of dataset attributes (A), the coverage (COV) and the degree (DEG). We initialize the set $\mathbf{d}$ that will contain the resulting d-formulas (line 1), the conciseness (line 2) as suggested by Heuristic 1 and the desirable coverage intervals (lines 3-5) derived using Equation 3.2 introduced by Heuristic 2. The d-formulas of each partition (i.e. Θ) are computed separately (lines 6-11). We generate the "atomic predicates" (i.e. $(a = v)$) for each distinct value v of a in the partition p (line 11). Atomic predicates are generated only if they are within the expected coverage (i.e., $[\text{COV}_p, \text{COV}_u]$) as defined by Heuristic 2 (line 10). Atomic predicates are combined together to generate conjunctions of predicates (line 12). The resulting d-formulas, for each partition, are then returned as output (line 13).

The `ValidDFormulas` procedure of Algorithm 3 combines together predicates and returns only those combinations (i.e., d-formulas) that are valid. The combination of the input predicates Θ is organized in a lattice. The main loop of lines 2-12 dynamically generates the lattice. In each iteration, we generate one level of the lattice (the n-th) by combining pairs of the previously generated predicates (lines 4-10). One predicate might get combined with another predicate many times. For this reason, we use the function `GetDistinctPredicates` that efficiently select a combinable predicate only once (line 5). Pairs of predicates are combined with the operator $\oplus$ (line 7). The combination is a d-formula that is either: (1) a new predicate created from the union of two predicates on the same attribute (e.g., $\{a = 4\} \oplus \{a = 5\} = \{a \in \{4, 5\}\}$), or (2) a conjunction of predicates on different attributes (e.g., $\{a \in \{4, 5\}\} \oplus \{b \in \{5, 6\}\} = \{a \in \{4, 5\} \ \wedge b \in \{5, 6\}\}$). We keep only combinations of predicates in the desired interval of coverage values $[\text{COV}_l, \text{COV}_u]$. $\sigma_d(p)$ are the tuples covered by the d-formula d in partition p. Only combinations of predicates whose coverage is within the interval (line 8) are kept for the next level of the lattice (line 9 and line 11) and in the list of final results Θ^* (line 10). The generation of the lattice ends when the threshold, as per Heuristic 1, is reached ($n = \text{CONC}$) or when it is no longer possible to generate predicates ($|\Theta| = 0$).

3.6.3 Building top-k descriptions

Selecting the best d-formulas in isolation does not necessarily lead to the best descriptions. We need to search for the optimal set of d-formulas across partitions that all together minimize the cost. Given the complexity of the problem, we use dynamic

programming. Our current implementation is based on *LVA* [SS94], which is a generalization of Viterbi Algorithm [For73] for top-k solutions. The complexity of List Viterbi is $\mathcal{O}(\rho \times k \times d^2)$, where ρ is the number of partitions, k is the number of top elements to retrieve and d is the number of d-formulas. Viterbi Algorithm and its variants require to model the search space as a trellis. We construct the trellis in the following way. We build a vertical slice for each partition. The nodes in the slice are the d-formulas we have found in the previous phase. Nodes of a slice are connected to all the nodes of the next slice via weighted directed edges. In this way, a node has an incoming path from each node of the previous slice. A path represents a list of d-formulas, with at most one d-formula for partition. An example of our trellis is in Figure 3.2. The algorithm is used to find the best full path, that is a path that starts in the first slice and terminates in the last slice of the trellis. Note that, for the final results, the order of the slices is irrelevant since all nodes of a slice are connected to all the nodes of the next slice.

Algorithm 3 `ValidDFormulas`

Input: A set of predicates Θ, CONC, COV_l, COV_u, and a partition p.
Output: The d-formulas with coverage between COV_l and COV_u for partition p.

$\Theta^* \leftarrow \Theta$
$n \leftarrow 0$
repeat
  $\Theta' \leftarrow \varnothing$
  for each $\theta_i \in \Theta$ **do**
    $\Theta_i \leftarrow$ `GetDistinctPredicates`(θ_i, Θ)
    for each $\theta_j \in \Theta_i$ **do**
      $d \leftarrow \theta_i \oplus \theta_j$
      if $\frac{|\sigma_d(p)|}{|p|} \in [\text{COV}_l, \text{COV}_u]$ **then**
        $\Theta' \leftarrow \Theta' \cup d$
        $\Theta^* \leftarrow \Theta^* \cup d$
      end if
    end for
  end for
  $\Theta \leftarrow \Theta'$

until $|\Theta| = 0$ **or** $n =$ CONC
return Θ^*

Viterbi needs an objective function w to score a path, which intuitively represents the score of the (intermediate) descriptions we are computing. Given a trellis composed of N nodes and S slices, the score of a path terminating in a node j of a slice s of the trellis (i.e. $w_j(s)$) can be recursively computed. Therefore, the best description D^* is the one maximizing the score of the node in the final slice S:

$$Cost(D^*) = \max_{1 \leq i \leq N} [w_i(S)] \tag{3.3}$$

The description D^* is constructed by backtracking the computation of Equation 3.3, which determines the nodes in the corresponding path. *LVA* generalizes Equation 3.3 to compute the top-k solutions. We devise the function w used to (recursively) score a path as:

$$w_j(s) = \begin{cases} \max_{1 \leq i \leq N} [w_i(s-1) \ * L_{ij} * \Delta_{div}(i,j)] * \Delta_{deg}(j) * \ (1 - H(j)), & \text{if } s > 1 \\ \Delta_{deg}(j) * \ (1 - H(j)), & \text{if } s = 1 \end{cases} \tag{3.4}$$

L is the adjacency matrix of the trellis, where $L_{ij} = 1$ if there is an edge between node i and node j, otherwise $L_{ij} = 0$. Δ_{div} and Δ_{deg} score diversity and degree in terms of adherence to the preferences of the user DIV and DEG, respectively. More specifically, Δ_{deg} is defined as $\Delta_{deg}(d) = h(deg(d))$, where h is the Normal Distribution function centered on DEG and with variance equal to 0.2. The variance is chosen experimentally. In this way, we give a boost to descriptions whose degree is close to DEG. Instead, Δ_{div} considers a pair of d-formulas d_i and d_j, out of which we compute the partial description D' that includes both of them. Then, analogously to the degree factor, the diversity factor is defined as $\Delta_{div}(d_i, d_j) = g(div(D'))$, where g is the Normal Distribution function centered on DIV with variance equal to 0.2. $H(j)$ is a measure of *entropy* for evaluating how discriminative attributes are. It is computed as the normalized *Shannon entropy* value[4] of the attributes used by the d-formula associated to node j.

Intuitively, the scoring function measures (1) the adherence between the features of the description and the expectation of the user, and (2) the ability of the attributes to describe a particular partition. Note that while DEG is required in Equation 3.4 because only indirectly enforced by CONC in Heuristic 1 we can omit COV from Equation 3.4 because Heuristic 2 makes sure that only descriptions with the proper coverage are computed.

Example 3.9. *Let us consider the dataset* SENSOR *described in Table 3.1 and a user interested in descriptions explaining the behavior of each sensor (i.e.,* $A_x = \{$`SensorID`$\}$*).*

[4] We use the standard definition of normalized Shannon entropy for a probabilistic variable X in a finite domain $\{x_i\}$, that is $H(X) = \frac{-\sum_{i=1}^{N} P(x_i) * \log_2 P(x_i)}{\log_2 N}$.

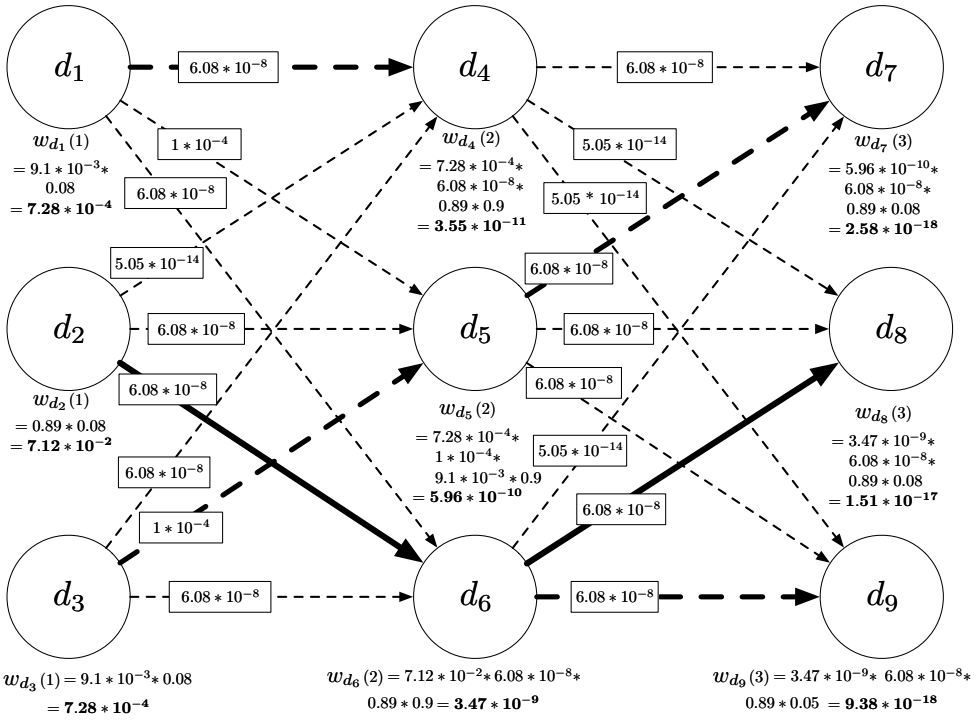


Figure 3.2: The Viterbi Algorithm applied to SENSOR.

According to our partitioning mechanism (see Section 3.6.1), the dataset is divided into three partitions, one for each sensor. The application of the techniques described in Section 3.6.2 generates a large number of d-formulas. Some of them are shown in Table 3.3 along with their Δ_{deg} *and entropy values that are used by* LVA *for computing the descriptions. Figure 3.2 shows the application of the Viterbi to the d-formulas in Table 3.3. The trellis is divided into three vertical slices corresponding, from the left-side to the right-side of the figure, to the partitions for* `SensorID = 1`, `SensorID = 2` *and* `SensorID = 3`. *Each node represents one of the d-formulas. The weights assigned to the nodes have been computed according to Equation 3.4. Note that nodes in the first slice are computed following the formula of the initialization weight; while for each node in the other slices, only one incoming edge (the one maximizing the weight) is kept.* Δ_{deg} *values (shown in Table 3.3) and* Δ_{div} *values (shown in Figure 3.2 on the edges) have been computed taking into account the users' preferences, that is for high coverage, low degree and low diversity. Once the trellis is completed, the last slice is analyzed and the top-k nodes maximizing the weight are selected. The highest ranked path is* $d_2 \rightarrow d_6 \rightarrow d_8$ *with a score of* $1.51E-17$. *This description corresponds to the fourth row of Table 3.4.*

Table 3.4 shows some descriptions generated by varying users' preferences. The first three descriptions describe low coverage settings. In the first case, the attribute Voltage

SensorID =1	SensorID =2	SensorID =3
d_1 = Te ∈ {35} ∧ H ∈ {0.3} Δ_{deg} = 0.0091 H = 0.92	d_4 = H ∈ {0.5} Δ_{deg} = 0.89 H = 0.1	d_7 = H ∈ {0.4} Δ_{deg} = 0.89 H = 0.92
d_2 = Te ∈ {35} Δ_{deg} = 0.89 H = 0.92	d_5 = Te ∈ {35} ∧ H ∈ {0.5} Δ_{deg} = 0.0091 H = 0.1	d_8 = Te ∈ {35} Δ_{deg} = 0.89 H = 0.92
d_3 = Te ∈ {34,35} ∧ H ∈ {0.3,0.4} Δ_{deg} = 0.0091 H = 0.92	d_6 = Te ∈ {35} Δ_{deg} = 0.89 H = 0.1	d_9 = Te ∈ {35, 100} Δ_{deg} = 0.89 H = 0.95

Table 3.3: A subset of the d-formulas generated from SENSOR.

COV	DEG	DIV	Description
0.2 (LOW)	0.1 (LOW)	0.1 (LOW)	({S = 1 ∧ V ∈ {2.6} (33.33%)}) ∨ ({S = 2 ∧ V ∈ {2.6} (33.33%)}) ∨ ({S = 3 ∧ V ∈ {2.6} (33.33%)})
0.2 (LOW)	0.1 (LOW)	0.9 (HIGH)	({S = 1 ∧ H ∈ {0.4} (33.33%)}) ∨ ({S = 2 ∧ V ∈ {2.6} (33.33%)}) ∨ ({S = 3 ∧ Te ∈ {100} (33.33%)})
0.2 (LOW)	0.9 (HIGH)	0.1/0.9 (LOW/ HIGH)	({S = 1 ∧ Te ∈ {34} ∧ Ti ∈ {11} ∧ V ∈ {2.6} ∧ H ∈ {0.4} (33.33%)}) ∨ ({S = 2 ∧ Ti ∈ {11} ∧ V ∈ {2.6} (33.33%)}) ∨ ({S = 3 ∧ Te ∈ {100} ∧ Ti ∈ {12} (33.33%)})
0.8 (HIGH)	0.1 (LOW)	0.1 (LOW)	({S = 1 ∧ Te ∈ {35} (66.67%)}) ∨ ({S = 2 ∧ Te ∈ {35} (100.0%)}) ∨ ({S = 3 ∧ Te ∈ {35} (66.67%)})
0.8 (HIGH)	0.9 (HIGH)	0.1/0.9 (LOW/ HIGH)	({S = 1 ∧ H ∈ {0.3} ∧ Ti ∈ {1, 12} ∧ V ∈ {2.7} ∧ Te ∈ {35} (66.67%)}) ∨ ({S = 2 ∧ H ∈ {0.5} ∧ Ti ∈ {1, 12} ∧ V ∈ {2.7} ∧ Te ∈{35} (66.67%)}) ∨ ({S = 3 ∧ H ∈ {0.5,0.4} ∧ Ti ∈ {1} ∧ V ∈ {2.3, 2.6} ∧ Te ∈ {35} (66.67%)})
0.8 (HIGH)	0.1 (LOW)	0.9 (HIGH)	({S = 1 ∧ V ∈ {2.7} (66.67%)}) ∨ ({S = 2 ∧ H ∈ {0.5} (100.0%)}) ∨ ({S = 3 ∧ Te ∈ {35.0} (66.67%)})

Table 3.4: Descriptions with different users' preferences.

has been used for all the d-formulas since the desired diversity level is low. In the second case, three different attributes are used for describing the sensors. In the third example, a description with a high degree is shown. Due to the low number of attributes in the dataset, we obtain the same description when the user selects low and high diversity. The other descriptions have high coverage and different levels of degree and diversity. The actual coverage of the d-formulas in the corresponding partition is enclosed between

	SENSOR	CRIME	MUSHROOM	SENSORRANDOM
Max Gini Index	0.86	0.86	0.63	0.85
Min Gini Index	0.75	0	0.62	0.83
Avg Gini Index	0.79	0.65	0.63	0.85
Std of Gini Indexes	0.025	0.159	0.001	0.005

Table 3.5: Dataset heterogeneity.

parentheses. The low cardinality of the example dataset (9 tuples) makes it necessary to map the COV *into a large interval.*

3.7 Experimental evaluation

In this section we evaluate our approach over *efficiency* (Section 3.7.1), *objective effectiveness* using a list of quantitative measures (Section 3.7.2), and *subjective effectiveness* using the judgment of the human-in-the-loop (Section 3.7.3). All the experiments are performed on a machine running Ubuntu 12.04 with 16 processors, 128 GB of RAM and a 1 TB of storage. We implemented all algorithms using multi-threading in Python version 3.6.

Datasets. In the experiments we use three datasets with complementary characteristics. By default, we transform numeric attributes into categorical via data binning with 100 equal width bins. Section 3.7.1 illustrates how the number of bins impacts on performance. The *Sensor* dataset [5] has 2.3 million data points collected from 54 sensors deployed within the Intel Berkeley Research lab. Each point is a set of measurements (`light`, `humidity`, `temperature` and `voltage`) registered at an `epoch` and it is also stored with a timestamp formed by `year`, `month`, `day`, `hour`. The *Crime* dataset contains information about the crimes committed during the first quarter of 2018 in Chicago. It consists of 76k records and 9 attributes (`Month`, `Day`, `Hour`, `Block` – categorical attribute having 19800 distinct values, `Description` – categorical attribute having 266 distinct values, `Location Description` – having 119 distinct values, `Arrest` – boolean, `Domestic` – bolean, and `Primary Type` – categorical with 30 distinct values) taken from the Chicago Police Department's CLEAR database [6]. Finally, the *Mushroom* dataset [7]

[5] http://db.csail.mit.edu/labdata/labdata.html
[6] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2
[7] http://archive.ics.uci.edu/ml/datasets/Mushroom

contains data related to different species of mushrooms. It has 8k records with 23 categorical attributes representing morphological features of the mushrooms (`cap-shape`, `gill-attachment`, `veil-color`, etc).

Scenarios. We defined four test scenarios to highlight the behaviour of our approach in different and complementary use cases.

Scenario 1 – SENSOR. This scenario evaluates the explanation for a large dataset (i.e. SENSOR) with a small number of (numeric) features partitioned on a large number of equi-sized classes. The dataset has 24 equi-sized partitions over `hour`.

Scenario 2 – CRIME. This scenario has the goal of evaluating the explanations in a large dataset (i.e. CRIME) with few features (both numeric and categorical) partitioned on a large number of unbalanced classes. CRIME has been partitioned on `Primary Type`, resulting in partitions of different size (the avg size is 2458, standard deviation is 4225).

Scenario 3 – MUSHROOM. This scenario aims at evaluating the ability of our approach to cope with a large number of features. We partitioned MUSHROOM in two equi-sized classes representing edible and poisonous mushrooms.

Scenario 4 – SENSORRANDOM. This scenario evaluates the robustness of the approach in a data-driven mode with a bad partitioning. A synthetic attribute `Category` has been added to the SENSOR dataset and used to generate 24 random partitions, thus to simulates a "bad" partitioning.

To understand whether our approach is robust in presence of data with high dispersion, we computed the Gini Index on each attribute of the datasets with respect to the class used in the partitioning. The index is 1 with maximum dispersion and 0 with no dispersion. Table 3.5 shows the maximum, the minimum, the average and the standard deviation for the Gini Index values across the attributes of the datasets used in our four scenarios. MUSHROOM and CRIME scenarios contain the most homogeneous data distributions. Conversely, SENSORRANDOM has the most skewed distribution since the dataset has been randomly partitioned.

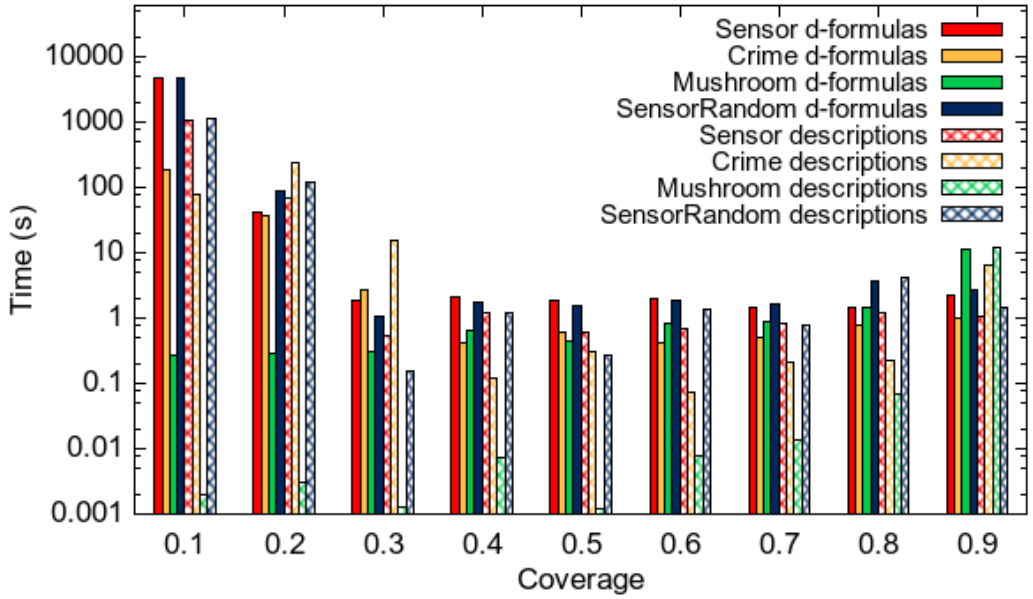


Figure 3.3: Runtime performance of the different scenarios.

3.7.1 Efficiency

In this section we evaluate the efficiency of generating d-formulas and combining them into descriptions.

Coverage ($\alpha = 0.14$)	Crime	Mushroom			Sensor			Sensor random
		$\alpha/2$	$3/4 \times \alpha$	α	B10	B100	B1000	
0.1 ± 0.0014	854	0	1	1	27958	29681	29899	29912
0.2 ± 0.0056	3500	2	4	5	939	982	3880	1569
0.3 ± 0.0126	446	7	10	12	50	52	194	24
0.4 ± 0.0224	45	12	15	20	45	94	34	70
0.5 ± 0.035	87	6	6	15	31	56	28	46
0.6 ± 0.0504	31	9	14	26	10	66	18	72
0.7 ± 0.0686	54	9	18	32	4	74	35	52
0.8 ± 0.0896	59	15	21	78	42	96	17	120
0.9 ± 0.1134	309	15	83	1439	359	90	72	72

Table 3.6: Number of d-formulas w.r.t. coverage.

We start with reporting in Table 3.6 the number of generated d-formulas with respect to the input coverage. All experiments are executed with a default of $\alpha = 0.14$ and with DEG = 0.25, which limits d-formulas to maximum 5 predicates. In Mushroom we also test the sensitivity on different α-settings (half and three fourths of the default value).

The number of generated d-formulas is higher when the coverage is low for Sensor, SensorRandom and Crime, and with high coverage for Mushroom. This is due to

the fact that the frequency distribution of *Crime* and *Sensor* datasets' attributes follow the Zipf's rule: the majority of values appear only few times. This results in many low-coverage descriptions. In the *Mushroom* dataset we have attributes with few values that repeat many times. Therefore, most of d-formulas are generated with these values at high coverage. In the results of the MUSHROOM scenario we can also see that the parameter α is useful to play around the user's preference for coverage but does not affect the efficiency much.

On SENSOR we show that the number of bins does not vary the number of generated d-formulas much. The large number of d-formulas generated for the coverage equal to 0.9 and 10 bins is motivated by the distribution of values that follow the pattern previously described for MUSHROOM.

In Figure 3.3, we show the time required to generate d-formulas and to compute the final descriptions at different coverage settings. The final execution time is the sum of the two components. In general we notice that descriptions are produced at interactive time (few seconds) in most of the cases. The harder cases to process are with low coverage because the system has to sift throw several thousands of descriptions. We can also see that the time for computing the descriptions is quadratic with the number of d-formulas, thus confirming the theoretical complexity of *LVA*.

The efficiency of the heuristics, taken in isolation or together, compared against the exhaustive case (i.e., No Heuristic) is in Figure 3.4 (for MUSHROOM scenario). For Heuristic 2, we consider different degree values and we report the conciseness (from 2 to 400) derived from the degree variations from 0.1 to 1. We observe that the approach requires up to 30k seconds, whereas the combined use of the heuristics makes the approach usable in interactive applications. The runtime performance are supported by the number of generated d-formulas in Table 3.7. The separate application of the heuristics is able to reduce the number of d-formulas up to 3 orders of magnitude, while the combined application reduces the number of d-formulas up to 5 orders of magnitude.

3.7.2 Quantitative evaluation of effectiveness

To assess the effectiveness of the descriptions generated by our approach, we evaluate them using a set of quality metrics along three perspectives: (1) *structural quality*, i.e.,

Cov	No Heur.	Heur. 1								Heur. 2	Heur. 1+2							
		c=2	c=5	c=10	c=25	c=50	c=100	c=250	c=400		c=2	c=5	c=10	c=25	c=50	c=100	c=250	c=400
0.2	14570	6944	14497	14570	14570	14570	14570	14570	14570	52	2	5	11	26	42	51	52	52
0.4	115762	18710	107087	115757	115762	115762	115762	115762	115762	258	8	20	37	124	167	249	256	258
0.6	664226	39385	499940	664205	664208	664219	664219	664226	664226	636	6	26	59	138	368	489	624	630
0.8	1431480	51451	856218	1431460	1431480	1431480	1431480	1431480	1431480	10091	28	78	188	702	2021	4635	9759	10007

Table 3.7: Impact of heuristics on the number d-formulas.

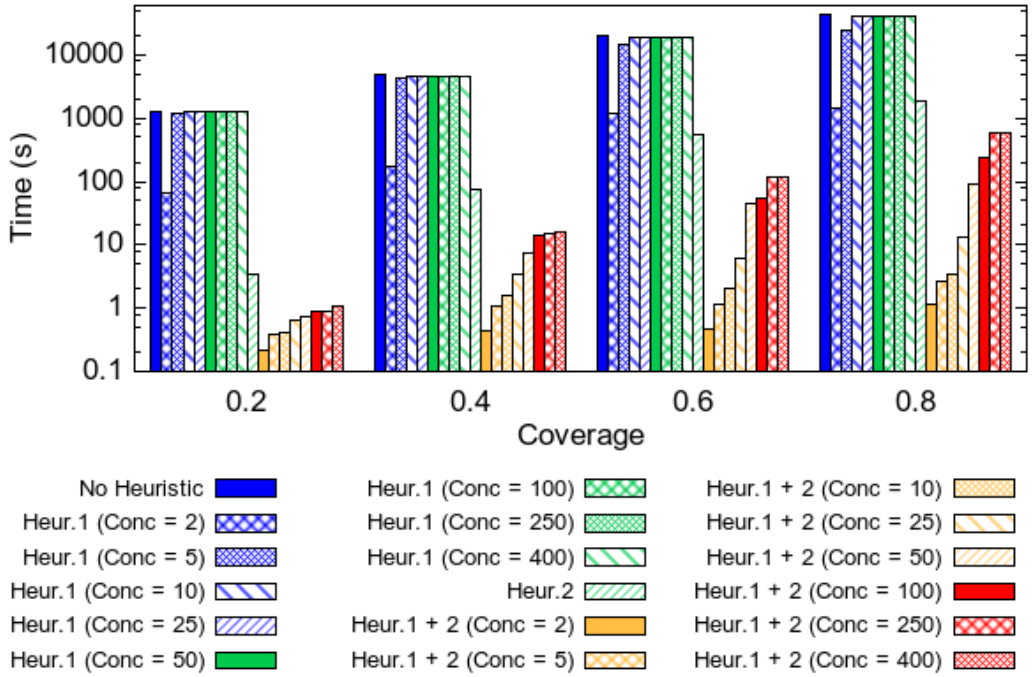


Figure 3.4: Efficiency of heuristics (MUSHROOM scenario).

characteristics of the descriptions independently of the use case; (2) *understandability*, i.e., "how easy it is to understand a description?"; and (3) expressiveness, i.e., "how many details are captured by a description?".

Structural metrics. We use *overlapping* and *precision* metrics for the structural quality. *Overlapping* measures the percentage of tuples in the dataset for which more than one d-formula hold:

$$overlap(D) = \frac{2}{\rho * (\rho - 1)} * \sum_{\substack{d_i, d_j \in D \\ i \leq j}} \frac{|I(d_i) \cap I(d_j)|}{|I|}$$

where $I(d_i)$ represents the set of tuples of the dataset I that are true for d_i. This measures how discriminative the d-formulas are when taken in isolation. Generic descriptions have high percentage of *overlap*, whereas peculiar description have small *overlapping*.

Orthogonally to *overlap*, we consider the *precision*. The higher the number of values used in d-formula predicates, the higher the probability that a d-formula describes combinations of values that are not existing in the actual data. For instance, the d-formula $(a \in \{\alpha, \beta, \gamma\} \wedge b \in \{1, 2\})$ describes three possible values for a and two for b. This is six combinations in total: $(a = \alpha \wedge b = 1)$, $(a = \alpha \wedge b = 2)$, etc. The *precision* checks how

many combinations correspond to tuples that actually exist in the dataset. For instance, let us suppose that no tuple is true for $(a \in \{\alpha\} \wedge b \in \{1\})$. The *precision* highlights that one out of six unfolded formulas is extra. This is measured with:

$$prec(D) = \frac{1}{\rho} * \sum_{d \in D} \frac{\sum_{u \in C^{\times}(d)} eval(u, d)}{|C^{\times}(d)|}$$

where: $C^{\times}(d)$ returns all the combinations by computing the cartesian product between the values of the atomic predicates in d; and $eval(u, d)$ is a boolean function that returns true if u holds on d, and false otherwise.

Understandability metric. A description is understandable when its predicates are discriminative. The *purity* quantifies the understandability by measuring the dispersion of the values of the attributes across the d-formulas. We rely on the complement of the Shannon Entropy H to compute the *purity*. The purity gets better (i.e. it tends to 1) when we select the most appropriate attributes to describe a set of tuples. The purity of a description is the complement of the weighted average of the entropy of its d-formulas:

$$purity(D) = 1 - \left[\frac{1}{\rho} * \sum_{d \in D} \left(\frac{1}{|A_d|} * \sum_{a \in A_d} H_a(d) \right) * \frac{|I(d)|}{|I|} \right]$$

with $H_a(d)$ being the Shannon entropy of the values of attribute a in the set of tuples described by the d-formula d.

Expressiveness metrics. We evaluate the expressiveness of descriptions with *average number of predicates* per d-formula and with the *number of distinct attributes* per description. The *average number of predicates* provides a measure of the complexity of d-formulas: in fact, d-formulas composed of many attributes are clearly hard to read. This difficulty can be mitigated by using the same attributes across d-formulas. In fact, a low *number of distinct attributes* per description is preferable in terms of expressiveness.

3.7.2.1 Benchmark Experiments

For each scenario, we set the coverage at pre-determined intervals, and we sweep over the degree and diversity (with steps of 0.1 points). For each combination, we compute the top-25 descriptions. In this way we simulate a wide range of possible inputs provided by

scenario	cov	purity	precision	overlap	# pred d-formula	distinct attrs
Crime (30 part.)	0.10	0.21 (0.01)	1.00	0.01 (0.01)	1.00	3.333 (0.47)
	0.30	0.32 (0.09)	1.00	0.03 (0.01)	1.00	5.000
	0.50	0.38 (0.08)	1.00	0.05 (0.01)	1.09 (0.08)	4.000
	0.70	0.39 (0.04)	1.00	0.06 (0.02)	1.07 (0.07)	3.803 (0.79)
	0.90	0.59 (0.24)	1.00	0.26 (0.07)	1.91 (0.70)	4.384 (1.62)
Mushroom (2 part.)	0.10	0.03	1.00	0.01 (0.01)	1.00	1.833 (0.37)
	0.30	0.16 (0.17)	1.00	0.06 (0.05)	1.00	1.822 (0.38)
	0.50	0.17 (0.19)	1.00	0.24 (0.08)	1.00	1.844 (0.36)
	0.70	0.42 (0.26)	0.94 (0.07)	0.34 (0.17)	1.40 (0.40)	2.299 (0.87)
	0.90	0.28 (0.27)	0.53 (0.35)	0.57 (0.20)	6.23 (3.06)	6.990 (3.12)
Sensor (24 part.)	0.10	0.10 (0.04)	1.00	0.04 (0.01)	1.00	4.000
	0.30	0.26 (0.02)	1.00	0.09 (0.01)	1.00	3.000
	0.50	0.29 (0.04)	1.00	0.22 (0.01)	1.00	2.444 (0.50)
	0.70	0.37	1.00	0.44 (0.06)	1.07 (0.07)	2.333 (0.47)
	0.90	0.58 (0.17)	0.98 (0.02)	0.72 (0.03)	1.78 (0.62)	2.679 (0.47)
SensorRandom (24 part.)	0.10	0.01	1.00	0.04 (0.01)	1.00	1.00
	0.30	0.02	1.00	0.31 (0.01)	1.00	1.00
	0.50	0.19	1.00	0.47 (0.01)	1.00	1.00
	0.70	0.24	1.00	0.63 (0.02)	1.00	1.836 (0.37)
	0.90	0.38 (0.21)	1.00	0.83 (0.03)	1.00	2.378 (0.58)

Table 3.8: Effectiveness of the different scenarios.

	rules/ part.	coverage	degree	diversity	purity	overlap	# pred d-formula	distinct attrs
Crime	161.97	99.95%	0.77	0.25	0.26	0.00	23.05	6.00
Mushroom	6.5	100%	0.2	0.93	0.19	0.00	5.00	9.00
Sensor	2635.13	48.84%	0.70	0.25	0.13	0.00	20.32	6.00
SensorRandom	6944.46	16.62%	0.79	0.24	0.07	0.00	26.46	6.00

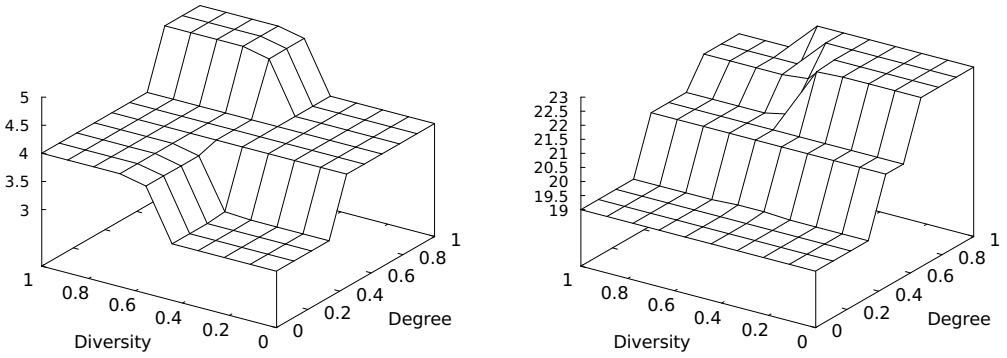
Table 3.9: Comparison with Decision Trees. Coverage in this case is the final accuracy of the generated model.

the users. The result is the computation of 3025 descriptions per coverage level, which are evaluated according to the aforementioned metrics. Table 3.8 shows the average values (and the standard deviations between parentheses when it differs from zero) of the descriptions of each series, grouped by coverage.

Analyzing the results, we first observe that when then coverage increases, the purity increases as well. This is due to Heuristic 2 that avoids the combination of predicates with different coverage levels. In fact, the data distributions in the scenarios follow the Zipf's law, i.e., there is a small number of values frequently repeated. The typical d-formula for a low level coverage is instead composed of predicates with rather infrequent values, thus delivering a lower purity. Furthermore, we can notice that the number of

	# rules	cov	degree	diversity	purity	overlap	# pred d-formula	distinct attrs
Decision sets [LBL16]	14	1	0.1	0.9	0.77	0.88	2.43	5.00
Our approach	2	1	0.1	0.9	0.75	0.77	2.00	4.00
			0.1	0.1	0.77	0.77	2.50	4.00
			0.9	0.1	0.38	0.61	3.50	5.00
			0.9	0.9	0.51	0.61	2.50	5.00

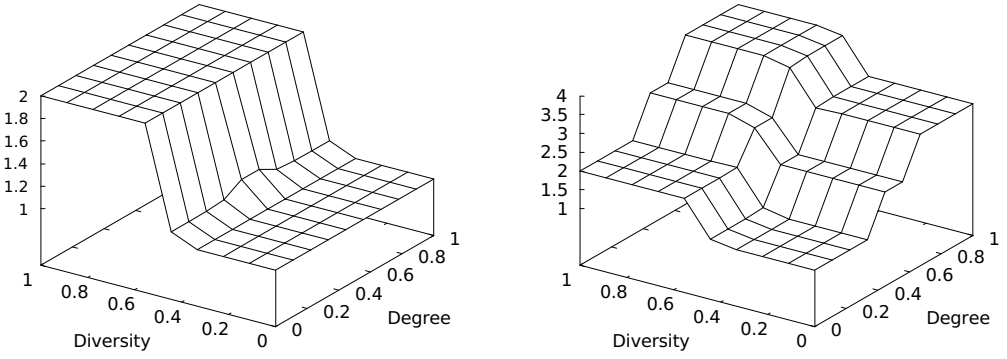
Table 3.10: Comparison with Decision Sets over MUSHROOM. The other scenarios timed-out before generating any description.



(a) CRIME: low COV - distinct attr.

(b) CRIME: low COV - # attr.

Figure 3.5: Impact of the DEG and DIV parameters on CRIME's descriptions.

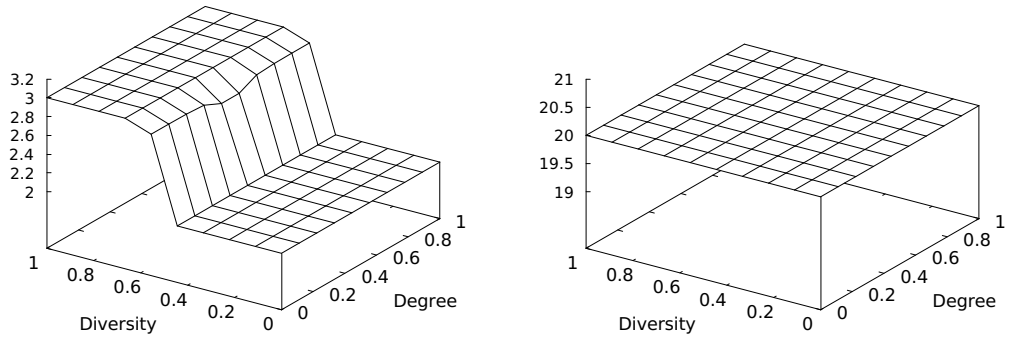


(a) MUSHROOM : low COV - distinct attr.

(b) MUSHROOM : low COV - # attr.

Figure 3.6: Impact of the DEG and DIV parameters on MUSHROOM's descriptions.

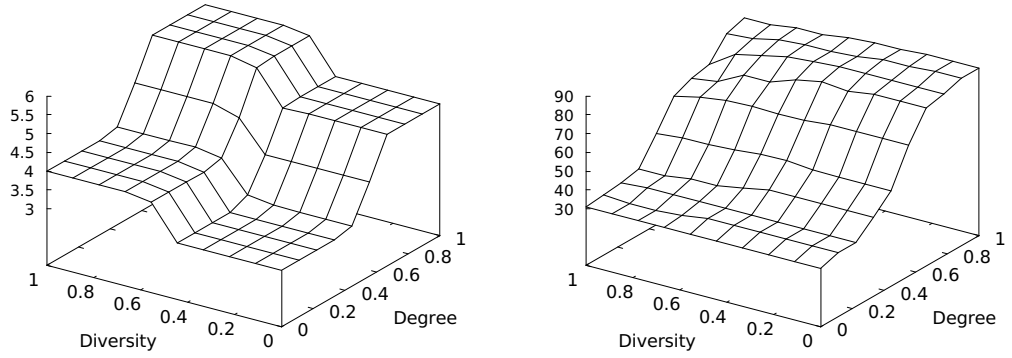
distinct attributes per description is typically low. This means that the descriptions are usually easy to read and are composed of d-formulas that can be compared with each other (because attributes are largely shared). The number of predicates, overlap and precision also show a common trend. The overlap typically increases with the number of predicates while the precision decreases. The motivation is that with more predicates



(a) SENSOR : low COV - distinct attr.

(b) SENSOR : low COV - # attr.

Figure 3.7: Impact of the DEG and DIV parameters on SENSOR's descriptions.



(a) CRIME: high COV - distinct attr.

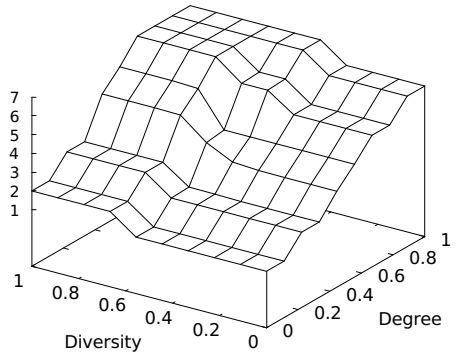
(b) CRIME: high COV - # attr.

Figure 3.8: Impact of the DEG and DIV parameters on CRIME's descriptions.

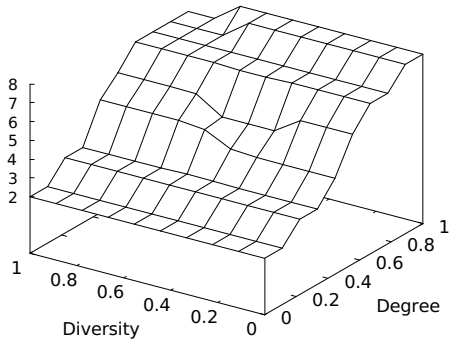
we increase the possibility to describe combinations of values that do not exist in the dataset. The partitions in the SENSORRANDOM scenario have been randomly generated, and contain data that is poorly related. This results in descriptions with high overlap and low purity w.r.t. the SENSOR scenario where the data was partitioned over hour. Indeed this latter result shows that the approach is quite robust against random or "bad" partitioning.

3.7.2.2 Comparison with Other Approaches

In this section we compare our approach against a Decision Tree Classifier (DTC) and Decision Sets (DS) [LBL16].

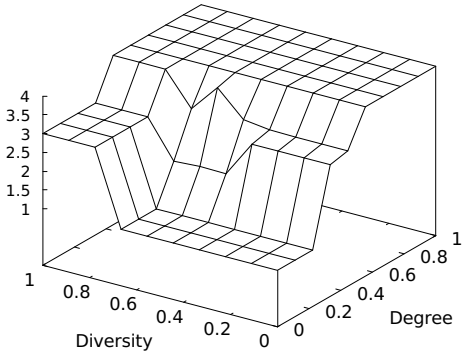


(a) MUSHROOM : high COV - dist. attr.

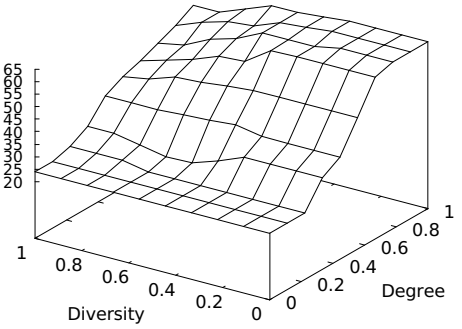


(b) MUSHROOM : high COV - # attr.

Figure 3.9: Impact of the DEG and DIV parameters on MUSHROOM's descriptions.



(a) SENSOR : high COV - distinct attr.



(b) SENSOR : high COV - # attr.

Figure 3.10: Impact of the DEG and DIV parameters on SENSOR's descriptions.

DTC. For each scenario we trained a tree over all data and against the partitioning attribute. The rules obtained are translated into d-formulas composing a description. We used the DTC provided by the Scikit-learn platform [8] with different parameters to find a configuration minimizing the number of rules, and using the entropy H (the same used by *LVA*) as optimization metric for the construction of the tree.

We can observe from Table 3.9 that a decision tree is not a flexible choice since users cannot easily customize any parameter analogously to degree, diversity, and coverage. Moreover, by using the classifier we cannot constraint the generation of a single d-formula per partition. Only for MUSHROOM the number of rules is reasonable. In other scenarios, DTCs generate thousand of rules, thus making impossible for a human to get insights

[8] https://scikit-learn.org/stable/

from them. Note that when the partitions of the dataset are randomly generated, the number of rules explodes, and the coverage of the result is very low.

DS. Decision sets are set of independent if-then rules. Because each rule can be applied independently, decision sets are considered as simple, concise, and easily interpretable. [LBL16] proposes a classification algorithm based on decision sets obtained by selecting from a set of rules generated with the A-PRIORI algorithm [AIS93] the ones maximizing accuracy and interpretability.

We run an experiment using the DS implementation provided by the authors [9] on the MUSHROOM scenario. The A-PRIORI algorithm returned 46 rules and the optimization process took more than 85 hours to return the best 14 rules that we considered a description for the dataset [10]. Then, we computed the coverage, degree and diversity of that description as shown in in Table 3.10. Additionally, we customize our approach to generate a description with similar values. The description takes a couple of minutes to be computed. The second line of Table 3.10 shows that our description is simpler and more interpretable than the one generated by decision sets since it is composed of two rules only (our approach generates one rule per partition), and has less overlap while keeping a similar purity level. The rest of the Table 3.10 shows that the best description is obtained setting high degree and low diversity values.

3.7.2.3 Degree/Diversity Impact on Descriptions

We finally evaluate the effect of user-selected DEG and DIV on the generated descriptions. In particular, we run combinations of degree and diversity levels (DEG and DIV varying from 0.0 to 1.0 with an offset equal to 0.1) and we evaluate the total number of attributes and distinct attributes of the top-1 computed description. Figures 3.5 - 3.10 show the results of this evaluation. For each scenario, only one plot for low and high coverage levels is reported; the other settings show a similar trend. We see that the user's preferences drive the generation of descriptions with the desired features, as expressed by the input

[9] https://github.com/lvhimabindu/interpretable_decision_sets

[10] The small number of rules generated by the A-PRIORI algorithm is due to the high support selected (0.8). Reducing the support we obtained more candidate rules to be selected by the optimization process. With the existing implementation, the optimization process does not converge when more rules are evaluated (we set a 300 hours timeout). Note that the optimization algorithm uses sampling techniques and therefore is not deterministic in time and results.

parameters. When the DEG increases, the number of attributes increases too. Low levels of DIV correspond to a low number of distinct attributes.

3.7.3 Qualitative evaluation of effectiveness

We finally asses the effectiveness of our approach qualitatively using the by submitting the questionnaire shown in Figure 3.11 to a selected number of users (21 PhD Students). The questionnaire is composed of 12 questions related to 8 descriptions representing the MUSHROOM dataset (4 describing a low coverage setting, 4 a high coverage). In the first part of the questionnaire, we evaluate the user's preferences in terms of degree and diversity. 6 questions ask the users to select between two descriptions, the one they would have selected for solving a specific explanation task (i.e. a task that would have required either a broad description or an outlier detection). The proposed descriptions is conceived to force the users to select between a high and a low diversity description or between a high and a low degree description. The result of the experiment is that users prefer low diversity descriptions (61.90%). High diversity descriptions are preferred by 19.05% of the users and the remaining 19.05% of the participants did not express any real preference selecting for similar tasks different kinds of descriptions. The users' preferences in terms of degree are less marked: 42.86% of the users preferred high degree descriptions, 23.80% low degree, and 33.34% did not express any real preference. In the second part, we evaluate the ability of the users to get insighta from a dataset via descriptions. 6 assertions have been proposed to the users who, analyzing the descriptions, had to confirm their insightfulness. Users were able to select the assertions generated by the given descriptions in 68.25% of the cases. Finally, we asked the users to rate their overall experience in analyzing a dataset via a set of rules. The average of the evaluation achieved by a rule-based system as the one proposed was 4.1/5.

To assess the reliability of agreement of the users involved, we computed the Fleiss' kappa measure of the evaluations collected. The result is 0.607 that is typically considered as a substantial agreement among the users.

The Mushroom Data Set

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, and poisonous. The dataset is composed of 8124 instances. It is available at https://archive.ics.uci.edu/ml/datasets/mushroom

		POISONOUS	EDIBLE
D1	LOW COVERAGE	({class = poisonous AND cap-color ∈ {'cinnamon'}})	({class = edible AND cap-color ∈ {'purple'} }) }
D2		({class = poisonous AND ring-number ∈ {'none'} })	({class = edible AND ring-type ∈ {'flaring'} }) }
D3		({class = poisonous AND cap-color ∈ {''cinnamon' '} AND ring-number ∈ {'none '} AND population ∈ {'clustered'} AND odor ∈ {'musty'} AND ring-type ∈ {'none'}})	({class = edible AND cap-color∈ {'purple '} AND spore-print-color ∈ {'chocolate'} AND ring-type ∈ {'flaring'}})
D4		({class = poisonous AND cap-color ∈ {''cinnamon'} AND ring-number ∈ {'none '} AND population ∈ {'clustered'} AND odor ∈ {'musty'}})	({class = edible AND spore-print-color ∈ {'chocolate'} AND ring-type ∈ {'flaring'} })
D5	HIGH COVERAGE	{class = p AND spore-print-color ∈ {'chocolate', 'white'} })	({class = e AND spore-print-color ∈ {'brown', 'black'} }})
D6		({class = poisonous AND bruises ∈ {'no'} })	({class = edible AND odor ∈ {'none'} })
D7		({class = poisonous AND spore-print-color ∈ {'chocolate', 'white'} AND ring-number ∈ {'one'}}) OR	({class = edible AND spore-print-color ∈ {'brown', 'black'} AND cap-surface ∈ {'scaly', 'smooth', 'fibrous'} AND ring-number ∈ {'one'}}) }
D8		({class = poisonous AND ring-number ∈ {'one '} AND bruises ∈ {'no'} }) OR	({class = edible AND spore-print-color ∈ {'brown', 'black'} AND cap-surface ∈ {'scaly', "smooth', 'fibrous'} AND bruises ∈ {'no', 'yes'} AND ring-number ∈ {'one'} })

Let us suppose that you want to know some anomalies about the dataset.

1. Do you prefer D1 or D5?
2. Do you prefer D1 or D2?
3. Do you prefer D1 or D3?
4. Is there any real difference between the knowledge of the dataset content you obtain from D3 and D4? YES / NO

Let us suppose that you want to have a broad, generic description about the dataset.

1. Do you prefer D5 or D6?
2. Do you prefer D5 or D7?

Would you eat

1. a mushroom with a cap-color purple? YES / NO
2. a mushroom with a cap-color cinnamon? YES / NO
3. a mushroom with 1 ring? YES / NO
4. a mushroom with 1 ring and no bruises? YES / NO

Is it true that

1. an edible mushroom has no odor? YES / NO
2. an edible mushroom has spore-print-color brown or black? YES / NO

Figure 3.11: Questionnaire administered to the users.

3.8 Summary

In this chapter, we introduce the problem of generating data descriptions: a compact and insightful set of predicates allowing data enthusiasts to inspect datasets at a glance. We propose a set of metrics and heuristics to drive the generation of data descriptions in a computationally efficient manner while returning high-quality results. We run extensive experiments over real-world dataset to assess the performance of our approach both quantitatively and qualitatively. We think that data descriptions are a first step towards the goal of helping users inspecting a set of records.

Chapter 4

Discovering Order Dependencies through Order Compatibility

In the previous chapter, we have studied the data description problem, in which a user retrieves the top-k descriptions of datasets in order to its content. In this chapter, instead, we study a problem related to dependency discovery, i.e., relationships between columns of a dataset.

In this context, we study the problem of order dependency discovery, we provide a new solution to this problem that could be easily parallelized and outperform existing approaches. This chapter describes our findings and solutions for this problem. We conclude with a thorough experimental assessment on several real and synthetic relational databases.

4.1 Contributions

In particular, our contributions are as follows:

- We introduce a definition of minimality for a set of order compatibility dependencies that we show being complete in the sense that it can recover all valid order compatibility dependencies that hold over a given instance of relational data;
- We propose a novel algorithm for finding order dependencies that is complete and it performs the detection of order dependencies in parallel.

- We perform an exhaustive experimental evaluation that shows the performance of our algorithm in comparison with existing works, including a study of its scalability over big datasets and multiple threads.
- We discuss possible solutions for the discovery of the most important order dependencies in the case of datasets that could not be managed (higher number of columns) in a reasonable amount of time.

4.2 Outline

This chapter is structured as follows. First, Section 4.3 presents an illustrative example. Then in Section 4.4, we review the relevant definitions and theorems that formalize the connection between order dependencies and order compatibility dependencies. In Section 4.6 we prove that order dependency discovery can be guided by order compatibility dependencies without losing completeness. Our novel algorithm is presented in Section 4.7, while Section 4.8 contains the discussion of our experimental evaluation.

4.3 Motivating Example

An example of functional dependency can be seen in Table 4.1, that shows a relational table with data regarding yearly incomes, savings and taxes. Assume that the tax system is a progressive one that categorizes the different incomes into brackets, each of them characterized by a tax percentage. Thus, there is a functional dependency from the income amount to the tax brackets, i.e., `income` → `bracket`. Since for every income range the percentage is fixed, there are two other functional dependencies from the income to the tax amount and vice-versa, i.e., `income` → `tax` and `tax` → `income`. Using the transitive property of functional dependencies a new one can be inferred, i.e., `tax` → `bracket`.

A closer look at Table 4.1 can illustrate another, stronger, form of dependency: as the income is increasing, bracket and tax amount are increasing as well. In other words, if we were to order the table based on the income column, each one of the bracket and the tax amount columns will also end up being ordered. This form of dependency is known

as an *order dependency* and is typically noted with the $\mapsto$ symbol, i.e., `income` $\mapsto$ `tax`, which is read as: `income` *orders* `tax`.

The most important application of order dependencies is their use in the optimization of queries; in particular, they can be used to rewrite the `ORDER BY` clauses in SQL queries in ways similar to that of functional dependencies for the `GROUP BY` statements [SSM96, SGG+14]. Consider the following query:

```
SELECT income, bracket, tax
FROM TaxInfo
ORDER BY income, bracket, tax
```

Given that the order dependencies `income` $\mapsto$ `tax` and `income` $\mapsto$ `bracket` hold, the query optimizer can infer that sorting by `income` makes the ordering on the other two columns redundant, so the `ORDER BY` clause can be simplified to `ORDER BY income`.

In the example of Table 4.1, the point-wise order dependency `income`, `tax` $\rightsquigarrow$ `bracket` holds because if both of the tuples (`income`, `tax`) and (`tax`, `income`) are lexicographically ordered, then the column `bracket` is ordered in the same way.

There are cases where two lists of attributes order each other when taken together. This property is known as *order compatibility* and is denoted with the symbol $\sim$. In Table 4.1, e.g., it holds that

$$(\texttt{income}, \texttt{savings}) \mapsto (\texttt{savings}, \texttt{income}) \qquad \text{and}$$
$$(\texttt{savings}, \texttt{income}) \mapsto (\texttt{income}, \texttt{savings})$$

TaxInfo

name	income	savings	bracket	tax
T. Green	35,000	3,000	1	5,250
J. Smith	40,000	4,000	1	6,000
J. Doe	40,000	3,800	1	6,000
S. Black	55,000	6,500	2	8,500
W. White	60,000	6,500	2	9,500
M. Darrel	80,000	10,000	3	14,000

Table 4.1: A relational table with financial information.

and thus `income` ~ `savings`. Another way to see an order compatibility dependency between two columns is that their values are both monotonically non-decreasing when they are considered pairwise.

4.4 Background

To provide the background of our problem, we first review the definition of order dependency and its axiomatization, then we describe the formal relation between order dependencies (ODs), functional dependencies (FDs) and order compatibility dependencies (OCDs).

4.4.1 Notational Conventions and Definitions

We adopt the notational conventions summarized in Table 4.2, consistently with the literature [SGG12b]. Let $\boldsymbol{R}$ be a relation over the set of attributes $\mathcal{U}$ and $\boldsymbol{r}$ be a table instance of $\boldsymbol{R}$, i.e. a set of tuples under $\boldsymbol{R}$'s schema. A tuple p can be projected over a single attribute A, over a set of attributes $\mathcal{X}$ and over a list of attributes $\mathbf{X}$ by subscripting the tuple as follows: p_A, $p_{\mathcal{X}}$, $p_{\mathbf{X}}$.

We assume that a total ordering is defined over each of the attributes, denoted $\leq_A$; in the following, however, we will drop the attribute specification and use $\leq$, as the attribute will be always clear from the context. Order dependencies are defined based on the operator $\preccurlyeq$, which is equivalent to a lexicographical ordering over a list of attributes, and is defined by:

Definition 1 (operator $\preccurlyeq$)**.** Given a list of attributes $\mathbf{X} := [A|\mathbf{T}]$ and two tuples $p, q \in \boldsymbol{r}$, the operator $\preccurlyeq$ (and its associated operator $\prec$) are defined as follows:

$$\begin{aligned} p_{\mathbf{X}} \preccurlyeq q_{\mathbf{X}} \quad &\Leftrightarrow \quad \big(p_A < q_A\big) \vee \\ &\qquad \big((p_A = q_A) \wedge (\mathbf{T} = [\cdot] \vee p_{\mathbf{T}} \preccurlyeq q_{\mathbf{T}})\big) \\ p_{\mathbf{X}} \prec q_{\mathbf{X}} \quad &\Leftrightarrow \quad p_{\mathbf{X}} \preccurlyeq q_{\mathbf{X}} \wedge p_{\mathbf{X}} \neq q_{\mathbf{X}} \end{aligned} \tag{4.1}$$

The $\preccurlyeq$ operator reproduces the ordering clause `ORDER BY ASC` in SQL [SGG12b].

Based on the comparison operator of Definition 1, we can introduce the concept of *order dependency* [SGG12b].

Definition 2 (Order dependency (OD))**.** Given a relation $\boldsymbol{R}$ and two lists $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X} \mapsto \mathbf{Y}$ is an *order dependency* if, for any instance $\boldsymbol{r}$ of $\boldsymbol{R}$ and for every pair of tuples $p, q \in \boldsymbol{r}$, the following implication holds:

$$p_{\mathbf{X}} \preccurlyeq q_{\mathbf{X}} \Rightarrow p_{\mathbf{Y}} \preccurlyeq q_{\mathbf{Y}} \tag{4.2}$$

If both $\mathbf{X} \mapsto \mathbf{Y}$ and $\mathbf{Y} \mapsto \mathbf{X}$ hold, we say that $\mathbf{X}$ and $\mathbf{Y}$ are *order equivalent* and we write $\mathbf{X} \leftrightarrow \mathbf{Y}$. If $\mathbf{XY} \leftrightarrow \mathbf{YX}$ we say that $\mathbf{X}$ and $\mathbf{Y}$ are *order compatible* and we write $\mathbf{X} \sim \mathbf{Y}$. We will discuss the latter relation in Section 4.4.2.

Order dependencies satisfy the set of axioms $\mathcal{J}_{\text{OD}}$, introduced by Szlichta et al. [SGG12b], that are reported in Table 4.3. These axioms are analogous to the Armstrong axioms for functional dependencies [Arm74].

4.4.2 Decomposing Order Dependencies

We show how an order dependency can be decomposed in a pair composed of one functional dependency and one order compatibility dependency.

Relations: ▸ $\boldsymbol{R}$, written as a capital letter in bold italics, is a *relation* over a set of attributes $\mathcal{U}$; ▸ $\boldsymbol{r}$, written as a lowercase letter in bold italics, is a *table instance* over $\boldsymbol{R}$, i.e. a *set of tuples*; ▸ Single *attributes* are represented with capital letters: A, B, and C; ▸ *Tuples* are represented with lowercase letters: p, q, s, and t.
Lists: ▸ Bold capital letters are *lists* of attributes: $\mathbf{X}$,$\mathbf{Y}$, and $\mathbf{Z}$. they can represent the empty list $[\cdot]$; ▸ A list is denoted with square brackets $[A, B, C]$. A list $[A\|\mathbf{T}]$ is composed by a *head* A and a *tail* $\mathbf{T}$; ▸ $\mathbf{XY}$ is a shorthand for $\mathbf{X} \circ \mathbf{Y}$, $\mathbf{X}A$ and $A\mathbf{X}$ are shorthands for $\mathbf{X} \circ [A]$ and $[A] \circ \mathbf{X}$ respectively, AB denotes $[A, B]$.

Table 4.2: Notational conventions

AX1: *Reflexivity*

$$\mathbf{XY} \mapsto \mathbf{X}$$

AX2: *Prefix*

$$\frac{\mathbf{X} \mapsto \mathbf{Y}}{\mathbf{ZX} \mapsto \mathbf{ZY}}$$

AX3: *Normalization*

$$\mathbf{WXYXV} \leftrightarrow \mathbf{WXYV}$$

AX4: *Transitivity*

$$\frac{\mathbf{X} \mapsto \mathbf{Y} \quad \mathbf{Y} \mapsto \mathbf{Z}}{\mathbf{X} \mapsto \mathbf{Z}}$$

AX5: *Suffix*

$$\frac{\mathbf{X} \mapsto \mathbf{Y}}{\mathbf{X} \leftrightarrow \mathbf{YX}}$$

AX6: *Chain*

$$\frac{\begin{array}{c}\mathbf{X} \sim \mathbf{Y}_1 \\ \forall_{i \in [1,n-1]} \mathbf{Y}_i \sim \mathbf{Y}_{i+1} \\ \mathbf{Y}_n \sim \mathbf{Z} \\ \forall_{i \in [1,n]} \mathbf{Y}_i \mathbf{X} \sim \mathbf{Y}_i \mathbf{Z}\end{array}}{\mathbf{X} \sim \mathbf{Z}}$$

Table 4.3: The set of axioms $\mathcal{J}_{\text{OD}}$ for order dependencies

Functional dependencies. Functional dependencies encode the fact that, in a relation, one attribute determines completely another attribute.

Definition 3 (Functional dependency (FD))**.** Given a relation $\boldsymbol{R}$ and two sets of attributes $\mathcal{X}$ and $\mathcal{Y}$, $\mathcal{X} \to \mathcal{Y}$ is a *functional dependency* if, for any instance $\boldsymbol{r}$ of $\boldsymbol{R}$ and for every pair of tuples $p, q \in \boldsymbol{r}$, the following implication holds:

$$p_{\mathcal{X}} = q_{\mathcal{X}} \Rightarrow p_{\mathcal{Y}} = q_{\mathcal{Y}} \tag{4.3}$$

Order Compatibility Dependencies. Order compatibility dependencies encode the fact that in a relation two lists of attributes show the same monotonicity. If we order either combination of the lists in non-decreasing order, they end up both ordered such their values are both monotonically non-decreasing.

Definition 4 (order compatibility dep. (OCD))**.** Given a relation $\boldsymbol{R}$ and two lists of attributes $\mathbf{X}$ and $\mathbf{Y}$ in $\boldsymbol{R}$, $\mathbf{X} \sim \mathbf{Y}$ is a *order compatibility dependency* if, for any instance $\boldsymbol{r}$ of $\boldsymbol{R}$ and for every pair of tuples $p, q \in \boldsymbol{r}$, the following implications hold:

$$p_{\mathbf{XY}} \preccurlyeq q_{\mathbf{XY}} \Leftrightarrow p_{\mathbf{YX}} \preccurlyeq q_{\mathbf{YX}} \tag{4.4}$$

Order dependencies are a stricter relation between two attributes with respect to functional and order compatibility dependencies; furthermore, when an order dependency

between two lists of attributes $\mathbf{X}$ and $\mathbf{Y}$ holds, an order compatibility dependency between $\mathbf{X}$ and $\mathbf{Y}$ holds as well. In this sense, order dependencies combine the fact that an attribute functionally determines and have the same monotonicity of another when ordered together.

We present previous results [SGG12b] that formally highlight the nature of the relationship between these dependencies, showing that when an order dependency does not hold there are only two possible scenarios called *split* and *swap*. Although the proofs of the theorems are not our own contribution, we have included it for ease of reference in the Section 4.5. When $\mathbf{X}$ does not order $\mathbf{Y}$, i.e. when the order dependency between $\mathbf{X}$ and $\mathbf{Y}$ does not hold, we write $\mathbf{X} \nmapsto \mathbf{Y}$.

Split. A split indicates the case where there exists a pair of tuples that have the same values when projected over the attributes $\mathbf{X}$, but have different values over the attributes $\mathbf{Y}$. Formally:

Definition 5 (split)**.** Two tuples $s, t \in \boldsymbol{r}$ form a split over two lists of attributes $\mathbf{X}$, $\mathbf{Y}$ iff $s_{\mathbf{X}} = t_{\mathbf{X}}$ but $s_{\mathbf{Y}} \neq t_{\mathbf{Y}}$, or equivalently:

$$\exists s, t \in \boldsymbol{r} \ : \ s_{\mathbf{X}} = t_{\mathbf{X}} \wedge s_{\mathbf{Y}} \succ t_{\mathbf{Y}}$$

When an OD between two attributes list is valid, then a FD is valid as well (Theorem 15, [SGG12b])

Theorem 6 (ODs subsume FDs)**.** *For every instance $\boldsymbol{r}$ of relation $\boldsymbol{R}$, if the OD $\mathbf{X} \mapsto \mathbf{Y}$ holds, then the FD $\mathcal{X} \rightarrow \mathcal{Y}$ holds.*

Whereas if there is a split between two lists of attributes $\mathbf{X}$ and $\mathbf{Y}$, there is no guarantee that ordering data will result in ordered tuples over $\mathbf{Y}$ and $\mathbf{XY}$; in other words, both the ODs $\mathbf{X} \mapsto \mathbf{Y}$ and $\mathbf{X} \mapsto \mathbf{XY}$ do not hold. Furthermore, a split falsifies the functional dependency $\mathcal{X} \rightarrow \mathcal{Y}$ as well.

The relationship between order and functional dependencies is formalized as follows (Theorem 13, [SGG12b]):

Theorem 7 (FD and OD correspondence)**.** *For every instance $\boldsymbol{r}$ of a relation $\boldsymbol{R}$, the functional dependency $\mathcal{X} \rightarrow \mathcal{Y}$ holds iff $\mathbf{X} \mapsto \mathbf{XY}$ holds for all lists $\mathbf{X}$ that order the attributes of $\mathcal{X}$ and all lists $\mathbf{Y}$ that order the attributes of $\mathcal{Y}$.*

	A	B
t_1	1	4
t_2	2	5
t_3	3	6
t_4	3	7
t_5	4	1

Table 4.4: A relational table containing both a split and a swap between the two attributes A and B.

In Table 4.4, tuples t_3 and t_4 form a split for the attributes A and B, thus $A \nrightarrow B$ and $A \nrightarrow AB$. The functional dependency $A \rightarrow B$ is not valid, as well.

Swap. A *swap* indicates the case where there exists a pair of tuples whose values projected over two lists of attributes $\mathbf{X}$ and $\mathbf{Y}$ are swapped, i.e. they are sorted differently when they are ordered with respect to $\mathbf{X}$ or $\mathbf{Y}$. Formally:

Definition 8 (Swap)**.** Two tuples $s, t \in \boldsymbol{R}$ form a swap over two list of attributes $\mathbf{X}$ and $\mathbf{Y}$, iff $s_{\mathbf{X}} \prec t_{\mathbf{X}}$ but $t_{\mathbf{Y}} \prec s_{\mathbf{Y}}$, or equivalently:

$$\exists s, t \in \boldsymbol{r} \ : \ s_{\mathbf{X}} \prec t_{\mathbf{X}} \wedge s_{\mathbf{Y}} \succ t_{\mathbf{Y}}$$

Swaps between $\mathbf{X}$ and $\mathbf{Y}$ falsify the ODs of the form $\mathbf{X} \mapsto \mathbf{Y}$, $\mathbf{Y} \mapsto \mathbf{X}$, and $\mathbf{XY} \leftrightarrow \mathbf{YX}$. For example, tuples t_1 and t_5 in Table 4.4 form a swap for attributes A and B, thus $A \nrightarrow B$, $AB \nrightarrow B$, and $AB \nrightarrow BA$.

Splits and swaps establish a correspondence between order dependencies, functional dependencies and order compatibility dependencies, as in the following theorem (Theorem 15, [SGG12b]):

Theorem 9 (OD = FD + OCD)**.** $\mathbf{X} \mapsto \mathbf{Y}$ *holds iff* $\mathcal{X} \rightarrow \mathcal{Y}$ *(*$\mathbf{X} \mapsto \mathbf{XY}$*) and* $\mathbf{X} \sim \mathbf{Y}$ *(*$\mathbf{XY} \leftrightarrow \mathbf{YX}$*).*

In summary, when an order dependency between two lists of attributes $\mathbf{X}$ and $\mathbf{Y}$ holds:

- a functional dependency $\mathcal{X} \rightarrow \mathcal{Y}$ holds, which implies the absence of split conditions;
- an order compatibility dependency between $\mathbf{X}$ and $\mathbf{Y}$ holds, which implies the absence of swap conditions.

We exploit this relation to guide our discovery algorithm as established in Section 4.7.2.

4.5 Proofs of fundamental theorems of OD

In this section, we first introduce the definition of functional and order compatibility dependencies and then we show how an order dependency can be decomposed in a pair of one FD and one OCD.

Theorem 10 (ODs subsume FDs)**.** *For every instance* $\boldsymbol{r}$ *of relation* $\boldsymbol{R}$*, if the OD* $\mathbf{X} \mapsto \mathbf{Y}$ *holds then the FD* $\mathcal{X} \rightarrow \mathcal{Y}$ *holds.*

Proof. Let $s, t \in \boldsymbol{r}$, such that $s_{\mathbf{X}} = t_{\mathbf{X}}$. Therefore $s_{\mathbf{X}} \preccurlyeq t_{\mathbf{X}}$ and $t_{\mathbf{X}} \preccurlyeq s_{\mathbf{X}}$. By the definition of OD, $s_{\mathbf{Y}} \preccurlyeq t_{\mathbf{Y}}$ and $t_{\mathbf{Y}} \preccurlyeq s_{\mathbf{Y}}$. We can then conclude that $s_{\mathbf{Y}} = t_{\mathbf{Y}}$ and thus $s_{\mathcal{Y}} = t_{\mathcal{Y}}$. □

The following theorem establishes a more refined relationship between order and functional dependencies (Theorem 13, [SGG12b]).

Theorem 11 (FD and OD correspondence)**.** *For every instance* $\boldsymbol{r}$ *of relation* $\boldsymbol{R}$*, the FD* $\mathcal{X} \rightarrow \mathcal{Y}$ *holds iff* $\mathbf{X} \mapsto \mathbf{XY}$ *holds for all lists* $\mathbf{X}$ *that order the attributes of* $\mathcal{X}$ *and all lists* $\mathbf{Y}$ *that order the attributes of* $\mathcal{Y}$*.*

Proof.

- ⇒ If $\mathbf{X} \mapsto \mathbf{XY}$ holds, by Theorem 6 $\mathcal{X} \rightarrow \mathcal{XY}$. By the Reflexivity axiom on FDs [Arm74], $\mathcal{XY} \rightarrow \mathcal{Y}$ is also true. By Transitivity on FDs [Arm74], we can prove that $\mathcal{X} \rightarrow \mathcal{Y}$ is true.
- ⇐ If, by contradiction, there are two lists of attributes $\mathbf{X}, \mathbf{Y}$ such that $\mathbf{X} \mapsto \mathbf{XY}$ does not hold, there exists $s, t \in \boldsymbol{r}$ such that $s_{\mathbf{X}} \preccurlyeq t_{\mathbf{X}}$ but $s_{\mathbf{XY}} \not\preccurlyeq t_{\mathbf{XY}}$. This implies that $s_{\mathbf{X}} = t_{\mathbf{X}}$ and $s_{\mathbf{Y}} \succ t_{\mathbf{Y}}$. Therefore $s_{\mathcal{Y}} \neq t_{\mathcal{Y}}$ and $\mathcal{X} \rightarrow \mathcal{Y}$ is not true.

□

Theorem 12 (OD = FD + OCD)**.** $\mathbf{X} \mapsto \mathbf{Y}$ *holds iff* $\mathbf{X} \mapsto \mathbf{XY}$ *and* $\mathbf{X} \sim \mathbf{Y}$ *(*$\mathbf{XY} \leftrightarrow \mathbf{YX}$*).*

Proof. Now we can prove the original theorem:

⇒ This part can be separated in two claims:

- ($\mathbf{X} \mapsto \mathbf{Y} \Rightarrow \mathbf{X} \mapsto \mathbf{XY}$): By applying the Union Theorem (Theorem 2, [SGG12b]) to $\mathbf{X} \mapsto \mathbf{Y}$ and the trivial OD $\mathbf{X} \mapsto \mathbf{X}$, we obtain $\mathbf{X} \mapsto \mathbf{XY}$.
- ($\mathbf{X} \mapsto \mathbf{Y} \Rightarrow \mathbf{XY} \leftrightarrow \mathbf{YX}$): If $\mathbf{X} \mapsto \mathbf{Y}$ holds, then by Suffix (AX5) $\mathbf{X} \leftrightarrow \mathbf{YX}$ holds as well. By Reflexivity (AX1) $\mathbf{XY} \mapsto \mathbf{X}$ and by Transitivity (AX4) $\mathbf{XY} \mapsto \mathbf{YX}$. Finally, from the previous point, $\mathbf{X} \mapsto \mathbf{Y}$ implies $\mathbf{X} \mapsto \mathbf{XY}$ and thus, then by Transitivity (AX4) $\mathbf{YX} \mapsto \mathbf{XY}$, thus proving that $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

⇐ If $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$, by Transitivity (AX4) $\mathbf{X} \mapsto \mathbf{YX}$. By Reflexivity (AX1) $\mathbf{YX} \mapsto \mathbf{Y}$ and by Transitivity (AX4) again $\mathbf{X} \mapsto \mathbf{Y}$.

□

4.6 Order Dependency Through Order Compatibility

This section introduces the concepts that lay the foundations to our approach.

4.6.1 Minimality of Discovered Dependencies

Similarly to what has been done for functional dependencies, we introduce the notion of minimality of a set of order compatibility dependencies. In principle, minimality is the property for which a set of dependencies equipped with inference rules can recover all the dependencies that are valid in a given instance of relational data. Axioms AX1-AX6 presented in Table 4.3 provide inference rules for order dependencies.

The concept of minimality for order dependencies, as for other types of dependencies, have several uses. First of all, it allows reasoning about the validity of pruning rules, e.g. to show that they do not lead to loss of information about valid dependencies in the relation.

Minimality serves also the purpose of compressing information to a manageable size: in fact, if we take a relation with n attributes, in the worst case where each of which is order equivalent to every other, the minimal set of ODs would contain $n-1$ dependencies ($A \leftrightarrow B, A \leftrightarrow C$, etc.), while the set of all valid dependencies would contain $\mathcal{O}((n!)^2)$

elements: all the possible combinations of attributes on the left-hand and right-hand sides, a prohibitively large number.

Finally, real applications may not need the whole list of dependencies: for example, in knowledge discovery, redundant dependencies do not add value to the properties discovered and too many dependencies can cause the most important ones to be missed; in query optimization, the only useful dependencies are those that can be applied to the queries to be performed.

Any pruning rule applied by a dependency discovery algorithm needs to respect minimality, in the sense that it should allow the recovery of the full set. A complete algorithm must find at least all *minimal* order dependencies over an instance $\boldsymbol{r}$ of a relation $\boldsymbol{R}$.

To introduce the concept of minimality for ODs, we start by presenting the concepts of closure and equivalence of sets of order dependencies.

Definition 13 (closure)**.** The *closure* of the set of ODs $\mathcal{M}$ denoted $\mathcal{M}^+$, is the set of ODs that are logically implied from $\mathcal{M}$ by the axioms $\mathcal{J}_{\text{OD}} = \{\text{AX1} - \text{AX6}\}$ defined in Table 4.3.

$$\mathcal{M}^+ = \{\mathbf{X} \mapsto \mathbf{Y} \mid \mathcal{M} \vdash_{\mathcal{J}_{\text{OD}}} \mathbf{X} \mapsto \mathbf{Y}\}$$

Definition 14 (equivalence of sets of ODs)**.** Two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ of order dependencies are *equivalent* if and only if they have the same closure $\mathcal{M}_1^+ = \mathcal{M}_2^+$.

The closure of a set of minimal dependencies is the set of all the dependencies that are valid over $\boldsymbol{r}$. We build this definition first showing which lists of attributes are in minimal form and then when an OCD is minimal. Finally, we prove that our definition of minimality is complete.

Order compatibility dependencies employ attribute lists instead of attribute sets, thus we introduce the concepts of *minimal* attribute list. An attribute list is minimal if in it has no embedded order dependency, i.e. the list of attributes is the shortest possible.

Definition 15 (minimal attribute list)**.** An attribute list $\mathbf{X}$ is minimal if there is no other list $\mathbf{X}'$ such that:

- $\mathbf{X}'$ is smaller than $\mathbf{X}$, and
- $\mathbf{X}$ and $\mathbf{X}'$ are order equivalent.

For example, the attribute list ABA is never minimal, in fact by the Normalization axiom (AX3) we know that $ABA \leftrightarrow AB$ and AB is a shorter list than ABA. Instead, AB is a minimal attribute list, unless $A \leftrightarrow B$.

An OCD is minimal if both sides are given as minimal attribute lists and there are no repeated attributes:

Definition 16 (minimal OCD)**.** An OCD $\mathbf{X} \sim \mathbf{Y}$ is minimal if:

- $\mathbf{X}$ and $\mathbf{Y}$ are minimal attribute lists;
- $\mathcal{X} \cap \mathcal{Y} = \varnothing$.

We show in the following theorem that OCDs with repeated attributes can be derived from OCDs without repeated attributes:

Theorem 17 (Completeness of minimal OCD)**.** *Order compatibility dependencies with repeated attributes can be derived from OCD without repeated attributes.*

Proof. The proof of this theorem is split in three cases:

1. OCDs of the form $\mathbf{XY} \sim \mathbf{XZ}$ can be derived from $\mathbf{Y} \sim \mathbf{Z}$;
2. OCDs of the form $\mathbf{XY} \sim \mathbf{MY}$ can be derived from $\mathbf{XY} \sim \mathbf{M}$ and $\mathbf{X} \sim \mathbf{MY}$;
3. OCDs of the form $\mathbf{XY} \sim \mathbf{MYN}$ can be derived from $\mathbf{X} \sim \mathbf{M}$, $\mathbf{XY} \sim \mathbf{M}$, $\mathbf{X} \sim \mathbf{MY}$ and $\mathbf{XY} \sim \mathbf{MN}$;

which are covered respectively by Theorems 22, 23 and 24 which are presented separatedly for clarity in Section 4.6.3. □

The following result extends the Downward Closure theorem (Theorem 12, [SGG12b]):

Theorem 18. *Downward closure for OCD*

$$\frac{\mathbf{XY} \sim \mathbf{Z}\,\boldsymbol{V}}{\mathbf{X} \sim \mathbf{Z}}$$

Theorems 17 and 18 provide the justification to the structure of the search tree used in our approach as explained in Section 4.7.2. In particular we derive the following pruning rule:

Theorem 19 (Pruning rule for OCD)**.**

$$\frac{\mathbf{X} \nsim \mathbf{Z}}{\mathbf{XY} \nsim \mathbf{Z}\boldsymbol{V}}$$

Proof. This theorem is the contronominal preposition of Theorem 18. □

Theorem 17 justifies the reduction of the search space that we highlight in the following section.

4.6.2 Dimension of the Search Space

The number of valid ODs over a relation $\boldsymbol{R}$ is vast: in fact if $\boldsymbol{R}$ has n attributes, then all permutations of length k of n elements must be considered both for the left-hand side and the right-hand side of the OD. We address a limitation of the previous work by Langer and Naumann [LN16] and we show that some order dependencies with repeated attributes can not be derived from other dependencies without repeated attributes.

For example, in Table 4.5 (a) we have that $AB \nmapsto B$, instead in Table 4.5 (b) we have $AB \mapsto B$ and $A \sim B$. For both tables the dependencies $A \nmapsto B$ and $B \nmapsto A$ do not hold, thus the validity of $AB \mapsto B$ and $A \sim B$ in this case can not be inferred from shorter ODs.

	A	**B**
t_1	1	4
t_2	2	5
t_3	3	6
t_4	3	7
t_5	4	1

A	**B**
1	4
2	5
3	6
3	7
4	7

Table 4.5: Two relations where the ODs $A \nmapsto B$ and $B \nmapsto A$ do not hold; furthermore in (a) $AB \nmapsto B$ while in (b) $AB \mapsto B$ and $A \sim B$.

Finding all valid order dependencies thus requires, in principle, the need for checking all combinations $\mathbf{X} \mapsto \mathbf{Y}$ where both $\mathbf{X}$ and $\mathbf{Y}$ can be permutations of length k of the n attributes in $\boldsymbol{R}$ with $1 \leq k \leq n$. If we denote with $S(n)$ the number of k-permutations of n elements we have:

$$S(n) = \lfloor e \cdot n! \rfloor - 1$$

The number k-permutations of n elements is given by:

$$kPn = \prod_{i=0}^{k}(n-k) \tag{4.5}$$

We can also write $kPn = (k-1)Pn \cdot (n-k+1)$ with $0Pn = 1$. We want to calculate the following sum:

$$\begin{aligned} S(n) &= \sum_{k=0}^{n} kPn = \sum_{k=0}^{n}\prod_{i=0}^{k}(n-k) \\ &= 1 + n + n\cdot(n-1) + \cdots + n! \end{aligned} \tag{4.6}$$

We also note that the $S(n)$ can be obtained from the recursive expression:

$$\begin{aligned} S(n) &= 1 + n + n\cdot(n-1) + \cdots + n! \\ &= 1 + n\cdot(1 + (n-1) + (n-1)\cdot(n-2) + \cdots + (n-1)! \\ &= 1 + nS(n-1) \end{aligned} \tag{4.7}$$

The series in 4.6 can be written as:

$$S(n) = n!\cdot\left(\frac{1}{0!} + \frac{1}{1!} + \cdots + \frac{1}{n!}\right) \tag{4.8}$$

Recalling the definition of the Napier constant $e := \sum_{i=0}^{\infty}\frac{1}{i!}$ we can write:

$$S(n) = n!\cdot e - \left(\frac{1}{(n+1)} + \frac{1}{(n+1)(n+2)} + \ldots\right) \tag{4.9}$$

We can denote the expression in parenthesis with $r(n) = \frac{1}{(n+1)} + \frac{1}{(n+1)(n+2)} + \ldots$. The following inequality holds:

$$r(n) \le \frac{1}{n} + \frac{1}{n^2} + \frac{1}{n^3} + \cdots = \sum_{k=0}^{\infty}\left(\frac{1}{n}\right)^k - 1 = \frac{1}{n-1} \tag{4.10}$$

For $n > 2$ then $r(n) < 1$ and therefore:

$$e\cdot n! - 1 < S(n) < e\cdot n! \tag{4.11}$$

Since $S(n)$ must be integer we have:

$$S(n) = \lfloor e\cdot n!\rfloor \tag{4.12}$$

Excluding trivial ODs of the form $\mathbf{X} \mapsto \mathbf{X}$, the number of candidates that needs to be checked would be:

$$C(n) = \big(S(n) - 1\big) \cdot \big(S(n) - 1\big) - \big(S(n) - 1\big) \propto \mathcal{O}((n!)^2) \tag{4.13}$$

With $n = 10$, there are more than $97 \cdot 10^{12}$ candidates ODs.

In contrast to general order dependencies, OCDs candidates with repeated attributes, i.e. $\mathbf{X} \rightarrow \mathbf{Y}$ or $\mathbf{X} \sim \mathbf{Y}$ where $\mathcal{X} \cap \mathcal{Y} \neq \varnothing$, are redundant in the sense that their validity can be inferred from the validity of other dependencies of the same type without repeated attributes and with shorter attribute lists.

Theorem 20. $\mathbf{X} \sim \mathbf{Y}$ *iff* $\mathbf{XY} \mapsto \mathbf{Y}$

Proof. We prove the implication in each direction:

- $\Rightarrow$ By definition $\mathbf{X} \sim \mathbf{Y}$ implies that both the order dependencies $\mathbf{XY} \mapsto \mathbf{YX}$ and $\mathbf{YX} \mapsto \mathbf{XY}$ are valid. By Reflexivity (AX1) $\mathbf{YX} \mapsto \mathbf{Y}$ and thus by Transitivity (AX4) the order dependency $\mathbf{XY} \mapsto \mathbf{Y}$ is valid.
- $\Leftarrow$ Conversely, if $\mathbf{XY} \mapsto \mathbf{Y}$, by Suffix (AX5) $\mathbf{XY} \mapsto \mathbf{YXY}$ and Normalization (AX3) $\mathbf{XY} \mapsto \mathbf{YX}$. □

□

This means that ODs of the form $\mathbf{XY} \mapsto \mathbf{Y}$ and OCDs of the form $\mathbf{X} \sim \mathbf{Y}$ are equivalent. We are thus enabled to solve the problem by considering only OCDs without repeated attributes, and thus the dimension of the search space is reduced to $\mathcal{O}(n!)$.

As shown in Theorem 9, if $\mathbf{X} \mapsto \mathbf{Y}$ then $\mathbf{X} \sim \mathbf{Y}$ is valid; we can thus derive the following theorem, which will provide the foundation for the pruning rules detailed in Section 4.7.2.1.

Theorem 21.

$$\frac{\mathbf{X} \mapsto \mathbf{Y}}{\mathbf{XZ} \sim \mathbf{Y}} \tag{4.14}$$

Proof. By the Augmentation theorem [SGG12b], $\mathbf{X} \mapsto \mathbf{Y}$ implies $\mathbf{XZ} \mapsto \mathbf{Y}$. By Theorem 9 of Section 4.4.2, $\mathbf{XZ} \mapsto \mathbf{Y}$ implies $\mathbf{XZ} \sim \mathbf{Y}$. □ □

4.6.3 Completeness of Minimal OCD

We divide the proof of the completeness of our definition of minimality for OCDs in three parts: first, in the following theorem we prove that attributes list with repeated attributes at the beginning are redundant:

Theorem 22 (Completeness of minimal OCD - 1)**.**

$$\frac{\mathbf{Y} \sim \mathbf{Z}}{\mathbf{XY} \sim \mathbf{XZ}}$$

Proof. By the Shift theorem [SGG12b] and the fact that $\mathbf{X} \leftrightarrow \mathbf{X}$ by Reflexivity (AX1):

$$\frac{\mathbf{YZ} \mapsto \mathbf{ZY} \quad \mathbf{X} \leftrightarrow \mathbf{X}}{\mathbf{XYZ} \mapsto \mathbf{XZY}}$$

by Normalization (AX3) and Replace [SGG12b] $\mathbf{XYXZ} \mapsto \mathbf{XZXY}$. Analogously by the Shift theorem [SGG12b] starting from $\mathbf{ZY} \mapsto \mathbf{YZ}$ we obtain $\mathbf{XZXY} \mapsto \mathbf{XYXZ}$. Thus $\mathbf{XYXZ} \leftrightarrow \mathbf{XZXY}$, i.e. $\mathbf{XY} \sim \mathbf{XZ}$ □

The following theorem proves that attributes list with repeated attributes at the end are also redundant:

Theorem 23 (Completeness of minimal OCD - 2)**.**

$$\frac{\mathbf{X} \sim \mathbf{Y} \quad \mathbf{XZ} \sim \mathbf{Y} \quad \mathbf{X} \sim \mathbf{YZ}}{\mathbf{XZ} \sim \mathbf{YZ}}$$

Proof.

1. using $\mathbf{XY} \leftrightarrow \mathbf{YX}$ and $\mathbf{XZY} \leftrightarrow \mathbf{YXZ}$, by Normalization (AX3) $\mathbf{XZY} \leftrightarrow \mathbf{XZYZ}$ and by Replace [SGG12b] $\mathbf{YXZ} \leftrightarrow \mathbf{XZYZ}$;
2. using $\mathbf{XY} \leftrightarrow \mathbf{YX}$ and $\mathbf{XYZ} \leftrightarrow \mathbf{YZX}$, by Normalization (AX3) $\mathbf{YZX} \leftrightarrow \mathbf{YZXZ}$, by Replace [SGG12b] $\mathbf{YXZ} \leftrightarrow \mathbf{YZX}$ and by Transitivity (AX4) $\mathbf{YXZ} \leftrightarrow \mathbf{YZXZ}$;

By Transitivity (AX4) $\mathbf{YXZ} \leftrightarrow \mathbf{XZYZ}$ and $\mathbf{YXZ} \leftrightarrow \mathbf{YZXZ}$ imply $\mathbf{XZYZ} \leftrightarrow \mathbf{YZXZ}$, i.e. $\mathbf{XZ} \sim \mathbf{YZ}$. □

Finally, the following theorem proves that attributes list with repeated attributes in the middle are redundant:

Theorem 24 (Completeness of minimal OCD - 3)**.**

$$\frac{\mathbf{X} \sim \boldsymbol{M} \quad \mathbf{XY} \sim \boldsymbol{M} \quad \mathbf{X} \sim \boldsymbol{MY} \quad \mathbf{XY} \sim \boldsymbol{MN}}{\mathbf{XY} \sim \boldsymbol{MYN}}$$

Proof. 1. from $\mathbf{XY} \sim \mathbf{MN}$, by Normalization (AX3) $\mathbf{XYMYN} \leftrightarrow \mathbf{MNXY}$;

2. from $\mathbf{XY} \sim \mathbf{M}$ and $\mathbf{X} \sim \mathbf{MY}$, using $\mathbf{X} \sim \mathbf{M}$ and Replace [SGG12b] we get $\mathbf{MYX} \leftrightarrow \mathbf{XYM}$ and $\mathbf{MXY} \leftrightarrow \mathbf{MYX} \leftrightarrow \mathbf{XYM}$;

3. from (2), by the Shift theorem [SGG12b] with $\mathbf{MY} \leftrightarrow \mathbf{MY}$ and $\mathbf{MNXY} \leftrightarrow \mathbf{XYMMYN}$ we get $\mathbf{MYMNXY} \leftrightarrow \mathbf{MYXYMMYN}$;

4. by Normalization (AX3) $\mathbf{MYMNXY} \leftrightarrow \mathbf{MYNXY}$;

5. from $\mathbf{MYXYMMYN}$, using $\mathbf{MYX} \leftrightarrow \mathbf{XYM}$ and Normalization (AX3) we get $\mathbf{XYMYMYN}$ and finally $\mathbf{XYMYN}$;

From points (3), (4) and (5) we finally get $\mathbf{MYNXY} \leftrightarrow \mathbf{XYMYN}$, i.e. $\mathbf{XY} \sim \mathbf{MYN}$. □

4.7 The OCDdiscover Algorithm

We present now the details of our algorithm, called OCDDISCOVER, by first examining its search strategy to cover all the possible combinations and then presenting an implementation in pseudo-code.

4.7.1 Column Reduction

Given that the search space grows with the number of columns, we start our discovery algorithm focusing on the columns showing special properties and we perform two operations: (a) removal of constant columns; (b) reduction of order-equivalent columns. The dependencies provided by these operations are an integral part of the results provided by our algorithm.

Removal of constant columns. Constant columns generate a huge amount of ODs; in fact, over an instance $\boldsymbol{r}$ a constant column C is ordered by any other attribute list $\mathbf{X}$.[1] Thus, we remove all constant columns and we collect the corresponding dependencies.

Reduction of order-equivalent columns. Order-equivalent columns as $A \leftrightarrow B$ describe a relation in which both the directions of the order dependency hold. By the Replace theorem (Theorem 6, [SGG12b]), we can replace any order dependency where A appears with another dependency with any instance of A replaced with B, that is:

$$\mathbf{X}A\mathbf{Y} \mapsto \mathbf{M}A\mathbf{N} \Leftrightarrow \mathbf{X}B\mathbf{Y} \mapsto \mathbf{M}B\mathbf{N}$$

We check any combination of order-equivalent dependencies, i.e. for all $A, B \in \mathcal{U}$ we verify the validity of $A \mapsto B$ and $B \mapsto A$, and we build the equivalence classes of columns using the Tarjan algorithm [Tar72].

From each of these equivalence classes we choose a representative and we remove all other columns. We store this information to later recover the redundant dependencies.

4.7.2 Search Tree

We use a breadth-first search strategy for identifying OCD relations in $\boldsymbol{r}$; in this way, shorter minimal dependencies are discovered before longer ones. At the first level, we consider the set of all pairs of single attributes. Given that OCDs are commutative, we build this set by enumerating all the attributes with $A_1, A_2, \ldots, A_n$ and taking all the pairs (A_i, A_j) such that $\{(A_i, A_j) \mid A_i, A_j \in \mathcal{U}, i < j\}$.

[1] If C is constant column, the following property holds for any tuple p, q in any instance $\boldsymbol{r}$ of $\boldsymbol{R}$: $p\mathbf{x} \leq q\mathbf{x} \Rightarrow p_C = q_C$, where the second part of the implication is always true by definition of constant column.

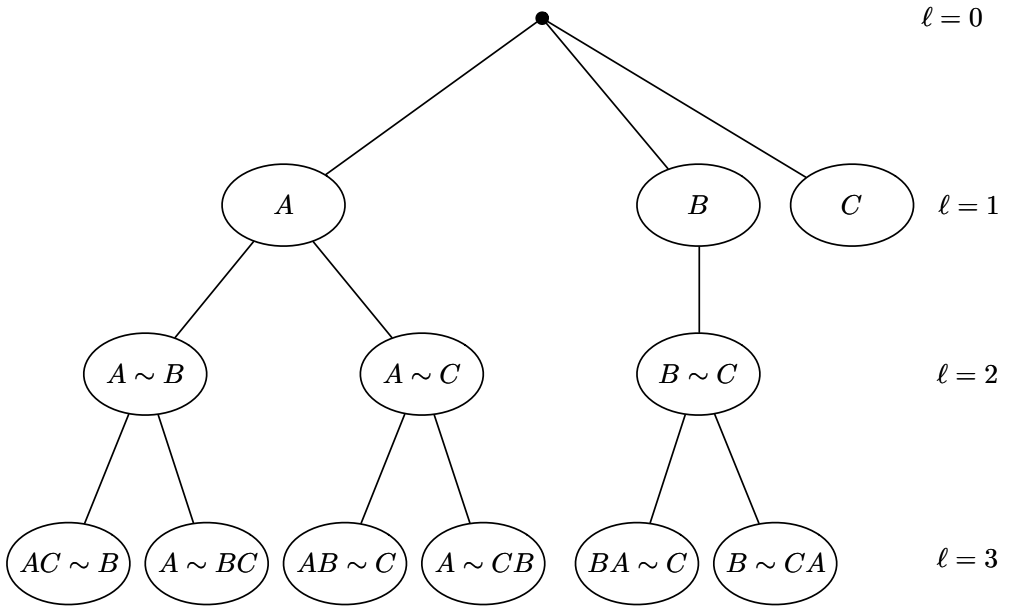


Figure 4.1: Permutation tree for a table with $n = 3$ attributes.

Figure 4.1 shows the tree $\mathcal{T}$ of generated candidates for a relation $\boldsymbol{r}$ with attributes $\mathcal{U} = \{A, B, C\}$ where all possible candidates are generated.

Each OCD candidate $\mathbf{X} \sim \mathbf{Y}$ is checked for order compatibility; we are then confronted with two possibilities:

- if the candidate is not order compatible, we do not generate any other candidate starting from it, as stated by Theorem 19 in Section 4.6.1;
- if the candidate is order compatible, we generate new OCD candidates in the following way: for each attribute not already present in the OCD, for each $A \in \mathcal{U} \setminus \{\mathcal{X} \cup \mathcal{Y}\}$, we add it to the right of each attribute list, i.e. $\mathbf{X}A \sim \mathbf{Y}$ and $\mathbf{X} \sim \mathbf{Y}A$, then we apply further pruning rules as explained in Section 4.7.2.1.

4.7.2.1 Pruning Rules

When we find a new OCD $\mathbf{X} \sim \mathbf{Y}$, we further check the validity of the OD $\mathbf{X} \mapsto \mathbf{Y}$ and $\mathbf{Y} \mapsto \mathbf{X}$.

From Theorem 21 we derive the following pruning rules:

- if $\mathbf{X} \mapsto \mathbf{Y}$ we do not generate the candidates of the form $\mathbf{XZ} \overset{?}{\sim} \mathbf{Y}$, i.e. the left-hand children candidates of $\mathbf{X} \sim \mathbf{Y}$ are pruned;
- if $\mathbf{Y} \mapsto \mathbf{X}$, we do not generate the candidates of the form $\mathbf{X} \overset{?}{\sim} \mathbf{YZ}$, i.e. the right-hand children candidates of $\mathbf{X} \sim \mathbf{Y}$ are pruned;

If both dependencies are valid we prune all the subtree from the given OCD candidate.

With reference to Figure 4.1, if the order compatible dependency $A \sim B$ is valid:

- if $A \mapsto B$, the children candidate $AC \overset{?}{\sim} B$ is pruned;
- if $B \mapsto A$, the children candidate $A \overset{?}{\sim} BC$ is pruned.

4.7.2.2 Parallelizability

OCDDISCOVER explores the tree of candidates breadth-first. Each branch of the tree can be visited independently, since each OCD candidate is independent from the others, furthermore, for each level a candidate is generated only if its father in the tree was a valid order dependency. We exploit this structure to parallelize the execution of OCDDISCOVER by assigning candidates from different branches to different queues, each queue is then processed by a different thread. With reference to Algorithm 4, if we have K cores available, we can have K independent subtrees $\mathcal{T}_\ell^1, \mathcal{T}_\ell^2, \ldots, \mathcal{T}_\ell^K$ (lines 7 — 12) each one containing the OCD candidates belonging to a different branch of the tree. The number of queues used by the algorithm can be chosen as a run-time parameter provided by the user.

4.7.3 Order Checking

One of the most important steps in our approach is the check of order compatibility candidates.

Single check. Given an OCD candidate in the form $\mathbf{X} \overset{?}{\sim} \mathbf{Y}$, we need to verify if it holds. The general definition of order compatibility states that $\mathbf{X} \sim \mathbf{Y} \equiv \mathbf{XY} \leftrightarrow \mathbf{YX}$, i.e. $\mathbf{X}$ and $\mathbf{Y}$ are order compatible if $\mathbf{XY} \mapsto \mathbf{YX}$ and $\mathbf{YX} \mapsto \mathbf{XY}$; however, with the

following theorem, we can reduce the problem of checking the validity of an OCD to a single check.

Theorem 25. $\mathbf{XY} \mapsto \mathbf{YX}$ *is valid iff* $\mathbf{X} \sim \mathbf{Y}$.

Proof.

$\Rightarrow$ we have to prove that $\mathbf{XY} \mapsto \mathbf{YX} \Rightarrow \mathbf{YX} \mapsto \mathbf{XY}$. If, by contradiction, $\mathbf{YX} \nmapsto \mathbf{XY}$, then:

$$\exists\ p, q \mid p_{\mathbf{YX}} \leq q_{\mathbf{YX}} \Rightarrow p_{\mathbf{XY}} > q_{\mathbf{XY}} \tag{4.15}$$

thus since $p_{\mathbf{XY}} > q_{\mathbf{XY}}$ we can distinguish two cases:

- if $p_{\mathbf{X}} > q_{\mathbf{X}}$, we can conclude that:

$$q_{\mathbf{XY}} < p_{\mathbf{XY}} \Rightarrow q_{\mathbf{YX}} > p_{\mathbf{YX}}$$

  thus $\mathbf{XY} \nmapsto \mathbf{YX}$;

- if $p_{\mathbf{X}} = q_{\mathbf{X}} \wedge p_{\mathbf{Y}} > q_{\mathbf{Y}}$, we always obtain a contradiction with the condition expressed in Eq. 4.15;

thus both $\mathbf{XY} \mapsto \mathbf{YX}$ and $\mathbf{YX} \mapsto \mathbf{XY}$ are valid and $\mathbf{X} \sim \mathbf{Y}$;

$\Leftarrow$ by definition if $\mathbf{X} \sim \mathbf{Y}$ then both $\mathbf{YX} \mapsto \mathbf{XY}$ and $\mathbf{XY} \mapsto \mathbf{YX}$ are valid;

□

Checking with Indexes. To compute if an order dependency candidate holds, we sort the relation by the left hand-side attributes of the candidate. We build an in index that contains only the position of each tuple in the order in which it appears. Then, we iterate over the tuples following this index and we check if the attributes on the right-hand side violate the ordering, specifically if we detected a pair of tuples forming a swap. We break the detection loop as soon as we find a violation and return true otherwise. In the worst case the number of comparison to be made is $O(m + m \log m)$ where m is the total number of tuples in the relation.

4.7.3.1 `NULL` Values

In real-world datasets, and in many of the test cases that will be analyzed in Section 4.8, data contains `NULL` values, which destroy the total ordering assumption because they can not be compared with the other values. We use the standard SQL semantics given by `set ansi nulls ON`, i.e. `NULL` equals `NULL`, and `NULLS FIRST` for sorting.

However, `NULL` values destroy the total ordering over an attribute, because they can not be compared with the other attributes.

Furthermore, `NULL` values can be interpreted following different semantics that we exemplify here using data in Table 4.6.

	A	B	C
t_1	1	4	8
t_2	2	5	?
t_3	3	6	10

A	B	C
1	4	8
4	4	?
3	6	10

Table 4.6: These two tables produce different results when testing the OCD candidate $A \overset{?}{\sim} BC$ and using different

`NULL` is equal to `NULL`. `NULL` values can be considered equal to themselves and they are order before any other value, this is equivalent to the SQL settings given by `set ansi nulls ON` and `NULLS FIRST`. With reference to the example, in Table 4.6 (a) the order dependencies $A \mapsto B$ and $B \mapsto A$ are valid; while in Table 4.6(b) the only valid order dependency is $C \mapsto B$, in this case the following relations are true:

- $t_{2,C} := ? \leq t_{1,C} := 8 \leq t_{3,C} := 10$
- $t_{2,B} := 4 \leq t_{1,B} := 4 \leq t_{3,B} := 6$

This is the standard semantic assumed by the previous work on order dependencies [LN16, SGG+17]. We adopt the same for our experimental evaluation in Section 4.8.

Ignoring `NULL` values. `NULL` values are considered as unknown values. Under this semantic if a comparison involves a `NULL` value the tuple will be ignored. Also in this case, in Table 4.6 (a) the order dependencies $A \mapsto B$ and $B \mapsto A$ are valid. Furthermore several other dependencies are valid: $A \mapsto C$, $A \mapsto C$ (and the corresponding ones with $A \mapsto B$). In fact, in this case any comparison involving $t_{2,C}$ is skipped. Unfortunately,

this semantics violates the Transitivity axiom (AX4), in fact, in Table 4.6 (b), in checking the order depencies $B \mapsto C$ and $C \mapsto A$, t_2 is ignored because of the `NULL` value and both dependencies hold. However, since $B \mapsto A$ by Transitivity (AX4) the $B \mapsto A$ would be valid, which is false. Using this sematics for order dependency detection would require a more in-depth study on the consequences of breaking the Transitivity axiom.

4.7.4 Description of the Algorithm

Algorithm 4 is the main algorithm implementing our approach. The input is given by the instance of relational data $\boldsymbol{r}$ and its set of attributes $\mathcal{U}$.

Algorithm 4 OCDDISCOVER

Input: $\boldsymbol{r}$: a relational instance
Input: $\mathcal{U}$: the set of attributes associated with $\boldsymbol{r}$

```
1:  function OCDDISCOVER(r,U)
2:      ℓ ← 2
3:      U' ← COLUMNSREDUCTION(U)
4:      T_1 ← {(A, B) | A, B ∈ U', B ≻ A}
5:      while T_ℓ ≠ ∅ do                                   ▷ Main loop
6:          T_{ℓ+1} ← ∅
7:          for each (X, Y) ∈ T_ℓ do
8:              if CHECKCANDIDATE(XY, YX, r) then
9:                  emit X ~ Y
10:                 T_{ℓ+1} ← T_{ℓ+1} ∪ GENERATENEXTLEVEL(X, Y, U')
11:             end if
12:         end for
13:         ℓ ← ℓ + 1
14:     end while
15: end function
```

In Line 3, function COLUMNREDUCTION() (not shown here) is called to apply the operations described in Section 4.7.1: the removal of constant attributes and the reduction of order-equivalent columns. This function prints a list of order-equivalence relations and a list of constant attributes and returns a reduced set of attributes $\mathcal{U}'$ where (a) the constant columns are removed; and (b) for each class of order-equivalent attributes one representative is chosen.

The initial tree of OCD candidates is built in Line 4; by the commutativity of OCDs, only half of the combinations are added. Figure 4.1 shows that the second level of the tree ($\ell = 2$) contains only the initial candidates $A \overset{?}{\sim} B$, $A \overset{?}{\sim} C$ and $B \overset{?}{\sim} C$.

Then, the algorithm continues with the main loop where each OCD candidate $\mathbf{X} \overset{?}{\sim} \mathbf{Y}$ is tested against the data in $\boldsymbol{r}$ in the form of an OD candidate $\mathbf{XY} \overset{?}{\mapsto} \mathbf{YX}$ using the function checkCandidate(), which is described in Algorithm 5.

The function checkCandidate() iterates over the index built on the left-hand side of the candidate with generateIndex() and checks that the values over the attributes in the right-hand side are in the same order. The loop is terminated early if a violation is detected.

Algorithm 5 checkCandidate

Input: $\mathbf{X}, \mathbf{Y}$: an OD candidate
Input: $\boldsymbol{r}$: the instance of relational data
Output: **true** if $\mathbf{X} \mapsto \mathbf{Y}$, **false** otherwise.

```
1:  function checkCandidate(X, Y, r)
2:      l_r ← len(r)
3:      index ← generateIndex(X, Y, r)
4:      for i ← 1 to l_r − 1 do
5:          for each A ∈ Y do
6:              if r[index[i], A] > r[index[i + 1], A] then
7:                  return false
8:              else if r[index[i], A] < r[index[i + 1], A] then
9:                  return true
10:             end if
11:         end for
12:     end for
13:     return true
14: end function
```

If the candidate is a valid OCD, we emit it as a result and generate the new candidates through generateNextLevel(), which is described in Algorithm 6.

The function generateNextLevel() builds a set containing all the OCD candidates of the form $\mathbf{X}A \sim \mathbf{Y}$ and $\mathbf{X} \sim \mathbf{Y}A$, where A is an attribute that does not already belong to the lists $\mathbf{X}$ and $\mathbf{Y}$, this corresponds to creating a new level of the tree presented in Section 4.7.2 using the pruning rules of Section 4.7.2.1. The function further checks if the ODs $\mathbf{X} \overset{?}{\mapsto} \mathbf{Y}$ or $\mathbf{Y} \overset{?}{\mapsto} \mathbf{X}$ hold. If so, it applies the pruning rules and returns the remaining candidates. We emit the valid ODs found in Lines 9 and 16 of Algorithm 6.

The new candidates are added to the queue of candidates to check for the next level in line 10 of Algorithm 4. The loop terminates when there are no candidates left.

Algorithm 6 GENERATENEXTLEVEL

Input: $\mathbf{X}, \mathbf{Y}$: an OCD candidate
Input: $\mathcal{U}'$: the set of reduced attributes of relation $\boldsymbol{R}$
Output: C: the candidate OCD generated from $\mathbf{X} \sim \mathbf{Y}$

```
1:  function GENERATENEXTLEVEL(X, Y, U')
2:      C ← ∅
3:      A+ ← U' − SET(X) − SET(Y)
4:      if ¬CHECKCANDIDATE(X, Y, r) then                    ▷ X ↛ Y
5:          for each A ∈ A+ do
6:              C.add((XA, Y))
7:          end for
8:      else                                                ▷ X ↦ Y
9:          emit X ↦ Y
10:     end if
11:     if ¬CHECKCANDIDATE(Y, X, r) then                    ▷ Y ↛ X
12:         for each A ∈ A+ do
13:             C.add((X, YA))
14:         end for
15:     else                                                ▷ Y ↦ X
16:         emit Y ↦ X
17:     end if
18:     return C
19: end function
```

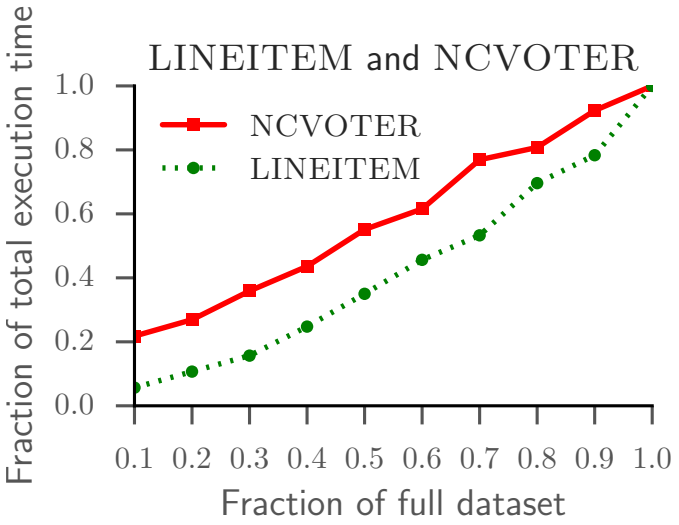


Figure 4.2: Normalized execution times for row scalability.

4.8 Experimental Evaluation

In this section we evaluate the results of our approach, with particular emphasis on its scalability in the number of rows and columns, and we compare it with previous work on order dependency detection. We analyze the results of OCDDISCOVER over 6 real-world

Dataset properties			FASTFDS [WGR01]	ORDER [LN16]		FASTOD [SGG+17]			OCDDISCOVER			
Dataset	$\|r\|$	$\|\mathcal{U}\|$	$\|\mathcal{F}_d\|$	$\|\mathcal{O}_d\|$	**time (ms)**	$\|\mathcal{O}_d\|$	$\|\mathcal{F}_d\|$	**time (ms)**	$\|\mathcal{O}_d\|$	$\|\mathcal{O}_c\|$	**time (ms)**	**checks**
DBTESMA	250,000	30	89,571	—*	5 h*	400	89,571	4,641,485	138	0	**337,289**	4,118
DBTESMA_1K	1000	30	11,099	—*	5 h*	30	11,099	5,799	138	0	**1,835**	4,118
FLIGHT_1K	1,000	109	—*	—†	—†	—*	—*	5 h*	3,216,069*	29,404,555*	5 h*	7,473,951
HEPATITIS	155	20	8,250	0	**182**	32,717	8,250	211,903	0	5	361	556
HORSE	300	29	128,727	31	46,907	—*	—*	5 h*	31	7	**618**	1,185
LETTER	20,000	17	61	0	**1,215**	—*	—*	5 h*	0	0	1,720	272
LINEITEM	6,001,215	16	—*	1	**982,075**	—*	—*	5 h*	1	0	1,039,517	255
NCVOTER_1K	1,000	19	758	18	**796**	2,333	758	90,000	18	1	872	338
NO	5	2	1	0	323	0	1	24	0	0	**4**	2
NUMBERS	7	4	4	0	331	6	4	325	0	0	**28**	12
YES	5	2	0	0	329	1	0	28	1	1	**3**	2

Table 4.7: Datasets and execution statistics for the OCDDISCOVER, ORDER [LN16], and FASTOD [SGG+17] algorithms. "*" indicates that the execution has reached the time limit of 5 hours, while "†" that it has exceeded the memory limit of 110GB. When the time limit is reached, for OCDDISCOVER we present partial results.

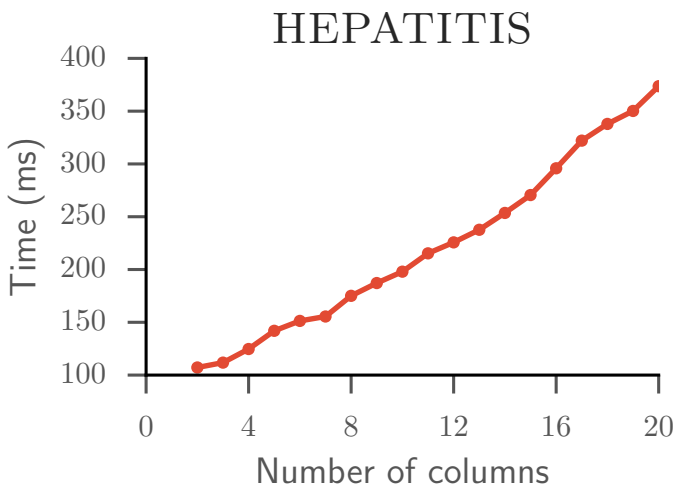


Figure 4.3: Execution times for column scalability for `HEPATITIS`.

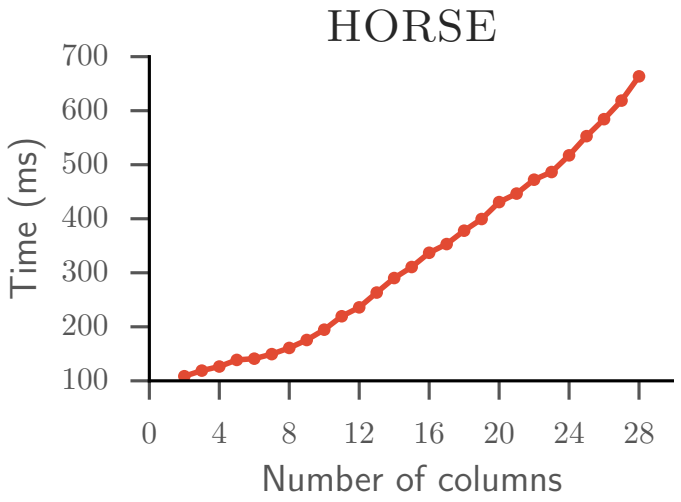


Figure 4.4: Execution times for column scalability for `HORSE`.

datasets and 5 synthetic datasets.

Our algorithm is implemented in Java 1.7 and is designed to work on the Metanome data profiling framework [PBF+15] as a multi-threaded program.

All experiments were run on a `i686` Intel Xeon E52440 2.40 GHz machine with 12 cores in hyper-threading and 128 GB RAM, over a Linux kernel `v4.15.0`. The execution environment is a 64-bit Oracle JDK version `1.8.0_171`, with the JVM heap space limited to 110 GB.

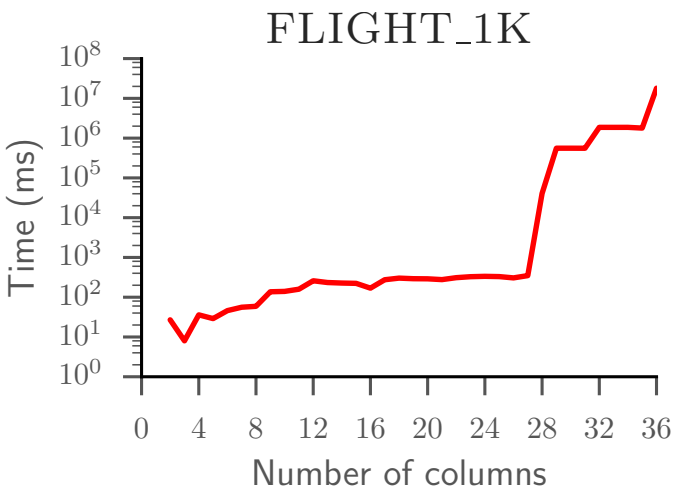


Figure 4.5: Execution times for column scalability for `FLIGHT_1K`.

A	B	C	D
1	3	1	1
2	2	3	2
2	3	2	2
2	5	2	2
3	1	2	3
4	4	4	2
4	5	3	2

Table 4.8: `NUMBERS` dataset.

4.8.1 Datasets

We use the datasets provided by the Information Systems Group of Hasso-Plattner-Institut.[2] These datasets are the same used by the previous work on order dependency discovery by Langer and Naumann [LN16]. We have also created three simple additional synthetic datasets, called `YES`, `NO`, and `SIMPLE` created to highlight the differences of our approach with previous works. In particular, `YES` and `NO` reproduce, respectively, the examples in Tables 4.5 (a) and 4.5 (b), while `SIMPLE` is shown in Table 4.8.

Table 4.7 presents the datasets and their properties; for each dataset, the table reports: the dataset name, the number of rows $|r|$, the number of attributes $|\mathcal{U}|$, the number of functional dependencies discovered by the FASTFDS algorithm [WGR01] $|\mathcal{F}_d|$, the number of ODs discovered $|\mathcal{O}_d|$ by ORDER. For FASTOD we provide: (a) the number of FDs discovered $|\mathcal{F}_d|$, (b) the number of ODs discovered $|\mathcal{O}_d|$. For OCDDISCOVER we provide: (a) the number of OCDs discovered $|\mathcal{O}_c|$, which are missed by ORDER [LN16], since they

[2] https://hpi.de/naumann/projects/repeatability/data-profiling/fd-algorithms.html

are order dependencies with repeated attributes; (b) the number of ODs discovered $|\mathcal{O}_d|$; and (c) the total number of dependency candidates checked during the execution of the algorithm.

Execution time is averaged across 5 independent runs and we set a time threshold at 5 hours. When the time limit is reached, for FASTOD we are unable to present the number of dependencies discovered so far, while for OCDDISCOVER we report the number of dependencies discovered and the number of checks made until the limit.

4.8.2 Comparison with Previous Work

We discuss the results of the extensive comparison with the previous state-of-the-art algorithms for detecting order dependencies. The code used for the comparison has been provided by the respective authors.[3]

To compare our algorithm with ORDER and FASTOD we need to transform OCDs back to ODs. In fact, in a relation $\boldsymbol{R}$ over attributes A, B and C where $A \leftrightarrow B$, $A \sim C$ and $B \sim C$, the set of OCDs $\mathcal{O} = \{A \sim B,\ A \sim C,\ B \sim C\}$ is minimal following Definition 16. However, COLUMNSREDUCTION() would discover the order-equivalence $A \leftrightarrow B$ and choose one attribute as a representative, e. g. A. Thus OCDDISCOVER would return as valid OCDs only $\mathcal{O} = \{A \sim C\}$. From this information we infer the remaining dependency $B \sim C$ using the axioms $\mathcal{J}_{\text{OD}}$. We perform this expansion and compare the results produced by OCDDISCOVER, ORDER and FASTOD. The function performing the expansion is not shown in Algorithm 4, but the times reported in Table 4.7 include it. This step did not impact the running time of OCDDISCOVER.

4.8.2.1 Comparison with Langer and Naumann [LN16]

For ORDER, dependencies are considered to be *completely non-trivial*, if their left- and right-hand side attribute lists are disjoint. However, we argue that limiting the discovery of order dependencies to candidates where the left-hand side and the right-hand side are completely disjoint gives incomplete results. Our algorithm, instead, is complete.

[3] the source code for ORDER is available at: https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html#c168192 while the source of FASTOD has been provided to us by the authors through direct communication.

Following Theorem 20 in Section 4.6.2, order dependencies of the form $\mathbf{XY} \mapsto \mathbf{Y}$ can be inferred from order compatibility dependencies of the form $\mathbf{X} \sim \mathbf{Y}$. Note that this dependencies have repeated attributes between its left- and right-hand sides.

We show the difference between our apporach and previous work with the `YES` and `NO` datasets. As reported in Table 4.7, the ORDER algorithm does not find any order dependency in either of the `YES` and `NO` datasets. OCDDISCOVER, instead, finds correctly the order compatibility dependency $A \sim B$, i.e. the order equivalence $AB \leftrightarrow BA$, in `YES`.

Our approach detects all the dependencies found by ORDER, and additional dependencies on `HEPATITIS`, `HORSE`, and `NCVOTER_1K`. For `FLIGHT_1K` and `NCVOTER_ALLC` we found several dependencies but we were not able to compare the results with ORDER because the latter does not report the discovered dependencies when the time limit is reached.

Provided that the order compatibility dependencies found by OCDDISCOVER are translated to the corresponding OD in the form $\mathbf{XY} \mapsto \mathbf{Y}$, OCDDISCOVER effectively discovers a minimal set of ODs even following the definition of minimality provided by Langer and Naumann [LN16].

When a candidate dependency is found to be false, pruning rules are applied. For this reason, notwithstanding the factorial dimension of the search space, datasets with several columns, such as datasets `HEPATITIS` and `HORSE`, are successfully and completely tested. When pruning cannot be applied, the generation of candidates grows – e.g., more than 7 million candidates are generated in `FLIGHT_1K`. For this dataset, OCDDISCOVER detects more than 32 million ODs. In this particular case, the number of checks is smaller than the number of discovered dependencies because we also count the dependencies inferred from constant and order-equivalent columns reported by the COLUMNSREDUCTION() function.

Furthermore, Table 4.7 shows that using order compatibility dependencies does not hinder the performance of the detection. In all dataset tested, the performance of our algorithm with respect to the execution time is comparable to ORDER and in some cases we obtain significant speedups up to a factor of 75, as in the case of `HORSE`.

4.8.2.2 Comparison with Szlichta et al. [SGG+17]

As shown in Table 4.7, OCDDISCOVER and FASTOD compute a different number of order dependencies. We claim that this difference, which also affects also the results published in [SGG+17], is due to an implementation error in the code. Table 4.8 presents an instance of a relational table where the implementation of FASTOD we received finds several order dependencies that are not actually present in the data, e.g. $[B] \mapsto [AC]$. Other datasets were also affected by this issue, but, unfortunately, we were not able to isolate and resolve the root cause of this incorrect behavior. In addition, FASTOD considers all columns as if they contain data of type `String`, thus using a lexicographical ordering, while ORDER and OCDDISCOVER perform type inference over the datasets provided, and use the natural ordering for real and integer numbers. We have also implemented for OCDDISCOVER the possibility of forcing lexicographical ordering, but we do not report these results since this change does not affect the execution time of our approach.

Furthermore, we would like to point out that for the experiments reported in [SGG+17], for the scalability experiments a trimmed-down version of the datasets has been used, for this reason we have both the `DBTESMA` and `DBTESMA_1K` datasets. For the row scalability experiments, the work does not specify how the trimming of the dataset was performed, but only states that for the scalability on the columns we were not able to reproduces those results, with the exception of `HEPATITIS`, `FLIGHT_1K`, and `NCVOTER_1K`, which were used in full.

We can compare the performance of OCDDISCOVER and FASTOD over those datasets. As shown in Table 4.7, OCDDISCOVER gains significant speed-ups. This highlights the fact that, while the worst-case complexity of OCDDISCOVER is $(O)(|r|!)$, which is greater than that of FASTOD, $O(2^{|r|})$, the execution time depends on the actual number of dependencies contained in the dataset.

4.8.3 Performance

In the following, we analyze the scalability of our approach with respect to the number of rows, the number of columns, and the number of parallel threads used.

4.8.3.1 Scalability in the number of rows

We performed our analysis on the synthetic dataset `LINEITEM` with $6,001,215$ rows and 16 columns. We also test the algorithm on the `NCVOTER` dataset. This dataset has $938,084$ rows and 94 columns, but since our algorithm did not terminate on the full datasets, we consider only a subset of 20 randomly chosen columns. Figure 4.2 shows the results for the scalability experiment on `LINEITEM` and `NCVOTER`. Ten samples of the original dataset have been created, ranging from 10% to 100% of the rows with a step of 10%. Five repetitions have been performed for each of the samples and the average is reported. Variance is very small and thus not shown.

The experiments show that the algorithm scales almost linearly with the number of rows, and it is able to find a complete set of OCDs over datasets with millions of rows.

The execution time would be expected to grow log-linearly with respect to the number of rows, due to the indexing phase; but an increasing number of rows may correspond to a smaller number of dependencies; thus, the pruning phase could reduce the amount of checks to be computed.

Previous work, instead, has shown the ability to scale linearly on the number of rows performing the check of dependency candidates with sorted partitions computed from the data. This method could be re-implemented in our approach as well, but it would be out of the scope of this work.

4.8.3.2 Scalability in the number of columns

Scalability over columns is the key challenge in detecting dependencies in relational data since in many cases the dimension of the search scales with the number of columns.

We choose the `HORSE` and `HEPATITIS` datasets, that are well-suited to evaluate the influence of an increasing number of columns, given that their execution completes. We also consider `FLIGHT_1K` that has a very high number of columns and does not terminate.

The evaluation approach is as follows: we start with two random columns from each dataset, and we incrementally add more randomly-chosen columns, until the total number of columns in the dataset is reached.

To avoid skewing the results, we generate 50 samples of each dataset with the process described above and we run our algorithm over these samples. We average the execution time of OCDDISCOVER of each sample with c columns over all the 50 samples.

Figures 4.3 and 4.4 show the results of the column scalability experiment on the `HEPATITIS` and `HORSE` datasets.

Figure 4.5 shows how the algorithm behaves when the number of dependencies discovered in the data grows on a single run. Times on the y-axis are in logarithmic scale. OCDDISCOVER is very susceptible to columns that are quasi-constant, i.e. attributes with very few distinct values, but not constant. In this case, OCDDISCOVER cannot eliminate these columns during the column-reduction phase. As argued in Section 4.7.1, constant columns are ordered by any other attribute; quasi-constant columns are associated with a large number of valid OCDs and consequently the size of the tree to be explored grows enormously.

In fact the slowdown corresponding to the sample with 28 columns in Figure 4.5 is caused by the addition of a column with 3 distinct values. This column appears on the right-hand side of more than 94% of the dependencies found in that sample.

4.8.3.3 Scalability over parallel threads

As describe in Section 4.7.2.2, OCDDISCOVER can be run over multiple threads.

	Time (s) vs number of threads			
Dataset	**1**	**2**	**4**	**8**
`LETTER`	5.7	4.6	3.6	3.4
`LINEITEM`	2,848.8	1,770.0	1,243.5	1,040.0
`DBTESMA`	2,228.9	1,240.0	686.0	414.0

Table 4.9: Execution time of OCDDISCOVER versus number of threads.

Figure 4.6 shows the results of the multithreading scalability experiments on the `LETTER`, `LINEITEM`, and `DBTESMA` datasets. On the y-axis times are normalized to the runtime over a single thread, which is the maximum for each case. Table 4.9 reports the executions times.

As it is shown in Figure 4.6, using multiple parallel threads shortens the execution time of our dependency discovery algorithm. This breadth of this improvement varies

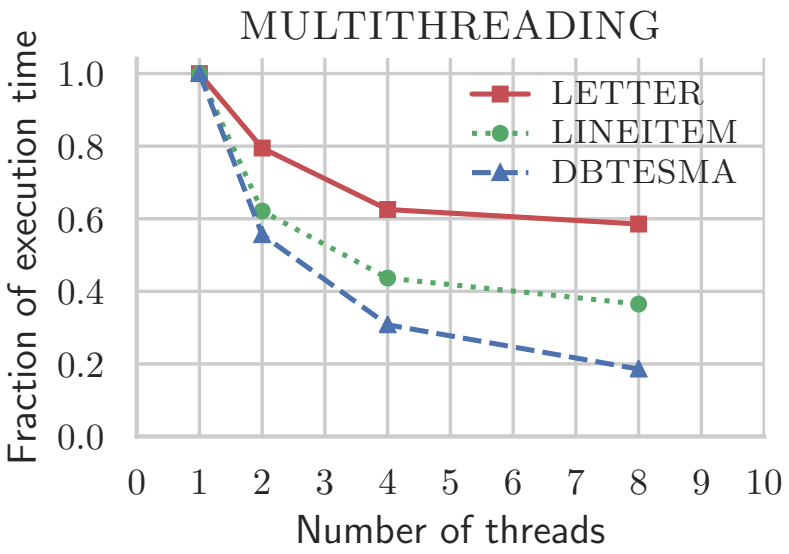


Figure 4.6: Execution times for `LETTER` (red straight line), `LINEITEM` (green dotted line), `DBTESMA` (blue dashed line), when executed over multiple threads, normalized over the runtime over a single thread.

based on the characteristics of each datasets and in particular depends on the of OCD candidates checked.

If we compare `LETTER` and `LINEITEM`, we see that while the number of checks performed on each dataset is comparable (272 for `LETTER` versus 255 for `LINEITEM`), the number of their rows differ by several orders of magnitude ($\sim 20k$ lines for `LETTER`, versus $\sim 6M$ for `LINEITEM`). This implies that checking the validity of an OCD candidate for `LINEITEM` takes longer than checking a candidate for `LETTER`. Thus the relative gain when splitting the work over multiple threads is greater for `LINEITEM`.

Instead, comparing `LINEITEM` and `DBTESMA` we can see that for the latter dataset the number of checks performed is much higher, thus the workload of candidate checking can be spread over multiple threads, leading to greater relative improvements.

4.8.4 Quasi-constant Columns

The quasi-constant column scenario is challenging for our algorithm. We further develop the idea of measuring how varied the values in a column are by measuring its *entropy*.

Definition 26 (Entropy)**.** Given an attribute $A \in \mathcal{U}$ of an instance $\boldsymbol{r}$ of a relation $\boldsymbol{R}$, the entropy of A is defined as:

$$H(A) = -\sum_{[a]} p_{[a]} \log\left(p_{[a]}\right) \tag{4.16}$$

where $[a]$ are the equivalence classes of distinct values in A and $p_{[a]}$ is the probability of extracting an instance of class a, computed as the relative frequency of instances class $[a]$ over the total number of tuples in $\boldsymbol{r}$:

$$p_{[a]} = \frac{|\{t \in \boldsymbol{r} \mid t_A \in [a]\}|}{|\boldsymbol{r}|}$$

For constant columns there is only one equivalence class and $p_{[a]} = 1$, thus $H(A) = 0$. If all values are distinct, for each $[a]$, $|[a]| = 1$ and $p_{[a]} = 1/|\boldsymbol{r}|$:

$$H(A) = -\sum_{[a]} \frac{1}{|\boldsymbol{r}|} \log\left(1/|\boldsymbol{r}|\right) = \log\left(|\boldsymbol{r}|\right)$$

We test the idea that progressively less diverse columns cause the slowdown of OCDDISCOVER by taking the `FLIGHT_1K` dataset and running it over multiple samples build with the following criteria: we calculate the entropy of each column in `FLIGHT_1K` and then we build samples of increasing size in the number of columns by adding progressively the columns with decreasing entropy, i.e. we start with the columns with the greatest number of distinct values and we progressively add columns with less distinct values. Eventually, the constant columns are added.

The result of the execution on OCDDISCOVER over this set of samples is reported in Figure 4.7. With 50 columns the OCDDISCOVER completes in 4 minutes, adding the 51st column the execution time grows by an order of magnitude to over 1 hour. With the addition of the 52nd column the algorithm reaches the time limit of 5 hours. The 50th, 51st and 52nd columns have respectively 4, 2 and 2 distinct values respectively.

With respect to applications, this insight could be exploited to develop algorithms that return results for the most diverse columns, which can be the most interesting with respect to other properties of the data such as unique column combinations (UCC). Detection of unique column combinations is usually performed to find primary keys

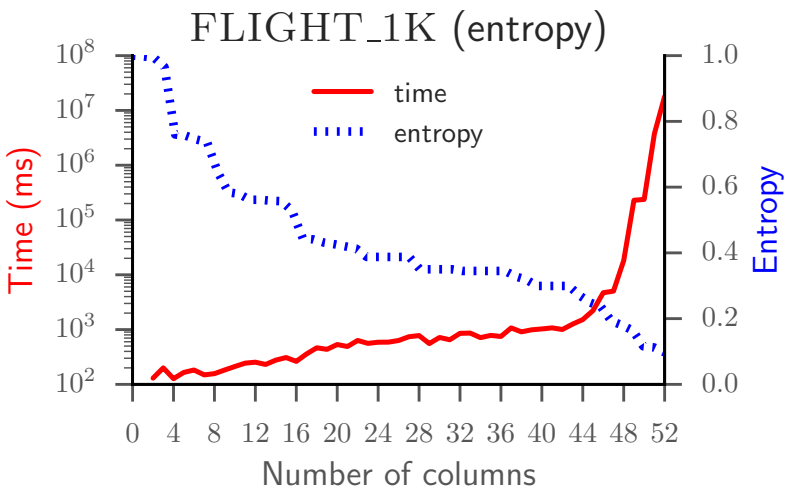


Figure 4.7: Execution times (red straight line) for the `FLIGHT_1K` when adding columns of decreasing entropy (blue dotted line). On the y-axis, times (left-hand side) are in logarithmic scale, entropy (right-hand side) is normalized with respect to the maximum value over the dataset.

candidates that may be also interesting candidates from the point of view of ordering and query optimization.

In summary, the column scalability experiments show that OCDDISCOVER can find a complete set of ODs over datasets with tens of columns. Furthermore, OCDDISCOVER can be easily adapted to perform the detection over a set of interesting columns, where the interestingness of an attribute can be determined providing a function measuring the properties chosen by the user.

4.9 Summary

In this chapter, we presented a novel method for discovering order dependencies (ODs). We based our approach on the fact that an order dependency is valid if and only if both a functional dependency (FD) and an order compatibility dependency (OCD) are valid. Thus, we designed a novel and efficient algorithm – called OCDDISCOVER – where the search for ODs is guided by checking the validity of OCDs. Our approach outperforms existing two state-of-the-art algorithms, ORDER [LN16] and FASTOD [SGG+17] with respect to ORDER, we are complete, meaning that we detect order dependencies that are ignored. We have shown that these dependencies cannot be inferred by other detected dependencies. While the worst-case complexity of OCDDISCOVER is greater than FASTOD, the execution time on real datasets depends on the actual number of dependencies

found, thus we are able to outperform FASTOD. Furthermore, we presented an extensive set of experiments that illustrate that our approach can be executed in parallel over multiple threads. We have also suggested that considering the entropy of attributes can lead to further developments in discovery the most interesting order dependencies.

Chapter 5

Extracting Historical and Current Event from Wikipedia

In previous chapters, we introduced two techniques oriented at the extraction of information from relational data. In this chapter, we deal with the extraction of information from unstructured data sources, focused our attention on the extraction of events from text. We introduce a novel event extraction technique that is based on the idea that set of entities that have a mutual connection within a similar period of time produce an indication for something happened between those entities. We propose a simple but effective model that handle the discovery of historical events from textual sources. While this approach could be applied over several datasets, we focus our attention on Wikipedia, the world largest encyclopedia available. This poses many challenges not only in terms of efficiency and scalability but also of quality. We experimentally evaluate our approach in terms of the quality of the extracted events. We propose an extensive evaluation, where we compare our results against several existing event collections and knowledge bases and additional we conduct a user study to validate our event sets.

5.1 Contributions

Our specific contributions are as follows:

- We propose a novel model for event extraction from Wikipedia that is based on temporal co-reference connectivity. In contrast to online monitoring and specific

type event detection techniques, our methods are able to extract a variety of highly heterogeneous events encoded in the Wikipedia text.

- We formally define the problem, providing an event model that is flexible but also expressive in terms of the whole existing representable events.
- We describe data structures and algorithms for the efficient and effective implementation of our event extraction algorithm that partition the co-reference graph into smaller portions and apply the checks for possible events within them.
- We perform an extensive experimental evaluation on the quality of the results and we show that our set of events is complementary to those in existing event datasets, and also that 88% of the events we identify are indeed real world events as confirmed by real users.
- We describe extensions and applications that our approach can find in exploring the history of an entity over time.

5.2 Outline

The chapter is organized as follows. In Section 5.3 we present a motivating example used through the rest of the chapter. Section 5.4 formally defines our data model and our problem. Section 5.5 introduces the principles that guide our event extraction technique, and Section 5.6 and Section 5.7 provide more details about event extraction technique. Finally, Section 5.8 presents the experimental evaluation.

5.3 Motivating Example

To illustrate this novel definition of a real world event, consider the three Wikipedia pages of DNA, James Watson and Francis Crick, a fraction of which are depicted in Figure 5.1. In the DNA page, we see a reference to James Watson and one to Francis Crick, both of them close to the indicated year 1953. In the James Watson page, we have a reference to DNA and a reference to Francis Crick, with the year 1953 mentioned again. Finally, in the Francis Crick page we find a reference to DNA and a reference to James Watson, with the indication 1953. This mutual reference (co-reference) between

each pair of these three entities with all of them around the year 1953 indicates that something important has happened in 1953 that involved all three entities. Indeed, it was in 1953 when J. Watson and F. Crick discovered the DNA molecular structure. This event, alongside many others of this kind, is not explicitly declared as an event in the Wikipedia content, neither it is identified by the existing information extraction algorithms that populate the flagship knowledge bases of YAGO and DBpedia. Clearly, such knowledge bases could benefit from such a mechanism that would allow them to enrich their contents.

5.4 Problem Statement

Given an underlying corpus of documents annotated with temporal expressions and entity mentions, our goal is to extract historical and contemporary events based on temporal entity co-references. To this end, we first define general concepts such as the time domain and an event and then formally define our problem.

Assuming an ordered set $\mathcal{H}$ of primitive temporal moments, a *time* domain is the set $\mathcal{T}=\{[t_s, t_e] \text{ — } t_s \epsilon \mathcal{H} \wedge t_e \epsilon \mathcal{H} \wedge t_s < t_e\}$. A textual characterization may often be used as a shorthand for the two temporal moments of an element in $\mathcal{T}$. For instance, we may say *Mar'91* to indicate the time [1991-03-01, 1991-03-31].

We also assume the existence of an infinite set of entities $\mathcal{E}$ modeling real-world objects like persons, organizations, locations, etc. Entities may relate to each other, and a relationship has some temporal validity. Furthermore, there may be many different reasons why two entities are related, and we can assume that a relationship has also

Entity	**An extract from the entity's Wikipedia page**
DNA	s_1 : Its molecular structure was identified by **James Watson** and **Francis Crick** in 1953, whose model-building efforts were guided by X-ray diffraction data acquired by Rosalind Franklin. ...
James Watson	s_2 : James Dewey Watson (born April 6, 1928) is an American molecular biologist, geneticist and zoologist, best known as one of the co-discoverers of the structure of **DNA** in 1953 with **Francis Crick**. ...
Francis Crick	s_3 : Francis Harry Compton Crick (8 June 1916 – 28 July 2004) was a British molecular biologist, biophysicist, and neuroscientist, most noted for being a co-discoverer of the structure of the **DNA** molecule in 1953 with **James Watson**. ...

Figure 5.1: The discovery of DNA in three Wikipedia pages.

some form of context, that is from a context domain $\mathbb{T}$. For generality, we can consider the set of all possible strings as the set of contexts, however, any other set of contexts can be considered.

Definition 1. A *relationship* between two entities e_1,$e_2 \in \mathcal{E}$ is a tuple $\langle e_1, e_2, \tau, t \rangle$, where $t \in \mathcal{T}$ is the validity time of the relationship and $\tau \in \mathbb{T}$ its context.

Previous work has built upon many different theories and definitions, on what constitutes an event. We consider an event to be the concentration of individuals at a specific period in order to engage to some activity. Thus, an event can be defined as a set of entities associated to some time period, together with some context.

Definition 2. An *event* is a tuple $\langle E, \tau, t \rangle$, where $t \in \mathcal{T}$, $\tau \in \mathbb{T}$, E is finite and $E \subset \mathcal{E}$.

In this notation, our DNA example event depicted in Figure 5.1 is denoted as $\langle$\{DNA, James Watson, Francis Crick\}, $\{s_1, s_2, s_3\}$, 1953$\rangle$, where the first set represents the entities involved in the event, the context is given by the set of sentences, and 1953 is the time the event takes place.

Wikipedia is a collection of documents providing information about real-world entities. A document is a sequence of terms that may be referring to real-world entities, to temporal periods, or have other meanings. Assuming the existence of an infinite set $\mathcal{W}$ of words, we introduce the set of *tokens* to be the set $\mathcal{O} = \mathcal{E} \cup \mathcal{T} \cup \mathcal{W}$ and define a *document* to be a finite sequence of tokens, divided into a set of *sentences*. We denote by $\mathcal{D}$ the set of all possible documents.

Wikipedia is a collection of web pages, with each page being about a specific real-world entity and providing information about that entity in the form of text.

Definition 3. A *page* is a pair $\langle e, D \rangle$, where $e \in \mathcal{E}$ is the entity of the page and $D \in \mathcal{D}$ its document. The set $\mathcal{P}$ denotes the set of all possible pages. A *set of Wikipedia pages* is a finite set $P \subset \mathcal{P}$ such that $\forall \langle e_1, D_1 \rangle, \langle e_2, D_2 \rangle \in P$: if $e_1 = e_2$, then $D_1 = D_2$.

Problem Statement We will use Wikipedia in order to identify and extract events that have happened in the real world. Thus, our problem definition on an abstract level becomes: Given a set P of Wikipedia pages, return a set R of events described in these pages.

5.5 Relationship Identification

To extract events from Wikipedia, a first step is to identify sets of entities constituting an event. Our approach is first to use co-reference to identify important binary relationships between entities and then look for sets of entities for which the identified co-reference binary relationships are collectively both strong and meaningful.

Consider the document of the Wikipedia page of an entity e containing a reference to an entity e'. The existence of this reference is a strong indication that e has some form of relationship to e'. The best source describing this relationship is most likely the sentence itself in which the reference appears.

If the sentence contains some time reference, we assume that it is the time associated to the relationship with the entity e'. If the sentence contains more than one time reference, then the one closest to e' is most likely to be the one with this role.

Based on these assumptions, for every appearance of an entity e' in a sentence $s_{e'}$ of a document of the Wikipedia page of the entity e, we can create a relationship $l = \langle e, e', s_{e'}, t_{s_{e'},e'} \rangle$, where the time $t_{s_{e'},e'}$ is the time reference in $s_{e'}$ closest to e'.

Since events have a highly temporal nature, lack of time references in a sentence means that an association is described for which time is not a significant factor. We will ignore these descriptions, and consider only those that have temporal references.

Recall that our fundamental principle for event identification is that events are created by entities connected in a specific time period. This means that if indeed two entities get related due to an event, both should be well aware of it, i.e., the page of the first should have a reference to the second, and the page of the second should contain a reference to the first. We refer to this dual reference situation as *co-reference*. It can happen that the two different references in our pages may be circumstantial and not related. To ensure that they are referring to the same event, we require them to relatively coincide, i.e., the two relationships they describe have some common time period. We refer to this as *time agreement*.

For the time agreement between two time values we adopt an exponential function that is based on Ebbinghaus' forgetting curve that models the nature of forgetting over time [Ebb13]. Given two times t_1 and t_2, their time agreement is computed as:

$$ta(t_1, t_2) = \begin{cases} e^{-\frac{distinct\_days(t_1,t_2)}{M}} & \text{if } t_1 \text{ and } t_2 \text{ overlap} \\ 0 & \text{otherwise.} \end{cases}$$

where $distinct\_days(t_1, t_2)$ is the absolute number of days which are not in common between the two times and M denotes the exponential decay. For our approach, we consider the number of days to be a reasonable choice, but other time metrics may be used as desired. The function returns 1 for a 0 distinct days time difference, converges to 0 for very different time intervals and equals 0 for non-overlapping time intervals. We can now formally define the co-reference relationship.

Definition 4. Given a threshold θ and two relationships $l_e = \langle e, e', s_{e'}, [t'_s, t'_e] \rangle$ and $l_{e'} = \langle e', e, s_e, [t_s, t_e] \rangle$, for which $ta([t'_s, t'_e], [t_s, t_e]) > \theta$, a *co-reference relationship* is a tuple $\langle e, e', s, t \rangle$ for which $s = \{s_{e'}, s_e\}$ and $t = [max(t_s, t'_s), min(t_e, t'_e)]$.

Intuitively, the co-reference relationship is a relationship that has as time the overlapping part of the times of the two relationships and as context the union of the contexts of the two relationships. The specific threshold θ as well as the value of the decay factor M are application and data dependent. Section 5.8.1 provides more information on the values we have selected.

Once the co-reference relationships have been created, it is possible that two entities have more than one co-reference relationship between them with the same time (because an entity is mentioned more than once in the same page). If this happens, then all these co-reference relationships are collapsed into one that has as context the union of their individual contexts and as time the common time value that all share.

Let $\hat{L}$ denote the set of co-reference relationships extracted from the Wikipedia set of pages according to the aforementioned procedure. Since each reference relationship is a binary relationship between entities, the set $\hat{L}$ can be modeled as a graph where the nodes represent entities and the edges the co-reference relationships between them. Each edge is annotated with time and context (i.e., the sentences of the Wikipedia page) of the respective co-reference relationship. We refer to this graph as *co-reference graph*.

Definition 5. A *co-reference graph* is a graph $G{:}\langle E, C, f_\tau, f_t \rangle$ where E is a set of nodes, $C \subseteq E \times E$ is a set of edges, while $f_t | C \rightarrow \mathcal{T}$ and $f_\tau | C \rightarrow \mathbb{T}$ are a time-assigning and context-assigning function, respectively, on the edges.

The creation of the *co-reference graph* is described in Algorithm 7. The first step is the generation of relationships between entities. For each page $p \in P$ it generates relationships between the entity of the page e and the entities contained into the sentences of the page. The set of all relationship is called L.

From these relationships, *co-reference* relations are extracted with the function GETCO-REFERENCERELATIONS. First, relationships L are grouped by their entities into the set L_g, and consequently L_g is divided in two distinct sets L_{g_e} and $L_{g_{e'}}$ based on page from where the relationship has been created.

Then, each element of the two sets is compared each other and if the time agreement between the two time period of the relationships is greater than the threshold θ a *co-reference* relationship is created and added to the set of co-reference relationships $\hat{L}$. Finally, the *co-reference* graph is created from the set of co-reference relationships $\hat{L}$.

Algorithm 7 Co-reference Graph Creation

procedure GENERATE CO-REFERENCE GRAPH(P)
  $L \leftarrow \{\}$
  for each $p \in P$ **do**
    $L \leftarrow L \cup$ GETRELATIONS(p)
  end for
  $\hat{L} \leftarrow$ GETCO-REFERENCERELATIONS(L)
  $G \leftarrow$ CREATECO-REFERENCEGRAPH($\hat{L}$)
  return G
end procedure

procedure GETCO-REFERENCERELATIONS(L)
  $\hat{L} \leftarrow \{\}$
  $L_g \leftarrow groupbyRelationByEntities(L)$
  for $l_e : \langle e, e', s_{e'}, [t'_s, t'_e] \rangle \in L_{g_e}$ **do**
    for $l_{e'} : \langle e', e, s_e, [t_s, t_e] \rangle \in L_{g_{e'}}$ **do**
      if $ta([t'_s, t'_e], [t_s, t_e]) > \theta$ **then**
        $s = \{s_{e'}, s_e\}$
        $t = [max(t_s, t'_s), min(t_e, t'_e)]$
        cl $= \langle e, e', s, t \rangle$
        $\hat{L} \leftarrow \hat{L} \cup cl$
      end if
    end for
  end for
  return $\hat{L}$
end procedure

5.6 Event Extraction

Once the co-reference binary relationships have been created, the next step is to identify groups of entities that indicate a strong and meaningful connection between them, highly likely representing an event. The co-reference graph offers an excellent structure for performing this task since it models not only the entities, but also the way that they relate to each other through the co-reference relationships, alongside context and validity time of these relationships.

There are many possibilities to define what constitutes a strong and meaningful connection between entities. As our initial assumption is that an event is a set of entities associated within a specific time period, all our strategies are based on the same initial step: the division of the co-reference graph into a set of components of temporally connected entities. We identify this kind of component with the term *time agreement subgraph*. With this step, we aim to identify a maximal set of connected entities in such a way that the time difference between any two co-reference relationships that exist between these entities is within a specific threshold. Given the resulting set of subgraphs, we can add further constraints, e.g. taking the context into account. We do not need to apply these constraints on the complete co-reference graph. Instead, they are applied on the connected components, resulting in more efficient extraction. This process is described in Algorithm 8.

We will describe and evaluate three different strategies. Each of them returns as answers subgraphs of the input co-reference graph. Turning an answer subgraph into an event is straightforward: (i) the event entities are composed by the nodes and those entities that the subgraph's sentences have in common, (ii) the event time is the intersection of all times assigned to the co-reference relationships (edges), and (iii) the event context is composed of the union of all involved contexts.

Algorithm 8 Event extraction process

1: **procedure** EVENT EXTRACTION(P)
2: $G \leftarrow$ GENERATECO-REFERENCEGRAPH(P)
3: $G_c \leftarrow$ GRAPHCOMPRESSION(G)
4: $\mathcal{G}_{maximal} \leftarrow$ EXTRACTTIMEAGREEMENTSUBGRAPHS(G_c)
5: $R \leftarrow$ EXTRACTEVENTS($\mathcal{G}_{maximal}$)
6: **end procedure**

5.6.1 Time Agreement subgraphs

The co-reference graph may contain multiple co-reference relationships between the same entity in the same time period. In order to extract, time agreement subgraphs we first compress edges, between two entities, related to the same periods, into a single edge by joining the all contexts together.

The second step of our approach is the extraction of time agreement subgraphs. Given the co-reference graph, we are looking at discovering maximally connected components in the co-reference graph such that the time agreement between the time of any two edges (co-reference relationships) in the connected component is greater than a specific threshold θ.

To materialize such a result, we start from an edge in the co-reference graph and incrementally expand it with new edges in a breadth-first fashion. This procedure is shown is Algorithm 9.

In order to extract time agreement subgraphs we iterate over the set of edges of *co-reference* graph. For each edge $c \in C$, the function ExtractTTC extracts the time agreement subgraph that contains the edge c. This function is implementing a breadth-first visit in the graph starting from the input edge c that is the first node inserted in the queue q.

After this initial setup, until the queue is empty a node c is extracted from the queue q, and the neighbors of the edge c has been extracted. For every neighbor edge that is analyzed, the time agreement between its time and the time of each edge already included is calculated (line 23). The function checkTimeAgreement checks if all the time periods previously encountered have a time agreement greater than θ.

If there is at least one case in which that time agreement is less than the threshold θ, then the visited edge is not added to queue. If not, the visited edge becomes part of the connected component and it will be added to queue q and also the time period of the edge is added to $timeIntervals$ set (line 23 - 27) and the breadth-first traversal continues. Once no more edges can be added, the connected component G' is emitted.

This procedure is repeated for every edge in the co-reference graph and each resulting subgraph G' is added to the set of subgraphs $\mathcal{G}$.

Algorithm 9 Extraction of Time Agreement SubGraphs

```
1: procedure ExtractTimeAgrementSubgraphs(G_c)
2:     G ← {}
3:     for c ∈ G_C do
4:         G' = ExtractTTC(c)
5:         G ← G ∪ G'
6:     end for
7:     G_maximal ← MaximalSubgraphs(G)
8:     return G_maximal
9: end procedure
10:
11: procedure ExtractTTC(c)
12:     visited ← {}, timeIntervals ← {}
13:     q ← {c}
14:     G ← ∅
15:     timeIntervals ← timeIntervals ∪ f_t(c)
16:     while q ≠ ∅ do
17:         c ← dequeue(q)
18:         visited ← visited ∪ c
19:         G' ← G ∪ c
20:         neighbors ← getNeighborsEdges(G, c)
21:         for n ∈ neighbors do
22:             if n ∉ visited then
23:                 if checkTimeAgreement(n, intervals) then
24:                     q ← q ∪ n
25:                     interval ← f_t(n)
26:                     intervals ← intervals ∪ interval
27:                 end if
28:             end if
29:         end for
30:     end while
31:     return G'
32: end procedure
```

It may happen that two processes starting from different edges produce the same connected component. We therefore perform duplicate elimination after all components have been constructed, this process is also in charge of selecting between duplicate component the maximal ones. The set of maximal subgraphs are contained into the set $\mathcal{G}_{maximal}$.

5.6.2 Temporally Related Events (TR)

The time agreement subgraph represent the basis of discovery events. These subgraph encodes relationships of a set of entities in similar time period. The first approach aims

at extracting events from these temporal components. Not all of these components might represent events, but in all cases provide a good basis for further methods with stricter constraints. Each subgraph $G' \in \mathcal{G}$ is transformed into an event. This procedure is described in Algorithm 10.

Algorithm 10 Extraction of TR events

```
1: procedure ExtractTR(G_maximal)
2:     R ← {}
3:     for G' ∈ G_maximal do
4:         R = R ∪ GenerateEvent(G')
5:     end for
6:     R' ← RemoveDuplicateEvents(R)
7:     return R'
8: end procedure
```

Entity	Sentence
e_1: Nintendo	s_1: Intending to broaden the **3DS** market, **Nintendo** released 2013's cost- reduced **Nintendo 2DS**.
e_2: Nintendo 2DS	s_2: On 31 October 2013, **Nintendo** president **Satoru Iwata** admitted that the **Nintendo 2DS** lacked awareness among prospective purchasers.
e_3: Nintendo 3DS	s_3: On October 31, 2013, **Nintendo** abruptly suspended the **Swapnote**/Nintendo Letter Box SpotPass functionality ...

Figure 5.2: Sentences related to Nintendo and its products.

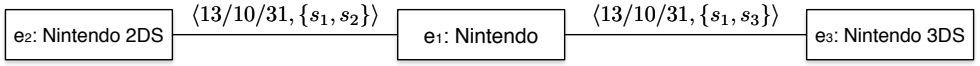


Figure 5.3: Co-reference graph and **TR** subgraph for the example in Figure 5.2.

Example 5.1. *Consider three Wikipedia pages with sentences that mention each other as indicated in Figure 5.2. The first (e_1) references the second and the third as seen by sentence s_1. The second (e_2) references the first but not the third, and the third (e_3) references the first but not the second. As a result, the co-reference graph is the one illustrated in Figure 5.3, where the times on the edges are computed as the intersection of the times in the respective sentences. Since these times are the same for both edges, the procedure will build the same connected subcomponent (the whole graph itself) for both edges. After duplicate elimination, one copy remains as result that is afterwards transformed into the set of **TR** events.*

5.6.3 Fully Temporally Related Events (FTR)

The previous criterion for event generation was based on time agreement but allowed the entities to be connected to each other in any possible way. However, if an event is considered as the concentration of entities at a specific point in time for a specific reason, it is only reasonable to assume that each entity participating in the event will be connected to each of the others. Thus, we can enforce an additional constraint which asks for all entities participating in an event to be fully connected. To implement this, we run the Bron and Kerbosch algorithm [BK73] on the time agreement subgraphs $\mathcal{G}_{maximal}$, in order to identify and extract from each one of them the maximally fully connected component (clique) that they contain G_{clique}, or all of them in case they contain more. Each clique serves as an answer, i.e., is considered to represent an event. This process is described in Algorithm 11.

Algorithm 11 Extraction of FTR events

```
1:  procedure ExtractFTR(G_maximal)
2:      R ← {}
3:      for G' ∈ G_maximal do
4:          G_clique ← ExtractClique(G')
5:          for G_c ∈ G_clique do
6:              R ← R ∪ GenerateEvent(G_c)
7:          end for
8:      end for
9:      R' ← RemoveDuplicateEvents(R)
10:     return R'
11: end procedure
```

Example 5.2. *Clique discovery applied on the result of the* ***TR*** *approach in Figure 5.3 yields two subgraphs as shown in Figure 5.4.*

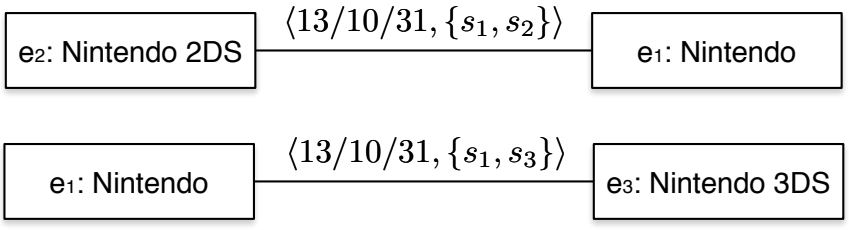


Figure 5.4: Subgraphs extracted using **FTR**.

5.6.4 Content-Temporally Related Events (CTR)

Two sentences with the same entities and time expressions could describe different events if there are multiple actions between them, as in Figure 5.2. To distinguish between such events, we investigate a third method that looks for content-temporally related events (**CTR**). A key insight here is that semantics does play an important role in the case of concurrent events involving the same entities: the proposed extraction of **TR** and **FTR** events is solely based on connectivity and temporal constraints, but the sentences (i.e., context) from which the relationships originate are not considered. To incorporate them, we demand that entities involved in an (**CTR**) event not only share a similar time period (i.e. time agreement), but they need to be extracted from semantically similar sentences. This is done by first clustering similar sentences and then extracting events based on these clusters.

Algorithm 12 Extraction of CTR events

```
1:  procedure ExtractCTR(G_maximal)
2:      R ← {}
3:      for G' ∈ G_maximal do
4:          G_D ← ∅
5:          for ⟨i, j⟩ ∈ G' do
6:              S_{i,j} ← GetSentences(i, j)
7:              S^N_{i,j} ← ClusterSentences(S_{i,j})
8:              for S^k_{i,j} ∈ S^N_{i,j} do
9:                  c_d = ⟨e_i, e_j, S^k_{i,j}, t⟩
10:                 G_D ← G_D ∪ c_d
11:             end for
12:         end for
13:         G^K_D ← ExtractCTRSubgraphs(G_D)
14:         for G^k_D ∈ G^K_D do
15:             R = R ∪ GenerateEvent(G^k_D)
16:         end for
17:     end for
18:     return R
19: end procedure
```

Sentence Clustering

An edge is describing the relationship between two entities and the sentences is giving the describing this relationships. This whole procedure is described in Algorithm 12. Given a time agreement subgraph G', the set of sentences $S_{i,j}$ between each pair of

connected entities e_i and e_j needs to be clustered. The goal of this step is clustering set of contexts based on their similarity based on the event that they describe. In order to compare sentences, we used the set of tokens (i.e., entities, times and words) of the sentence itself. Therefore, we employ a distance measure based on two token sets:

$$dist(O_1, O_2) = 1 - (\alpha \cdot simE(E_1, E_2) + (1 - \alpha) \cdot simW(W_1, W_2)),$$

where $simE(E_1, E_2)$ is an entity-based similarity function, $simW(W_1, W_2)$ is a word-based similarity function, and α is used to balance between their values. Similarity between two sets of words W_1 and W_2 is computed using the pre-trained Google News word embeddings [1]: for both word sets W_1 and W_2, the sums of their embeddings are compared using cosine similarity [ML08]. The similarity function between the two entity sets E_1 and E_2 is defined as their overlap coefficient:

$$simE(E_1, E_2) = \frac{|E_1 \cap E_2|}{min(|E_1|, |E_2|)}.$$

At the beginning of the clustering phase, each sentence $s \in S_{i,j}$ resembles a cluster.

Iteratively, two clusters are joined if the distance of their union of tokens is lower than the threshold γ. This clustering step results in a set of clustered sentences $S^1_{i,j} \dots S^n_{i,j}$.

We exploit these sentence clusters to create a decompressed multigraph $G_D = (E, C_D)$ with the same set of nodes as the co-reference subgraph G'. For each cluster $S^k_{i,j}$, there is an edge in the form $\langle e_i, e_j, S^k_{i,j}, t \rangle$ (t is computed as the intersection of the closest time mentions of each sentence in $S^k_{i,j}$) (lines 4 - 12).

Example 5.3. *Starting from the subgraph in Figure 5.3, the clustering phase results in four distinct edges containing a single sentence each, due to the dissimilarity of the sentences: $dist(O_1, O_2) > \gamma$ and $dist(O_1, O_3) > \gamma$. The resulting decompressed multigraph is shown in Figure 5.5.*

CTR Subgraph Extraction

Given the decompressed multigraph G_D, we extract **CTR** event subgraphs under the constraint that no subgraph may include more than one edge between a pair of entities.

[1] https://code.google.com/archive/p/word2vec/

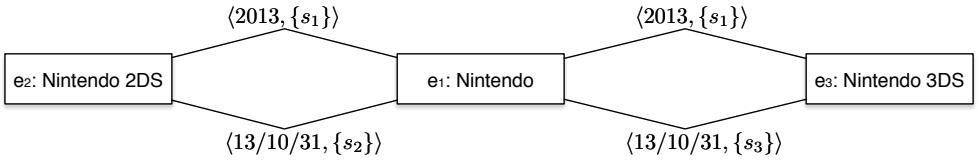


Figure 5.5: Decompressed multigraph G_D created from the co-reference graph in Figure 5.3.

We collapse semantically similar edges in a greedy manner. Starting from any pair of entities, a new subgraph is created for each edge between them. Distances between these edges and all adjacent edges are computed based on the union of their tokens. The subgraph connected to the least-distant edge is extended by this edge if the distance is lower than γ. This process continues until all edges in the graph G_D have been visited and assigned to a subgraph. During that process, if an edge cannot be connected to any subgraph (distance greater than γ), it constitutes a new subgraph. At the end, the subgraphs ${G_D}^1 \dots {G_D}^k$ constitute the basis for **CTR** events. Each cluster of sentences on the edges represents the context.

Example 5.4. *The decompressed multigraph G_D in Figure 5.5 is divided into three different subgraphs: starting from the entity pair e_1 and e_2, two subgraphs are created, one containing the edge $\langle 2013, \{s_1\}\rangle$, the other one $\langle 13/10/31, \{s_2\}\rangle$. As $dist(O_{s_1}, O_{s_1}) < \gamma$, the two edges $\langle 2013, \{s_1\}\rangle$ that connect e_1 with e_2 and e_3 are merged into one subgraph. No other connections are possible. As $dist(O_{s_2}, O_{s_3}) \geqslant \gamma$, the two edges $\langle 13/10/31, \{s_2\}\rangle$ and $\langle 13/10/31, \{s_3\}\rangle$ compose their own subgraphs. The resulting three subgraphs are shown in Figure 5.6.*

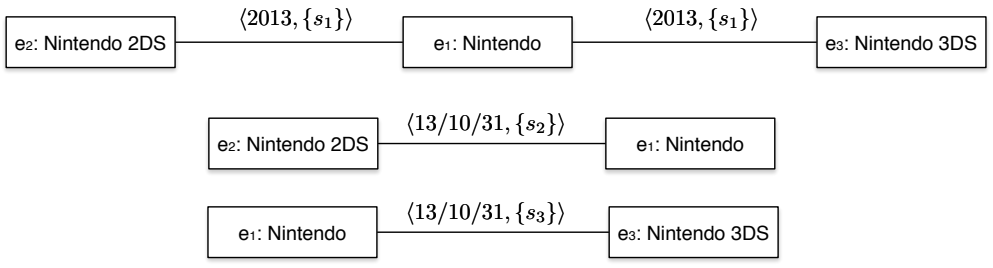


Figure 5.6: Subgraphs extracted using **CTR**.

5.7 Event Clustering and Expansion to Asymmetric Events

The generated events can be semantically described by their entities, time span and textual context. To extend these semantic descriptions, we now present an approach to create clusters of conceptually similar events, based on key terms and entity types. The use of key terms is motivated by the concepts of Folksonomy [SM07] where similar items are grouped based on tags like "sports" or "music". Using entity types such as "Person" and "Company" is a common way to describe relations [PB13, ABK+07a].

In addition to the semantic description, more events can be found using the generated clusters. Until now, the described methods enable an extraction of high-quality events that are mentioned not only in the context of one entity, but in the context of multiple entities. However, this means that events of asymmetric importance among their participants may be overseen. By exploiting the structure of the clusters, such events of asymmetric importance can be detected.

5.7.1 Event Clustering

The goal of the event clustering is to generate a set of clusters and to assign our events to them. A cluster is defined as follows:

Definition 6. A *cluster* is a tuple $c = \langle r, c' \rangle$, where $r \subseteq R$ is a set of events and c' is its centroid.

Algorithm 13 gives an overview of the event clustering approach. Given as an input the set G of all co-reference relations, the set R of generated events, a sample of events $R' \subset R$ and some parameters explained later, the following steps are executed in order to create the clusters: First, key terms and an entity type hierarchy are extracted that are later used in the similarity functions (line 2 - 3). Based on the key terms and entity types of the events in R', blocks are created for efficiency reasons: event distances are computed only for events within the same blocks (line 4 - 5). Finally, the clusters are created in two steps explained later (line 6 - 7).

Algorithm 13 Event Clustering

1: **procedure** EVENT CLUSTERING($G, R, minSupport, minImpact, R', keyTermTh, \epsilon$)
2: $K \leftarrow$ KEYWORDEXTRACTION($G, R, minSupport, minImpact$)
3: $T \leftarrow$ EXTRACTTYPES
4: B = CREATEBLOCKS($R', keyTermTh$)
5: INITDISTANCES(B)
6: C = CLUSTEREVENTS(R', ϵ)
7: C = MERGECLUSTERS(C, ϵ)
8: **return** C
9: **end procedure**

5.7.1.1 Key Term Extraction

In order to limit the number of tokens within the context of each event, a set of key terms is created first. These key terms are specifically important terms in the contexts of events. We employ and approach similar to [KL08], where important terms in a specific corpus are distinguished from the terms in a generic corpus. Precisely, key terms are extracted from the events by comparing the frequencies of the words in all co-reference relations G to the frequencies in the sentences contained in the relationship set R' as defined in Section 5.5.

The context of each event is split into tokens. For each token k, the support is computed as the relative frequency of that token in the given co-reference corpus (here G):

$$Support_G(k) = \frac{|\{e \in G | k \in \tau(e)\}|}{|G|} \tag{5.1}$$

We also compute the token's impact as the ratio of occurrences in R' compared to G:

$$Impact_{R',G}(k) = \frac{Support_{R'}(k)}{Support_G(k)} \tag{5.2}$$

Now, the set of key terms is the set if tokens whose support and impact exceeds the input threshold parameters $minSupport$ and $minImpact$:

$$K = \{k \in \mathbb{T}, Support_{R'}(k) \geq minSupport \wedge Impact_{R',G}(k) \geq minImpact\}$$

5.7.1.2 Type Extraction

The second feature to compare events in the cluster are entity types: From an external resource, a type hierarchy is extracted first. With this, each entity can be assigned a set of entity types with sub and super types: For example, Nintendo is both a toy company and video game publisher, which are both (transitive) sub types of "company". By resolving the type hierarchy, each entity is assigned a type frequency vector of length y, where Y is the total number of types and where each entry denotes the number of occurrences of the respective type in the entity's type and super types.

5.7.1.3 Similarity Metrics

The distance between two events v_1 and v_2 during the clustering phase is based on their key terms and entity type distance. These two sub scores are weighted using $0 \leq \alpha \leq 1$.

$$distance(v_1, v_2) \;=\; \alpha \cdot keyTermDistance(v_1, v_2) + (1 - \alpha) \cdot typeDistance(v_1, v_2)$$

To compute key term based distance scores, we first introduce a term similarity metric: $termSimilarity(k_1, k_2) \mapsto [0, 1]$ that gives a higher scores to semantically similar terms based on word embedding similarity. The key term based distance between two events v_1 and v_2 is then computed by averaging over pairs (k_1, k_2) of least distant key terms in both events:

$$keyTermDistance(v_1, v_2) = \frac{1}{2} \cdot \frac{\sum_{k_1 \in v_1.K} \min_{k_2 \in v_2.K} \{1 - \text{termSimilarity}(k_1, k_2)\}}{|K_1|} + \frac{1}{2} \cdot \frac{\sum_{k_2 \in v_2.K} \min_{k_1 \in v_1.K} \{1 - \text{termSimilarity}(k_1, k_2)\}}{|K_2|}$$

The type-based distance scores are computed in a similar manner. Instead of the term similarity function, the cosine similarity between the entities' type frequency vectors is used:

$$typeDistance(v_1, v_2) = \frac{1}{2} \cdot \frac{\sum_{e_1 \in v_1.E} \min_{e_2 \in v_2.E}\{1 - \cos(e_1, e_2)\}}{|E_1|} + \frac{1}{2} \cdot \frac{\sum_{e_1 \in v_2.E} \min_{e_2 \in v_1.E}\{1 - \cos(e_1, e_2)\}}{|E_1|}$$

5.7.1.4 Blocking

To avoid computing the distance for each possible pair of events in R', blocks of presumably similar events are created first. Each block is characterized by a set of key terms and one entity type. To do so, similar key terms are grouped, such that there is a set of key term groups:

$$K' = \{K_1, K_2, \ldots, K_\kappa\}, \text{ where } \forall K_i \in K' : \forall k_p \in K_i : \exists k_q \in K_i, k_q \neq k_p, \text{ such that } \text{termSimilarity}(k_p, k_q) \geq keyTermSimilarityThreshold$$

Each entity type is combined with a key term group, such that there are $y \cdot |K'|$ blocks. Events are assigned with respect to their key terms and entity types (one event can be assigned to multiple blocks). Then, the distance scores between each pair of events within these blocks are computed.

5.7.1.5 Clustering

Using the similarity metrics, we create clusters of events using a similar approach as in [SAMA17] where events are clustered based on their nearest neighbors. The exact procedure is shown in Algorithm 14: For each event, events within a distance of ϵ are collected (line 4) and a new cluster c is created consisting of these neighbors and the event itself. Now, for each neighbor n that is already assigned to a cluster $n.cluster$, the cluster cohesiveness scores of c and n are computed as defined in [SAMA17] (line 7). If the new cluster's score is higher, this one is added to the set of clusters.

Algorithm 14 Event Clustering

```
1: procedure ClusterEvents(R', ε)
2:     C ← {}
3:     for e ∈ R' do
4:         N ← Neighbors(e, ε)
5:         c ← new Cluster(N ∪ e)
6:         for n ∈ N do
7:             if n.cluster ≠ ∅ ∧ clusterCohesiveness(c) > clusterCohesiveness(n.cluster) then
8:                 n.cluster.remove(n)
9:                 if n.cluster.isEmpty() then
10:                    C.remove(n.cluster)
11:                end if
12:            end if
13:        end for
14:        if ¬c.isEmpty() then
15:            C = C ∪ c
16:        end if
17:    end for
18:    return C
19: end procedure
```

The cluster cohesiveness is a strict metric that strives for highly coherent, but rather small clusters. Therefore, an additional post-processing step is added where clusters are merged based on the similarity of their centroids. As shown in Algorithm 15, a priority queue PQ of cluster pairs is created in order of increasing cluster distance, where a cluster distance below the input threshold ϵ is required (line 2 - 8). In that order, in each step the two most similar clusters c_1 and c_2 are merged into a common cluster c. In the following, the map is reorganized by removing all reference to the cluster pair and by inserting new cluster pairs containing c (line 9 - 19).

5.7.2 Expansion to Asymmetric Events

For efficiency reasons, the cluster generation is based on a sample set R'. Consequently, the event set $R'' = R \setminus R'$ has not been assigned to clusters yet. This needs to be done in an additional step with R'' and the set of clusters as inputs C, where events in R'' are assigned to clusters in C.

In a similar process, the clusters in C can be used to detect even more events: Until now, our event extraction approach relied on the assumption that significant event are told from the perspective of several involved subjects, so they can be detected by looking

Algorithm 15 Cluster Merging

```
1: procedure MergeClusters(C, ϵ)
2:     for c1 ∈ C do
3:         for c2 ∈ C do
4:             if c1 ≠ c2 ∧ distance(c1, c2) ≤ ϵ then
5:                 PQ.add((c1, c2))
6:             end if
7:         end for
8:     end for
9:     while ≠ PQ.isEmpty() do
10:        c = new Cluster(PQ.peek())
11:        C = C ∪ c ∖ {c1, c2}
12:        removeFromQueue(PQ, c1)
13:        removeFromQueue(PQ, c2)
14:        for c1 ∈ C do
15:            if c1 ≠ c ∧ distance(c, c1) ≤ ϵ then
16:                PQ.add((c, c1))
17:            end if
18:        end for
19:    end while
20:    return C
21: end procedure
```

for co-references. As demonstrated later in the evaluation, this approach enables to generate a set of events with high precision: However, it misses out events of asymmetric importance. For example, consider the event about Nintendo's foundation in Tokyo on 23 September 1889 may be important for Nintendo itself, but not for the city Tokyo. That means, there is a reference to the date and Tokyo in the Nintendo article, but the date is not mentioned in the Tokyo article. Moreover, that foundation event should even be detected if there was no mention of Tokyo in the Nintendo article: The term "founded" and the date reference should already make up an event.

To detect events of asymmetric coverage in Wikipedia or event with one entity mention only, we utilize the clusters C generated from the events extracted by exploiting the temporal entity co-reference. Our goal is to identify each sentence annotated with at least one entity and a time reference as an event if it can be assigned to one of the previously generated clusters.

As no new clusters are being created and no changes are made in their description and due to the high number of potential cluster similarity computations, we propose a straight-forward approach to assign asymmetric event candidates and those outside of the sample (R'') to clusters: For each event candidate, (i) select the clusters that

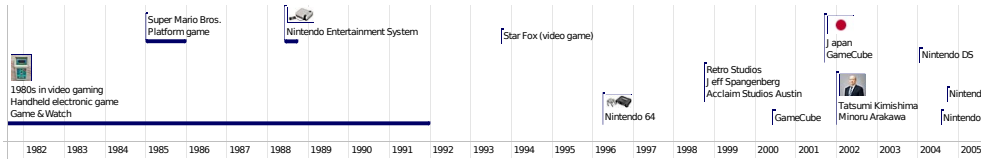


Figure 5.7: Part of the timeline for Nintendo. Above this frame, more events are displayed for the selected time span.

share the most key terms with the event candidate, (ii) From the clusters, choose that cluster that shares most entity types with the event candidate. At least one common key word and two common entity types are required. Otherwise, the event candidate is not assigned to the cluster. While we treat the events in R'' as events even when they can not be assigned to clusters, only those candidate events assigned to clusters are kept as events.

5.8 Experiments and Examples

5.8.1 Dataset annotation

As input corpus we used the article pages contained in the complete English Wikipedia dump of March 3, 2016. From each article we removed the infoboxes, tables, categories and citations. Page texts were split into sentences using the Stanford Parser [KM03], their time expressions were extracted using SUTime [CM12] (which also detects and normalizes implicit time expressions such as "one year earlier"), and links were retrieved from the Wikipedia markup language. Since entities in Wikipedia are linked just at their first occurrence in a page, we used DBpedia Spotlight [DJHM13] with a confidence threshold of 0.6 as a named entity disambiguation tool to extend the pages with further links. DBpedia Spotlight was shown to perform well and efficiently on Wikipedia texts [NHTW14].

5.8.2 Extraction Process

During the preprocessing step we found that $5M$ pages contain at least one entity mention. These pages were split into $121M$ sentences. Within these sentences we found $105M$ instances of entity mentions and $61M$ time reference instances. Experimentally, we set the parameters for the temporal similarity function ta to $M = 2000$ and $\theta = 0.6$.

With these parameter values, a time period composed by a single day can still be matched to another time period referring to an entire year. From the entire English Wikipedia we extracted a set of $684K$ co-reference relationships $\hat{R}$. This set was used for the creation of the co-reference graph that is composed of $552K$ nodes and $684K$ edges. For **CTR** we experimentally set $\alpha = 0.5$ and $\gamma = 0.7$. We implemented our approach in Java 1.8, as a distributed map-reduce application running on top of Apache Spark 2.1 [Spa16]. Our experiments were performed on a cluster of six virtual machines, each one with eight cores and 96GB of main memory. The total time needed for the event extraction (**TR**, **FTR** and **CTR**) was 5 hours, while the entity and time annotation took 4 days on a single machine. The **TR**, **FTR** and **CTR** events are available online, as well as the code and further data from the evaluation described in the following section.

5.8.3 Illustrative Results

Before getting into details, let us discuss some example events extracted with our approach under the constraints of temporal co-references to illustrate its flexibility. Our first example describes the foundation of the Canadian football team Montreal Alouettes by its three founders in 1946:

⟨{Eric Cradock, Lew Hayman, Léo Dandurand, Montreal Alouettes}, {"The **Alouettes** were first formed in 1946 by CFL hall of famer **Lew Hayman** along with businessmen **Eric Cradock** and **Léo Dandurand**.", "Along with **Lew Hayman** and **Léo Dandurand**, **Cradock** co-founded the original **Montreal Alouettes** in 1946."}, *1946*⟩.
The foundation plays an important role for all involved entities and clearly marks a specific event that can be mapped to a specific point in time. These kinds of events may potentially be found also in knowledge bases with a limited set of relations, as both the foundation of the team and the relation between a founder and the team can be expressed using binary relations. However, due to the use of binary relations as in [WZQ+10] and [ZSW15], the foundation is often not represented as a single and coherent event.

Another event, which involves different types of entities is the following, describing the killing of two Italian journalists working for Rai 3 on their way from Bosaso to Mogadishu, Somalia in 1994:

⟨{Bosaso, Ilaria Alpi, Italy, Miran Hrovatin, Mogadishu, Rai 3, Somalia}, *1994-03-20*⟩.[2] The event involves two persons (Ilaria Alpi and Miran Hrovatin), an organization (Rai 3), locations of the actual event (Bosaso, Mogadishu, Somalia) and a location referring to the journalists' citizenships (Italy). Given this complex structure it is not possible to map it to a general and reusable schema representing such events as done in other approaches [ZSW15, KW12]. Despite this, it clearly represents an important singular event in the real world. The killing of both journalists could also be described for each single journalist, though without providing an opportunity to detect the connection between these two deaths.

A broader event that includes not just the original event, but its consequences as well is the following:

⟨{Boom in the production of wheat, Epidemic, Stem rust, 1687 Peru earthquake, Peru, Chilean wheat cycle}, *1687*⟩.
After an earthquake and a stem rust epidemic in 1687, there was a heavy demand for wheat in Peru which could be met by Chilean exports. This event describes a long-running process involving two countries which plays an important role in their history. Such dependencies from long-running processes are often only seen afterwards and therefore not detectable when searching for peaks in user activity like in [LLL+12] and [TA14].

As demonstrated by these example events, our approach is able to detect important events that do not fit into predefined schemas, and does not have any limitations regarding the type of entities or the timing of events.

5.8.4 Entity Exploration

The generation and visualization of timelines has been discussed in several contexts, for depicting events involving a specific person [ADM+15b], topics in a given corpus [SA00] or a set of topically related events [ZDFB12]. Of particular interest is the first case which allows to efficiently get an overview of the events in a person's life and to explore them to gather more information. However, the timelines presented by Althoff et al. [ADM+15b] are restricted to human beings, their events are strictly filtered to

[2] For brevity, we omit the context information for the remaining exemplary events in this chapter.

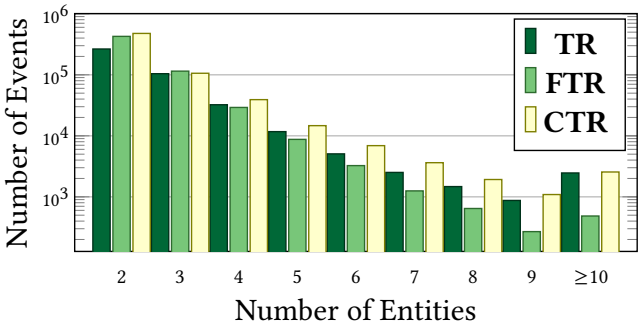


Figure 5.8: Distribution of event size.

follow some visual constraints, and they are based on facts from the deprecated Freebase knowledge graph. Consequently, these timelines are limited to very few selected events that are often taken from few available properties, such as starring in movies. With the set of events that we have generated by exploiting temporal entity co-references, we aim at providing a broad overview of an entity's life span, including not just human beings, but also other types of entities such as companies, military conflicts and others.

Figure 5.7 shows an example of our timeline application. It illustrates a selection of **FTR** events involving the entity "Nintendo". For each event, time and duration, involved entities, and images of them are shown. If the user clicks on one of the events, more detailed information is shown, including the Wikipedia sentences from where the selected event was extracted and the link to entities involved. The user can interact with the timeline to explore the entity in more details: By zooming in or out, the time span of interest can be specified, and by selecting one or more of the entities involved in the events, the set of events is filtered according to that selection. The Nintendo example provides a detailed overview of the company's history, e.g. the evolution of their systems over time. Nintendo started with arcade machines in the 1980s, later released their first video game console, and then developed handheld game consoles.

5.8.5 Statistics

Table 5.1 presents basic statistics for each of these methods: each of them leads to the generation of more than 425k events, with **TR** generating less events compared to the other two methods, but with a greater number of involved entities with respect to **FTR** and **CTR**. **CTR** leads to the highest number of events, as it distinguishes concurrent events between the same set of entities.

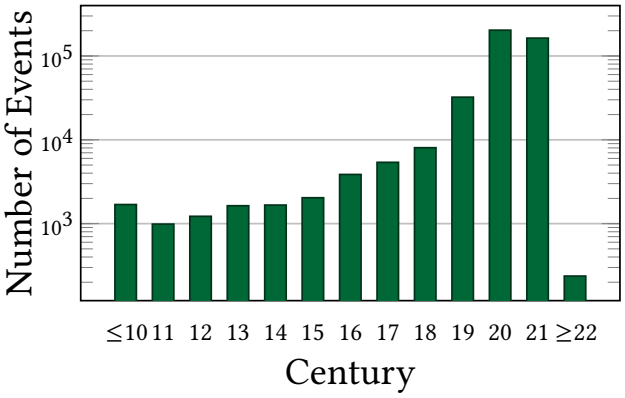


Figure 5.9: Temporal event distribution (TR).

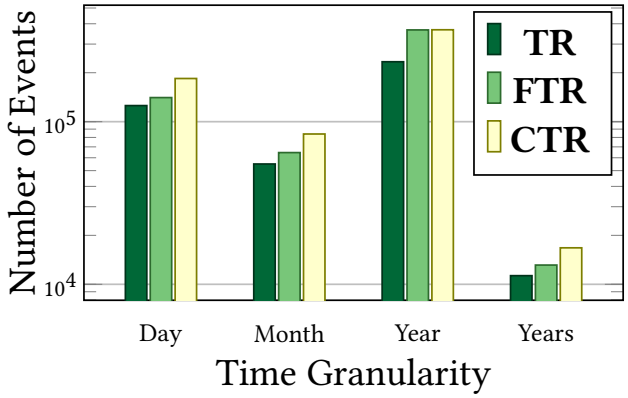


Figure 5.10: Event distribution by time granularity.

Method	#Events	Average size
TR	425,400	2.67
FTR	584,948	2.39
CTR	652,413	2.50

Table 5.1: Cardinality and average size of event sets.

These observations are also supported by Figure 5.8 which lists the number of events found for different sizes of entity sets. For all event sets the number of events decreases with the number of entities involved.

The **TR** method extracts less events than **FTR** and **CTR** which can be explained by the exploratory manner of the graph traversal, illustrated by the following example event:

⟨{Beijing West Railway Station, Baiziwan Station, Jiaohuachang Station, Jiulongshan Station, Nanlouzizhuang Station, Huagong Station, Happy Valley Scenic Area Station,

Dajiaoting Station}, *2014-12-28*⟩ .

This event represents the opening of a new line as part of the Beijing Subway. There is no page mentioning all the entities, instead the pages contain sentences like "It was opened on December 28, 2014 as a part of the stretch between Beijing West Railway Station and Jiaohuachang and is located between Huagong Station and Happy Valley Scenic Area Station" (from the "Nanlouzizhuang Station" page). The exploration of the **TR** subgraphs succeeds fusing this information together into a single event.

Time plays an essential role in our event detection process, as it is a mandatory component of each event and employed for the actual detection of entity co-references. In contrast to other approaches being limited to ongoing events or events covered by the time span of a news collection, our approach is not limited to any time period, including past events given the encyclopedic nature of Wikipedia. Figure 5.9 supports this claim by giving an overview of the temporal distribution of events detected with the **TR** method (**FTR** and **CTR** have a similar distribution): our data covers events from modern and post-classical history and even finds several events from ancient history, as well as (possible) futures, as shown by the following examples:

- ⟨{William II of England, Westmorland}, *1092*⟩: In 1092, William II took control of the area Westmorland previously claimed by the Scots.
- ⟨{100 Years (film), Robert Rodriguez}, *2115-11-18*⟩: The film "100 Years" directed by Robert Rodriguez is due to be released on 18 November 2115.

As shown in Figure 5.10, durations differ across the generated events: while single-day events tend to describe singular events like deaths or natural disasters, longer-lasting events represent election processes, military conflicts or similar.

5.8.6 Clustering Results

Extracting terms with a minimum support of 0.5% in $\hat{R}$ and a minimum support ratio of 1.25 gives us a set of 161 key terms. For example, the stemmed term "debut" appears in 2% of the relationships extracted from $\hat{R}$ (support) and more than 6 times as often in $\hat{R}$ than in the rest of Wikipedia (support ratio).

For efficiency, we apply this algorithm only to a sample of 50,000 events to create the set of clusters and then add the remaining ones by mapping them to the centroids. With that sample, we found 258 clusters and were able to cluster more than 80% of the events after the additional mapping.

#	Size	Key terms and entity types
1	25,149	Terms: released, album, band, studio, song Types: Person, Creator, Artist, Musician, Artifact
2	23,287	Terms: directed, film, starring, american, written Types: Movie, Person, Psychological Feat., (Social) Event
10	8,508	Terms: music, won, award, awarded, song Types: Person, Creator, Artist, Musician, Singer
11	7,971	Terms: party, elected, election, defeated, won Types: Person, Leader, Abstraction, (Social) Group
17	6,781	Terms: married, born, son, daughter, wife Types: Person, Leader, Communicator, Entertainer, Performer
21	5,237	Terms: company, acquired, purchased, million, announced Types: Abstraction, (Social) Group, Organization, Institution

Table 5.2: Selection of clusters found for the **CTR** events.

Table 5.2 gives an overview of selected clusters for the **CTR** events. We observe that some of the clusters such as clusters 10 and 17 directly resemble YAGO relations ("hasWonPrize", "isMarriedTo") and therefore show potential to extend knowledge bases based on existing properties. The two biggest clusters resemble the YAGO "created" and "wasCreatedOnDate" relations, but are more detailed, referring to the music or movie domain. Finally, we find relations not present in YAGO: cluster 11 is about political elections[3], while cluster 21 is about the acquisition of companies. This analysis of our events based on key terms and clustering not only confirms the presence of relations known from classic knowledge base schemes, but also demonstrates the ability of our approach to detect more types of events.

5.9 Evaluation

5.9.1 Coverage-Based Analysis

We have pointed out the flexibility of our event extraction approach with respect to the general types of real-world events covered, number of involved entities and temporal

[3]Note that due to our co-reference approach, opposing key terms such as "defeated" and "won" are often found in the same events.

flexibility. To gain more insight into the actual coverage of our event sets, we have compared them also to established human-curated datasets. Due to the variety of event models (e.g. named events, textual events and temporal relations) and the vast amount of real-world event described in web resources, such a comparison is difficult and often requires human judgments on small samples [SG16, AB15]. We will first present some automated comparisons to other resources and methods to understand their coverage of events and in addition assess the quality of our extracted events through human judgments.

First, we compare the **CTR** events and two baselines to (i) named events according to the subclass hierarchy of Wikidata [VK14], and (ii) textual events automatically extracted from the Wikipedia current events portal [TA14] and Wikipedia event lists, similar to [HL12]. From these resources, we retrieve 116,939 trusted named events and 178,933 trusted textual events. As baselines, we apply the event extraction approach described in [AB15] on Wikipedia. In that approach, events are identified by frequent collections of entity sets extracted from all sentences in the corpus that refer to at least one location and date, grouped by date. On Wikipedia, we demand for an itemset frequency of 5, which leads to a baseline set $B1$ of 91,367 events. When also allowing sentences without locations, this number rises to 274,150 ($B2$).

Table 5.3 shows the results for this comparison. There are few overlaps between the different event sets: Only 12.81% of the named events are found within the **CTR** events and 17.31% of the textual events can be mapped to **CTR** events (by demanding that they share at least two entities and similar time periods $ta \geq \theta$). Considering that only 7.20% of the textual events are found in the named events, these numbers confirm the aforementioned difficulty with respect to the different event representations, but also indicate the need to extend existing event collections. In both cases, we find clearly more trusted events than the frequency-baseline $B1$ (e.g. more than three times as many named events). Without the location restriction, more trusted events are found. However, the **CTR** result contains more than twice as many events in total compared to $B2$, which shows the potential of identifying events across sentences.

To compare our approach to another method that also aims at the extraction of events from the same resource i.e., a complete Wikipedia snapshot, we compare it to the temporal facts found in the YAGO knowledge base. Instead of searching for co-references

Method \ Trusted Events	Named Events	Textual Events
CTR	14,977 (12.81%)	23,406 (17.31%)
$B1$	4,536 (3.88%)	15,012 (11.10%)
$B2$	13,980 (11.95%)	22,886 (16.91%)

Table 5.3: Coverage-based comparison of our events and a frequency-based baseline with two sets of trusted events.

between entities across the pages, YAGO's relations are extracted from categories, textual patterns, and from structured information in the Wikipedia infoboxes [WZQ+10]. YAGO focuses on more static and concrete facts like birthdates and binary relations between entities, e.g. marriages or foundations. Table 5.4 shows the number of temporal YAGO relations that are also present in our event set using the **CTR** method (i.e. they share a time interval and the entities are part of the respective event in our set).

Relation	YAGO	CTR	Coverage
isMarriedTo	6,594	2,055	31.16%
created	1,372	387	28.21%
hasWonPrize	8,933	1280	14.33%
wasCreatedOnDate	973,464	185,079	19.01%
happenedOnDate	86,907	31,965	36.78%
diedOnDate	453,098	14,508	3.20%
wasBornOnDate	983,365	13,070	1.33%

Table 5.4: Overlap between temporal YAGO relations and **CTR** events grouped by selected binary and unary relation labels.

For the listed event relations found in both sets, we distinguish between unary ("wasBornOnDate") and binary ("isMarriedTo") relations. As our approach considers coreferences of entities, it is better at finding binary events that affect both involved entities. But even in the case of the unary YAGO relation "wasCreatedOnDate" (covering films, albums, organizations, etc.), our approach often provides additional information like the director, artist or founder. For marriages, YAGO finds relations not covered in our events because the "spouse" relation is an essential part of Wikipedia infoboxes about persons. As our approach is based on the unstructured Wikipedia page text, we can detect marriages only if covered in the entity's textual description.

Given that more than 248,344 events are shared between YAGO and our extracted entities, nearly 70%[4] of the events from our extraction cannot be matched with YAGO

[4]Consider that one of our events could be correctly mapped to multiple YAGO facts.

facts and therefore describe new information. Clearly, our set of events provides complementary information extending existing knowledge bases such as YAGO.

5.9.2 Qualitative Analysis

The coverage-based and clustering-based analyses have shown the high number and variety of new events found by using our approach, i.e. recall. Next, let us measure the quality of our results in terms of precision regarding the percentage of correctly retrieved events. To do this, it is not sufficient to compare our set of events against existing sources because (1) we aim at the extraction of events not representable by given temporal relations or named events in other sources and (2) if there is an overlap between the event sets we cannot easily detect which events match, as one of the sources could add some more descriptive entities to an event. Due to these issues, we decided to explicitly ask human annotators for the correctness of our extracted events.

In the user evaluation, we show users extracted events and ask them to categorize these events into one of the following categories:

- Specific real-world event (e.g. DNA discovery)
- Temporal relation during a specific period (e.g. football players of the same team in a specific season)
- Unrelated at the specified time

Through this, we identify two different categories of events: specific real-world events and temporal relations where the latter capture more general interactions of entities that often go beyond a single interaction between entities.

For the study, we randomly sampled 50 events per method. 17 users participated, categorizing 26 events on average, which results in a set of 450 ratings, where each event was rated by three users to allow for majority voting. Users were shown events one after another, and had to categorize them based on the involved entities and the timespan. As additional help we provided them with links to the entities' Wikipedia pages and the sentences the specific event was extracted from.

Method	Specific real-world event	Temporal relation	Unrelated
TR	67.39%	28.26%	4.35%
FTR	84.78%	15.22%	0%
CTR	88.00%	12.00%	0%
all	80.05%	18.50%	1.45%

Table 5.5: Precision of the extracted event sets.

Results

If at least two users agree on the class of an event, this class is chosen. Otherwise, the event is ignored in the following results analysis, which happened for 9 out of the 150 rated events only, even though the Fleiss' κ score of 0.11 suggests a slight agreement between the raters. This again shows how difficult it is to agree on what is an event and what is not an event, as also seen in the previous analyses in this section. Table 5.5 shows the study results: the **CTR** method outperforms both other method, given that 88.00% of the events in its sample were rated representing real-world events, and none of them not modeling any temporal relation at all, while only 67.39% of the results in the **TR** sample were rated as specific real-world events. Considering the event sets together, just 1.45% of our events were rated as unrelated and 98.55% of them were evaluated as real-world events or relations. We investigate some specific cases leading to disagreement between the raters or negative ratings:

- **Complex sentences**: For the event ⟨{Maryland Route 45, U.S. Route 111}, *1963*⟩, the sentence on the "U.S. Route 111" page is "The route disappeared in 1963, by then completely replaced by I-83, and running along I-83 except inside the Baltimore Beltway (where it used present Maryland Route 45 to the end)" which is difficult even for a human to understand and to map to an event.
- **Conceptual entities**: Events involving generic concepts like "Author", "Novelist" or "Actor" were often not rated as specific real-world events.
- **Erroneous time expressions**: Errors coming from the time extraction tool can be propagated to the final events.

Based on the above we conclude that our event extraction approaches are not only able to detect a vast amount of new events but also show high precision when finding

temporal relations and in particular events. Demanding for textual relatedness and strict connectivity constraints between the entities in an event contributes to the extraction of more precise events.

5.10 Discussion

With our methods, we were able to generate a corpus of high quality and heterogeneous historical and contemporary events. Due to the underlying assumption that important events are reported from the perspective of all participating entities, we may miss out further events of asymmetric importance among the participants.

By exploiting the clusters, we are able to find also this kind of events. From all sentences in Wikipedia that mention at least one entity and one time expression, we collect those sentences that have not been used to create our event sets. For each sentence, the closest cluster is identified using entity type and time overlap measures. If a sentence's distance to its closest cluster is below the clustering distance threshold, the sentence is considered to describe an event. We run this procedure over 21M additional Wikipedia sentences that contain both entities and time expressions and obtained close to 7M sentences potentially representing more events.

In this way, the characteristics induced from our event collections can help to identify further events with less connectivity constraints. As this extraction is solely based on sentences annotated with entities and time expressions, it could even be applied on other sources such as news collections.

5.11 Summary

We have introduced a novel approach to extract events exploiting temporal co-reference relationships between entities. We formally defined the problem, described its application on Wikipedia, and its implementation on a Wikipedia snapshot. Our results show that the extracted set of events is diverse with respect to the involved entities, occurrence times and event types, and we emphasize the role of word similarity in getting higher quality results. Comparing our results to existing event collections from Wikipedia and events extracted by YAGO, we find that the events we extract nicely complement and

extend existing collections and provide an important source of additional event information and temporal relationships. Measuring correctness of our results through a user study confirmed the high precision of our methods.

Chapter 6

Finding Synonymous Attributes in Evolving Wikipedia Infoboxes

Previous chapters present different approaches that extract data from unstructured and structured data that are not evolving. In this chapter and in the following chapter we deal with problems in which the data is evolving. In this context, we aim at extracting synonymous attributes, i.e. attributes that describe the same characteristic, on data that is changing over time. We focus our attention on Wikipedia Infoboxes that are semi-structured data structures organized in an attribute-value fashion. While policies establish for each type of entity represented in Wikipedia that the attribute names of the Infobox should selected from predefined templates. However, these requirements change over time and often users choose not to strictly obey them. As a result, it is hard to treat in an integrated way the history of the Wikipedia pages, making difficult to analyze the temporal evolution of Wikipedia entities through their Infobox and impossible to perform direct comparison of entities of the same type. To address this challenge, in this chapter we propose an approach to deal with the misalignment of the attribute names and identify clusters of synonymous Infobox attributes.

6.1 Contributions

The main contributions of this chapter can be summarized as follows.

- We introduce the problem of attribute name alignment in Wikipedia Infoboxes.

- We propose a novel approach to the problem of attribute name alignment in Wikipedia infoboxes that exploits co-occurrence information as a negative evidence and common attribute values as a positive evidence.
- We turn these evidences into metrics and treat the problem as a clustering problem, providing an efficient implementation of it that is based on lineal programming that provides a good approximation while maintaining a fast execution time.
- We apply the technique on a set of Wikipedia Infoboxes that span over 13 years and report our findings on how effective such approach is indeed.

6.2 Outline

The remainder of the chapter is structured as follows. Section 6.3 shows an illustrative example. In Section 6.4 we formally define the problem of attribute alignment in evolving Wikipedia infoboxes. The Section 6.5 introduces the principles our approach. Section 6.6 provides an extensive evaluation of our technique and reports our findings.

6.3 Motivating Example

Consider Figure 6.1, it contains four Infoboxes from different versions in time of the entity "Apple Inc.". Note the attribute that describes the location of the company. Initially (in 2007), the attribute name *location* was used to specify the country and other geographical data. In 2009 it disappeared. In 2014 two new attributes were introduced to indicate the location: the *location_country* and *location_city*. Finally, in 2017, the attributes were renamed to *hq_location_country* and *hq_location_city* to more accurately specify that the location is the location of the headquarters. The aforementioned example is also indicating another situation. The fact the same can happen to the attribute values. For instance, it can be noticed that the value indicating the country USA was originally "united states" and later changed to "u.s.".

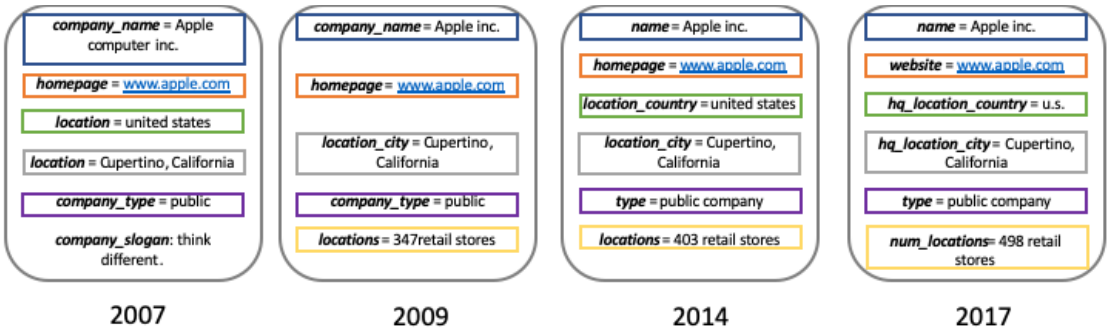


Figure 6.1: Evolution of schema and values of the entity Apple Inc.

6.4 Problem statement

This chapter deals with entities described in Wikipedia articles. We assume that an article describes only an entity, identified by an identifier *id* (e.g., the page title). The type T specifies the subject of the entity, e.g, an event, person, organization, product, etc. Wikipedia articles consist of two components: an unstructured textual component and a list of attribute-value pairs called *infobox*.

Definition 1 (Infobox and infobox schema)**.** We define the entity infobox $I = \{\langle a_1, v_1\rangle, ..., \langle a_n, v_n\rangle\}$, where $\langle a_i, v_i\rangle$ are attribute-value pairs. We denote with S_I the infobox schema, that is the set of attributes included within it, and with V_I its values.

For each type of entity T, Wikipedia policies specify a template for the infobox schema, i.e. the list of attributes that should describe that type of entity.

The data shown in an infobox may change over time. This happens mainly for two reasons: 1) the referred entity changes, i.e, the infobox values change and/or 2) Wikipedia releases new policies defining the infobox schema associated to a type of entity.

We define I_t the infobox at time t and E_t the entity at time t it is describing.

Definition 2 (Entity)**.** An entity at time t, denoted as E_t, is a triple $\langle id, T, I_t\rangle$ where id is the entity identifier, T is the entity type and I_t the associated infobox.

The set of all the changes occurred to the entity can be collected from all the infoboxes and constitutes the *entity evolution*.

Definition 3 (Entity evolution)**.** Assuming the existence of a set of times values $\mathcal{T}$ that correspond to all possible times instances t_i, we define entity evolution E as the triple $\langle id, T, I_{\mathcal{T}} \rangle$ where id is the entity identifier, T is the entity type and $I_{\mathcal{T}}$ the set of all infoboxes I_{t_i} describing the entity over time. We identify with $S_{I_{\mathcal{T}}}$ the schema of $I_{\mathcal{T}}$, that is the union of the schemas of all the infoboxes contained within it. Similarly, we define $V_{I_{\mathcal{T}}}$ as the set of values in all the included infoboxes.

The problem we want to address is to find, for each entity type, lists of synonymous attributes, i.e., attributes that are used over the time to describe the same property of an entity. The set of attributes used in the infoboxes of a specific entity type is called *set of entity type attributes.*

Definition 4 (Set of entity type attributes)**.** For each entity type T, A^T is the set of entity type attributes and includes all attributes used in at least an infobox schema at any time for describing an entity of type T.

Synonymous attributes are clusters of entity type attributes that describe the same real-world entity property.

PROBLEM *1 (Finding synonymous attributes) Given a set of entity type attributes A^T, we want to find a disjoint partitioning of A^T, denoted as $S = \{S_1, ..., S_m\}$, where the attributes $a_j \in S_i$ are used to describe the same real-world entity property.*

Furthermore, in the following, we denote:

- $V_I(a)$ as the set of values assumed over time by the attribute a within all the infoboxes $I_{\mathcal{T}}$ associated with an entity;
- $\Delta t_I(a)$ as the time interval (i.e., a list, even if not contiguous, of time instants) in which attribute a is *valid*, i.e. it appears in some infoboxes $I_{\mathcal{T}}$ associated with an entity;
- $\mathcal{I}$ as the set of the infoboxes $I_{\mathcal{T}}$ collected over time for a collection of entities.

6.5 The approach

In this section, we present our proposal for finding synonymous attributes in Wikipedia entities having the same type. For each pair of attributes, two measures are computed,

assessing the extent in which the attribute represent (and do not represent) the same entity property, respectively. In this way, they provide a positive and a negative evidence of the synonymy. The measures are presented in Section 6.5.1. Then Section 6.5.2 shows how to use the knowledge provided by these measures to generate clusters of synonymous attributes. For this purpose, we reduce our problem to the one addressed by correlation clustering [BBC04], where data points are partitioned into groups based on their similarity. A linear-programming approach has been adapted for this purpose. The work has been inspired from [HCCT16], where a similar technique has been adopted in the context of web search engines.

6.5.1 Positive and negative evidence for synonymy

We can model the synonymy relationship between attributes by analyzing their co-occurrences in the same infobox. In this perspective, we assume that synonymous attributes cannot appear simultaneously in the same infobox otherwise there would be information redundancy. In other words, we leverage the co-occurrence of two attributes in the same infobox as a negative evidence for their synonymy.

Example 6.1. *Consider for example the attributes name and type which definitely describe different aspects of an entity. They are very common attributes: in a random sample of* 60,760 *infoboxes describing companies collected over the last 13 years, they appeared, together or separately, in* 74.61% *of the cases (i.e. for describing* 45,332 *entities). Within these entities the attributes coexist in the same infobox in* 99.89% *of the cases (i.e.,* 45,281 *times), and they do not co-occur* 51 *times. According to our idea, they cannot be considered as synonyms. Conversely, the name and company_name attributes, that instead can be used to describe the same characteristic of an entity, show an inverse co-occurrence pattern: in* 0.08% *of the cases are present simultaneously in the same infobox and in* 99.92% *of the cases do not co-exist.*

More formally, given two attributes a_i, a_j belonging to infoboxes describing the same entity type, Equation 6.1 provides a measure of their "negative" co-occurrence.

$$NegCoocc(a_i, a_j) = \frac{|\{I_{\mathcal{T}} \in \mathcal{I} | \Delta t_I(a_i) \cap \Delta t_I(a_j) \neq \varnothing\}|}{|\{I_{\mathcal{T}} \in \mathcal{I} | a_i, a_j \in S_{I_{\mathcal{T}}}\}|} \tag{6.1}$$

To compute this measure, all infoboxes in the collection are evaluated. For each of them, the presence of both input attributes (i.e., a_i and a_j) in a time frame is verified. This check is carried out by identifying whether there are overlaps in their validity time interval (i.e., $\Delta t(a_i)$ and $\Delta t(a_j)$ respectively). The number of entities for which there is overlap, normalized by the overall number of entities in $\mathcal{I}$, provides the correlation value that we consider as negative evidence for their synonymy.

As the experimental evaluation shows, the adoption of this measure only is not enough to accurately identify the synonymous attributes.

Example 6.2. *The highest values obtained by the application of Equation 6.1 to the attribute company_logo, are with the attributes logo, name and type. The results, in our collection are respectively 0.9988, 0.965 and 0.94. Obviously, only the first pair of attributes are synonyms. The other pairs are attributes representing very different information. After a careful analysis of the temporal evolution of the infobox schemas we noticed that the attributes with the "company" prefix have been introduced with an old Wikipedia policy to identify all attributes describing "company" type entities. Today, this policy is no longer adopted, in favor of more concise and direct attributes (such as type and name instead of company_type and company_name respectively). However, a delay in the application of the new policy produces misalignments in the infoboxes and make Equation 6.1 not enough accurate.*

To produce more accurate results, we introduce a measure for positive synonymy evidence. In particular, we measure the values shared between attributes as the indication that they are really synonyms. We analyze the different value representations of the attributes throughout the entire history of Wikipedia and we calculate their fraction of overlap through the Jaccard similarity. We do not deliberately consider other string similarity techniques to have a more general approach, which does not rely on specific domain knowledge. In more detail, for each pair of attributes we select the values that generate the maximum fraction of overlap within the data collection. Equation 6.2 provides the formulation of the measure we adopt.

$$PosOverlap(a_i, a_j) = \frac{\sum_{I_\mathcal{T} \in \mathcal{I} | a_i, a_j \in S_{I_\mathcal{T}}} max\Big(\sum_{v_i \in V_I(a_i), v_j \in V_I(a_j)} Jaccard(v_i, v_j) \Big)}{|\{I_\mathcal{T} \in \mathcal{I} | a_i, a_j \in S_{I_\mathcal{T}} w\}|} \tag{6.2}$$

where, with reference to the notation introduced in Section **??**, $V_I(a_i)$ and $V_I(a_j)$ represent respectively all values assumed over time by the attributes a_i and a_j for all infoboxes in $I_{\mathcal{T}}$ where they are valid.

Example 6.3. *Let us consider the application of Equation 6.2 with the same input as in Example 6.2. We obtain the following results:* $PosOverlap(company\_logo, logo) = 0.8899$*;* $(company\_logo, name) = 0.003$*; and* $(company\_logo, type) = 0.007$*. We can observe that the high value computed for the pair* $(company\_logo, logo)$ *confirms the previous evidence of synonymy. The very low values for the other pairs do not confirm the evidence of synonymy resulting from Equation 6.1.*

6.5.2 Holistic approach for synonym discovery

The measures of synonymy between pairs of attributes are used to compute clusters of synonymous attributes which constitute the result of our work. Our idea is to model the synonymy relations between the attributes by means of a graph and to apply a clustering algorithm over the graph to extract groups of synonymous attributes.

Given some positive and negative evidence for attributes synonymy, we model attributes and their synonymy relationship as an *attribute-synonymy graph*, where the nodes correspond to the attributes and the edges to the synonymy relations between the attributes. The edges are labeled according to whether the measure associated with them should be interpreted as positive or negative evidence of synonymy.

Definition 5 (*attribute-synonymy graph*)**.** An *attribute-synonymy graph* is a graph $G = (V, E)$ with vertices representing the attributes of the infoboxes we want to analyze. The edges associate to each pair of vertices provide a measure of their synonymy through a weight $w_{i,j} \geq 0$. Let $L_{i,j}$ be the label associated to each edge (i, j). L can assume the value $+$ or $-$ according to whether the edge is representing the measure of the negative or the positive evidence for their synonymy expressed by Equation 6.1 and Equation 6.2, respectively. Let E^+ be the set of edges identified by a label of value $+$: $E^+ = \{(i, j)|L_{i,j} = +\}$, and, analogously, E^- (i.e., $E^- = \{(i, j)|L_{i,j} = -\}$) the set of edges identified by a label of value $-$. A representation of this graph is provided in Figure 6.2, where solid edges indicate edges with positive weights and edges with crosses the negative ones.

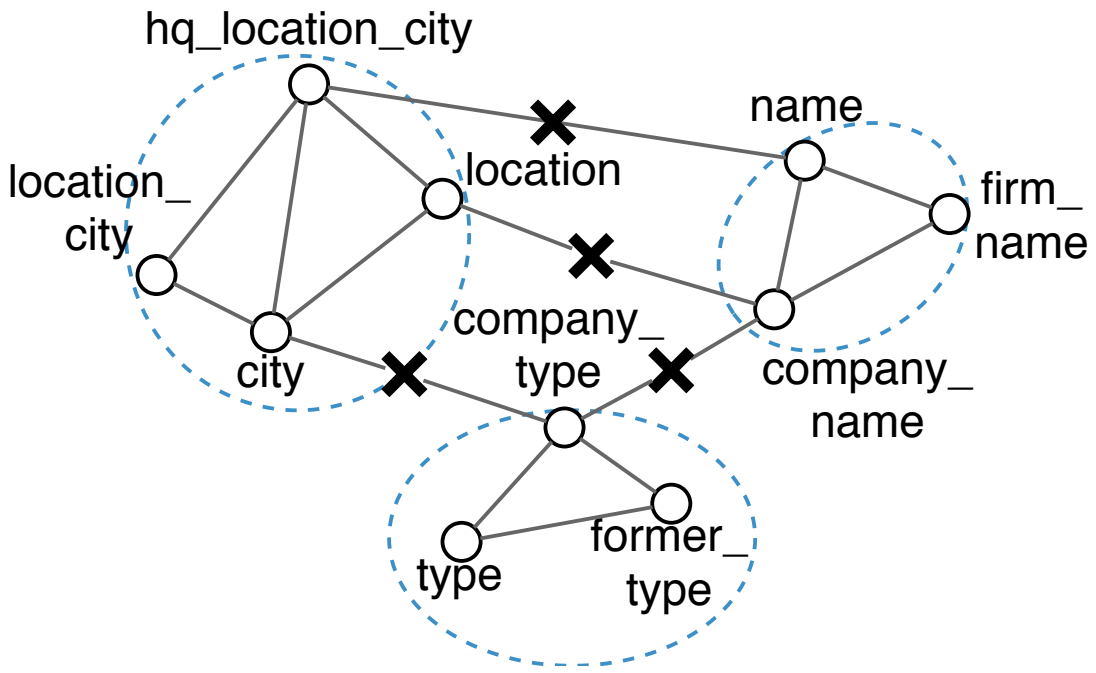


Figure 6.2: Attribute-synonym graph for "company" type entities

Our goal now is to apply a clustering strategy that partitions the nodes of the *attribute-synonymy graph* so that each attribute is associated with a single cluster with its synonyms (see the dashed blue circles of Figure 6.2). To obtain this result, we adopt a correlation clustering algorithm [BBC04] which provides a method for clustering data points into the optimum number of clusters based on their similarity without specifying that number in advance. In our implementation, the aim is to identify the partitioning of the infobox attributes that best respects the positive and negative evidence of synonymy provided as input.

PROBLEM *2 (discovery of synonymous infobox attributes) Given an* attribute-synonymy graph $G = (V, E)$, *we want to find a disjoint partitioning of* V, *denoted as* $\mathcal{S} = \{S_1, ..., S_m\}$, *that agrees as much as possible with the labels* L *associated to the edges* E *of the* attribute-synonymy graph. *More precisely, we want a clustering that maximizes the weight of agreements: the weight of + edges within clusters plus the weight of - edges between clusters.*

The resolution of this problem exploits a heuristic procedure already proposed in the literature [DEFI06] for solving the correlation clustering problem. This technique is divided into two steps. First a linear programming approach is used to provide an approximate solution to the problem. The results produced by this model are fractional

values that correspond to scores of synonymy between attributes. In a second step a technique called region-growing is applied to group attributes with a high synonymy level within the same cluster and remove the attributes that describe different information about the referred entity.

Linear-programming approach In the first phase of the approach the following linear model has to be solved.

$$\begin{aligned} &\text{minimize} && \sum_{(i.j)\in E^-} w_{i,j}(1 - x_{i,j}) + \sum_{(i.j)\in E^+} w_{i,j}x_{i,j} \\ &\text{subject to} && x_{i,j} \in [0,1], \qquad x_{i,j} + x_{j,k} \geq x_{i,k}, \qquad x_{i,j} = x_{j,i}. \end{aligned}$$

The goal of this model is to identify a valid assignment of the variable $x_{i,j}$ that minimizes the sum of the negative edges included in a cluster and maximizes the sum of positive edges. Intuitively, this variable provides an indication of the collocation of the nodes in the clusters (i.e., it assumes, in the borderline cases, the value 0 when two attributes are included in the same cluster and 1 in the opposite case). An assignment of $x_{i,j}$ is considered valid if $x_{i,j} \in [0,1]$ and $x_{i,j}$ satisfies the triangular inequality. This motivates the inclusion of the constraints in the problem formulation. The adaptation of this linear model to our problem requires the addition of a further constraint, which requires that the negative weights (i.e., $w_{i,j}$ in the first sum) are defined according to Equation 6.1, and the positive ones (i.e., $w_{i,j}$ in the second sum) according to Equation 6.2.

Region growing Once a first approximated solution to the problem is obtained, we apply the region growing technique. Its objective is to convert the approximate cluster membership indication of the attributes provided by this first solution, into an exact distribution of the attributes in the different clusters. More precisely, this technique is used to convert the fractional solution x in an integral solution which identifies if two attributes belong to the same cluster. Since this technique represents a classical clustering strategy, below we provide only an insight into its operation. More details instead can be found in [DEFI06]. The intuition behind this technique is to construct, in an iterative way and starting from randomly selected seed nodes, some balls (i.e., groups of graph nodes) modifying, step by step, their coverage radius on the graph. The growth of these balls is determined by the weights associated with the graph edges: a ball will continue to grow as the sum of the positive weights included inside the subgraph

identified by the ball is advantageous. On the contrary, the ball will stop growing, causing the creation of a new ball (or a new cluster), when its growth would incorporate dissimilar nodes compared to those already included in the cluster. The arrangement of these balls within the graph determines its final partitioning.

6.6 Experimental evaluation

In this section, we firstly provide a description of the dataset used for the experimental evaluation (Section 6.6.1) and then we qualitatively (Section 6.6.2) and quantitatively (Section 6.6.3) evaluate the effectiveness of the approach. Finally, a case study is presented (Section 6.6.4) to show how Wikipedia synonymous attributes can be used in a real scenario.

6.6.1 Dataset description

The dataset used in the experimental evaluation is a collection of infoboxes of entities having type associated to the "concept of company" (i.e., we consider entities having type company, organization, dot-com company, etc.). This collection includes, for each entity, its complete history between August 2004 to August 2017, i.e. all updates in the infobox schemas that have been introduced by Wikipedia users. The result is $60,760$ entities and around $1,861,252$ changes.

The number of attributes used in the infoboxes varies: it is not fixed per entity type and in the time. Table 6.1 provides some statistics about attributes and values. The average number of attributes and values per entity are 12.84 and 25.35 respectively. Moreover, the maximum number of attributes associated to an entity, in the considered period of time, is equal to 253, and the maximum number of different values is 503.

In Tables 6.2 and 6.3 the top 10 most frequent attributes and values are reported. The attribute "name" is the most used: it appears in 97% of the collected entities, while "company_name", appearing in 47% of the entities, is the 10th most used attribute. Concerning the values, "united states", "privately held companies" and "public companies" appear respectively in 31.78%, 31.63% and 23.47% of the entities and are the most frequently values used in the collection.

	avg	std	max	min
# attribute per entity	12.84	5.64	253	1
# value per entity	25.35	23.67	503	1

Table 6.1: Number of attributes and values per entity

attr	freq	freq (%)
name	59017	97.13%
industry	51845	85.33%
foundation	49033	80.70%
homepage	47076	77.48%
type	46015	75.73%
logo	40102	66.00%
key_people	36176	59.54%
products	33388	54.95%
location	32490	53.47%
company_name	28565	47.01%

Table 6.2: Frequencies of attributes and values: Top 10 most frequent attributes

value	freq	freq (%)
united states	19310	31.78%
privately held company	19221	31.63%
public company	14259	23.47%
united states dollar	9692	15.95%
private	7983	13.14%
subsidiary	7144	11.76%
united kingdom	5986	9.85%
worldwide	5973	9.83%
yes	5289	8.70%
chief executive officer	4793	7.89%

Table 6.3: Frequencies of attributes and values: Top 10 most frequent values

Table 6.4 and Table 6.2 provides an insight on the evolution of the attributes and values in the considered period of time. In particular, Table 6.2 shows the attributes whose values were most frequently subject to change, and Table 6.5 the top 10 entities affected by the greatest number of changes over time. Note that 10% of all the infobox updates involves the "key_people" attribute, and the most modified entity is "Eurosport".

A more detailed analysis of the evolution is shown in Figure 6.3 and Table 6.6, where the top 5 most updated types of entity are analyzed. Table 6.6 shows the number of entities collected per type and the total number of changes. Figure 6.3 plots some statistics about the number of updates per entity. Although the total number of updates in the "company" category is the highest, a particularly high number of average updates

attr	freq	freq (%)
key_people	162465	10.07%
products	112440	6.97%
location	94936	5.89%
foundation	87828	5.45%
industry	86707	5.38%
homepage	84182	5.22%
name	77894	4.83%
logo	62653	3.88%
type	61655	3.82%
revenue	59206	3.67%

Table 6.4: Updates in Wikipedia entries: Top 10 most changed attributes

entity title	freq	freq (%)
Eurosport	1924	0.10%
National Geographic (TV channel)	708	0.04%
Canada	672	0.04%
Apple Inc.	594	0.03%
Nintendo	580	0.03%
HBO	538	0.03%
Cuba	527	0.03%
Animax Asia	526	0.03%
General Motors	525	0.03%
Amazon.com	509	0.03%

Table 6.5: Updates in Wikipedia entries: Top 10 most changed entities

has been applied to entities belonging to the "television" type. The other categories of entities, on the other hand, present an average number of updates which is approximately the same (i.e., the range varies between 20 and 40 updates).

entity type	# entities	# total changes
company	57,553	1,494,245
defunct company	664	15,304
dot-com company	127	6,340
television	40	4,953
organization	155	4,819

Table 6.6: Entities and updates for the top 5 most updated entity types.

6.6.2 Qualitative evaluation of the effectiveness

In this section we qualitatively evaluate the effectiveness of our approach by analyzing a sample of its results. Table 6.7 shows 10 clusters of synonymous attributes generated

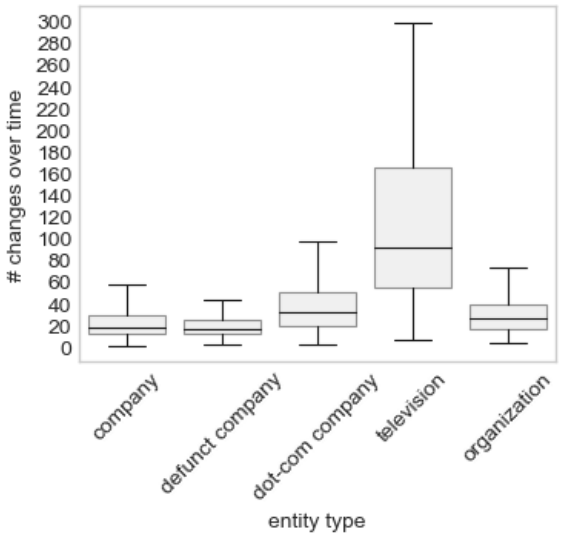


Figure 6.3: Number of changes of the top 5 most updated entity types.

by our approach. We can observe that our approach is able to identify interesting and non-trivial synonymy relations between attributes. For example, it is able to find the correspondences between attributes like "established" and "founded" or "predecessor" and "former_name" which would not be identifiable by a string similarity technique. Furthermore, we match attributes expressed in different languages, such as "employees" with "mitarbeiterzahl" and "city" with "sitz". Analyzing these results, we can observe the various textual forms used over time by the Wikipedia community to indicate the same characteristic of an entity. This variety of forms presumably derives from the adoption of different schema guidelines/policies imposed by Wikipedia[1]. The attributes "name", "company_name" and "type", "company_type" are examples of this situation. The inclusion of the prefix "company" has been introduced by a policy to make more explicit the type of entity described by the attributes.

6.6.3 Quantitative evaluation of the effectiveness

The effectiveness of the proposed approach is assessed in quantitative terms. The main goal of this analysis is to empirically demonstrate that both the measures contribute in identifying synonymous attributes. To perform this evaluation, firstly a ground truth has been manually created. We have exploited public attribute mappings directly provided by the Wikipedia Template pages to obtain a first minimal set of attribute matches.

[1]Note a cleaning procedure has been applied to the input infoboxes to remove the "noise" generated by human mistakes.

cluster
num_staff, employees, number of employees, num_employees, numemployees, mitarbeiterzahl
established, opened, formation, founded_date, start_year, date_founded, foundation, gründungsdatum, introduced, founded
logo, non-profit_logo, network_logo, company_logo, firm_logo
web, url, website
operating_profit, ebitda, operating income, operating_income
creator, founder(s), founder, founders
predecessor, former_names, former_name, predecessors
company_type, type, unternehmensform, former type, former_type, company type, non-profit_type
headquarters, headquaters, hq_city, location_city, city, sitz, residence, hq_location_city, location, place, hq_location
agency_name, network_name, group_name, name, non-profit_name, company_name, firm_name, company name

Table 6.7: Example of synonymous attributes produced by our approach

This basic information has been then extended with new manually inserted attribute correspondences. The generated ground truth includes about 2,000 attributes clustered in 454 groups of synonyms. Once an exact set of attribute correspondences has been generated, we evaluated our approach on a sample of the entire collection of Wikipedia infoboxes. In more detail, we tested our approach on an *attribute-synonymy graph*, generated starting from the input dataset, consisting of 6,854 attributes and 52,707 synonymy relations. The results provided by our approach were finally compared with the ground truth.

To provide a measure of the quality of the clusters generated by our approach with respect the ground truth, we adopted four measures: precision, recall, f1 score and rand index. We calculate precision as $\frac{\#\ true\ synonyms\ in\ cluster}{\#\ total\ attributes\ in\ estimated\ cluster}$, and recall as $\frac{\#\ true\ synonyms\ in\ cluster}{\#\ total\ attributes\ in\ real\ cluster}$. The f1 score is a combination of precision and recall defined as $2 * \frac{precision * recall}{precision + recall}$. Finally, the rand index [MR71] was used to evaluate the similarity between the clustering solution produced by our approach and that provided by the ground truth.

The experiment aims, in particular, to evaluate the contribution of each measure in obtaining the final result. To force such behavior, a linear combination of the two measures has been introduced. Its formulation is proposed in Equation 6.3, where the α parameter is used to weight the contribution of the measures.

$$SynonymyScore(a_i, a_j) = \alpha * PosOverlap(a_i, a_j) + (1 - \alpha) NegCoocc(a_i, a_j) \quad (6.3)$$

The results of this experimentation are given in Table 6.8. The results show that, with reference to the company entity type, linear combinations that assign more importance to the positive evidence of synonymy produce better results. Table 6.8 shows only α ranging from 0.6 to 1, however with lower α results follow a similar trend: precision decreases and instead recall increases. The best configuration is α = 0.8, that obtains the highest values in all evaluation measures. We observe that the configuration with α = 1, where there is no contribution from the negative evidence of synonymy, is the one that obtain the highest precision level. Nevertheless, in that configuration, the recall, rand and f1 score levels decrease considerably.

α parameter (positive contribution)	precision	recall	f1 score	rand index
0.6	0.375	0.764	0.372	0.146
0.7	0.817	0.759	0.757	0.886
0.8	0.797	0.767	0.760	0.947
0.9	0.791	0.754	0.752	0.942
1	0.831	0.668	0.723	0.858

Table 6.8: Effectiveness evaluation with different positive contributions

6.6.4 Case study

In this section we provide a small case study to show that synonymous attributes can support the extraction of high quality and accurate information from Wikipedia. Table 6.9 introduces 5 information needs a user would like to satisfy against the collection of infoboxes described in Section 6.6.1. Each information need has been transformed into 2 structured queries: one with the original filtering condition formulated by the user and the second where the attributes have been substituted with a number of disjunctive clauses, each one expressing the same information need but by using synonymous attributes. Table 6.9 shows the number of entities retrieved when both the queries are executed and the number of results it is expected to be retrieved. Last two columns show the same information in percentage. We observe that synonymous attributes largely support the retrieval of all results. The maximum improvement is obtained with the last

query	# query results				
	original clauses	using synonymous attributes	ground truth	original clauses (%)	using synonymous attributes (%)
location_city="tokyo"	268	293	298	89.93	98.32
founded<1900	681	705	705	96.59	100
location="USA", year=2014	381	531	531	71.75	100
location="united states", net_income>1 billion	371	429	429	86.48	100
type="public company", num_employees>10,000, year=2010	221	1154	1154	19.15	100

Table 6.9: Case study

query (i.e., `type="public company", num_employees>10,000, year=2010`) where the application of synonymous attributes allows us to retrieve all results, instead of 19.15% of them, as we obtain with the original formulation.

6.7 Summary

We introduced an approach that automatically defines clusters of synonymous temporal-evolving infobox attributes. The approach is mainly based on two kinds of knowledge: a negative evidence of synonymy provided by co-occurrences of the attributes in the same infobox in a given time instance, and a positive evidence of synonymy generated by co-occurrences of similar values for the attributes in different time instances. We formalized this issue as a correlation clustering problem over a weighted graph and we used a linear programming model to solve it. Our experiments, over the last 13 years infoboxes history, shows that our approach is effective in discovering synonymous attributes.

Chapter 7

Finding Movements of Values in Evolving Wikipedia Infoboxes

In the previous chapter, we treat the problem of finding synonymous attributes from evolving infoboxes. Evolving datasets, such as Wikipedia Infoboxes, open opportunities for new information to be extracted. In this context, we analyze how attributes values are moving between the different entities during a vast time period of time. In this chapter, we formally introduce the concept of movements of attributes, we propose an effective strategy for the identification of the movements, and we validate our results with real data.

7.1 Contributions

The main contributions of this chapter can be summarized as follows.

- We formally define the problem of attribute movement identification.
- We propose an effective strategy that identifies the movements through a pruning procedure that first remove frequent and systematic changes, then we score the remaining movements in order to select only relevant movements.
- We apply our approach on a set of Wikipedia Infoboxes that spans over a decade, and we discuss our shortcomings and possible further research directions.

7.2 Outline

The chapter is organized as follows. In Section 7.3, we present a motivating example. Section 7.4 formally defines movements; Section 7.5 introduces our approach in order to find movements between evolving Infoboxes. Finally, Section 7.6 contains the experimental evaluation.

7.3 Motivating Example

Consider the manager "Ursula Burns" that in 2017 left its CEO position at Xerox to later join the company VEON. On the data perspective, on the infobox of entity "Xerox" the attribute "Key people = Ursula Burns" changed to "Key people = Jeff Jacobson" at the beginning of 2017. In 2018, after a certain time the attribute "Key people" was changed with "Ursula Burns" in the entity VEON. We identify this kind of transfer as an attribute movement. This movement represents both the changes in jobs of the entity "Ursula Burns" and the change of essential figures between two top companies.

7.4 Problem Statement

In our data model the fundamental component is the entity, an artifact that correspond to a real world object. An entity is a data structure composed of a unique identifier and a set of attributes representing its aspects at a specific point time.

More formally, we assume the existence of an infinite set of identifiers $\mathcal{O}$ that corresponds to all the possible real objects, an infinite set of names $\mathcal{N}$ and an infinite set of atomic values $\mathcal{V}$.

An attribute is a pair $\langle n, v \rangle$, with $n \in \mathcal{N}$ and $v \in \{\mathcal{V} \cup \mathcal{O}\}$. Let $\mathcal{A} = \mathcal{N} \times \{\mathcal{V} \cup \mathcal{O}\}$ be the infinite set of all possible attributes. Thus, an attribute name $n \in \mathcal{N}$ describes the properties that characterize aspect of the entities $e \in \mathcal{E}$ such as name, address, and many others and this correspond to column of a in table or as the predicate in an RDF-triple. Values characterize these properties, and they can be null $\perp$.

The world is naturally evolving, and consequently entities need to reflect these changes. They are reflected in the attributes of the entities, where attributes values can be added, updated or deleted. Thus, each attribute is associated with a time validity period in order to capture all the changing that affect an entity.

We assume the existence of an infinite set of times values $\mathcal{T}$ that correspond to all the the possible times instances. Let a time-aware attribute be a tuple $\langle n, v, t_b, t_f \rangle$, with $n \in \mathcal{N}$, $v \in \{\mathcal{V} \cup \mathcal{O}\}$ and $t_b, t_e \in \mathcal{T}$, where t_b and t_f represent the begin and end time of the validity period of the attribute. We indicate with $\mathcal{A}^t = \mathcal{N} \times \{\mathcal{V} \cup \mathcal{O}\} \times \mathcal{T} \times \mathcal{T}$ the infinite set of possible time-aware attributes.

Definition 1. A time-aware entity e is a tuple $\langle id, A^t \rangle$ where $id \in \mathcal{O}$ is the entity identifier and $A^t \subseteq \mathcal{A}^t$, with A^t finite, is the set of time-aware attributes of the entity.

For simplicity we will refer to the time-aware entities as entities. As an example, consider a person that changes his/her address due to a new job, the attribute that represents the address changes its value in order to capture this change. The previous address was valid until the time that this changed.

For each time-aware attribute $a^t = \langle n, v, t_b, t_f \rangle \in A^t$ of an entity $e = \langle id, A^t \rangle$ we can define a time-aware entity attribute a^t_e as the tuple:

$$a^t_e = \langle id, n, v, t_b, t_f \rangle$$

Note that since we allow entity identifiers to appear as attribute values, our model supports also relationships between entities.

Whenever a change occurs in the value of an entity attribute, we say that the entity *evolved*. Changes can be of different kinds, i.e., updates, insertions, or disappearances: an update is characterized by the existence of two distinct tuples $\langle id, n, v', t'_b, t'_f \rangle, \langle id, n, v'', t''_b, t''_f \rangle \in A^t$, where $v' \neq v''$ and $t'_e < t''_b$, indicating that the value of n changed from v' to v''; an appearance is characterized by the existence of a tuple $\langle id, n, v, t_b, t_f \rangle$, where t_f is undefined; and a disappearance of a value v is characterized by the existence of a tuple $\langle id, n, v, t_b, t_f \rangle$, where t_f is defined.

A database D is a finite set of entities $\mathcal{E}$, where each entity partially models a real-world object. Since we live in a community where these objects (e.g. people, companies and

others) depend on other objects, we assume that the attribute values of an entity can change because of changes in the attributes of other entities.

In this work, we want to analyze how attribute values move between entities. The migration of a value v from an entity e_1 to an entity e_2 is called a *move*.

Definition 2. A *move* $m(v, e_1, e_2)$ of a value v from an entity e_1 to an entity e_2 is a tuple $m(v, e_1, e_2) = \langle v, e_1, e_2, t_d, t_a \rangle$ where $t_d = t_f^i$ for some $a_i = \langle e_1, n, v, t_b^i, t_f^i \rangle \in A_1^t$, $t_a = t_b^j$ for some $a_j = \langle e_2, n, v, t_b^j, t_f^j \rangle \in A_2^t$, and $t_f^i < t_b^j$.

By definition a move $m(v, e_1, e_2)$ occurs if and only if the value v disappears from e_1 and later appears in some attribute of e_2. We define the set of possible movements in a database D as $\mathcal{M}_D = \{m(v, e_i, e_j) | e_i, e_j \in \mathcal{D} \wedge e_i \neq e_j \wedge v \in \{\mathcal{V} \cup \mathcal{O}\}\}$.

For example, consider "Ursula Burs" that leaves a company "Xerox" to join another company "Veon". In this case we can say that the employee *moved* from the entity "Xerox" to the entity "Veon".

Notice that not every disappearance of an attribute value of an entity e_1 is related to a later appearance of the same value in another entity e_2, because appearances may occur by chance. As an example consider a public company e_1 that becomes "private" at time t_1 and a private company e_2 that becomes "public" at time $t_2 > t_1$. It is clear that the first update did not caused the second update, and therefore we would not consider these change as a move of "public" from e_1 to e_2.

Since not every change represents a move, we need a way to measure how likely a value actually moved from one entity to another one. Thus, we assume the existence of function $i : \mathcal{M}_D \to [0, 1]$ called *movability* score, which associates a level of relevance to each candidate move $m(v, e_1, e_2)$.

Finally, notice that a value may move between different entities multiple times. For example, an employee may change company more than once.

Problem 3. *(The movement identifiication problem.) Given a set of time aware entities $\mathcal{A}^t$, our goal is to identify all movement $\mathcal{M}_D$ of attribute values $v \in \mathcal{V}$ that move between those entities.*

7.5 Movement extraction

The extraction of movements from the dataset is a challenging task. Since an attribute movement is originated from a change, first we need to identify the changes that are not generating movements, and then we score the remaining possible movements in order to find the most relevant ones.

7.5.1 Systematic and frequent changes

We recall that not every change in an attribute can be considered a move of a value between two entities. Indeed, the amount of moves is negligible with respect to the number of changes that happened in the data. As we will see later, for each change of attribute value, we must perform a quadratic number of comparisons between entities in order to generate the candidate movements involving that value. Therefore, a crucial step of our approach is the identification of efficient and effective strategies to identify only the changes that most likely indicate actual movements. Among the changes that less likely suggest a move of a value between entities, we first focus on the *changes at system level* and the *changes in close-domain attributes*.

A change at system level is a change forced by the system (i.e., an external source) independently from the evolution of the entities. In Wikipedia, for example, the value "USA" has been progressively replaced by the value "U.S." due to a new data representation policy.

On the other hand, a change in a close-domain attribute happens when an attribute value changes from v to v' with $v, v' \in R \subseteq \{\mathcal{V} \cup \mathcal{U}\} \wedge |R| < k$ for some $k << |\mathcal{V} \cup \mathcal{U}|$. We disregard these changes, because the number of evolving entities is huge compared to the number of values that the attribute can take, and hence it is very likely that a value disappears from an entity and appears in another entity at the same time just by chance. An example, the field "type of company" in the Wikipedia infoboxes can assume value only among "private" and "public", and therefore it will likely happen that different companies change their company type to opposite values in similar time periods.

In order to prune set of values that are generating changes at system level and the changes in close-domain attributes we need to reconstruct the evolution of each attribute name

for each entity. To construct the evolutions of the attribute names of an entity e, we first split the set of time-aware attributes A_e^t into a set of sets $A_{e\restriction n}^t$, each of which contains the attributes referring to the same attribute name n, i.e., $A_{e\restriction n}^t = \{\langle \bar{n}, v, t_b, t_f\rangle | \langle \bar{n}, v, t_b, t_f\rangle \in A_e^t \wedge n = \bar{n}\}$. We propose two different approaches to create these sequences of values.

The first method models the evolution of n as a set of ordered sequences of values that n took during the lifespan of e, i.e., $S_{e\restriction n} = \{[v_1, \ldots, v_k] \,|\, \exists a_1, \ldots, a_k \in A_{e\restriction n}^t \wedge \forall i = 1, \ldots, k\,,\, a_i = \langle n, v_i, t_{b_i}, t_{f_i}\rangle \wedge t_{f_i} < t_{f_{i+1}}\}$. The second method, instead, models the evolution of n as a set of ordered pairs of values took by n, i.e., $S_{e\restriction n} = \{[v_1, v_2] \,|\, \exists a_1, a_2 \in A_{e\restriction n}^t \wedge \forall i = 1, 2\,,\, a_i = \langle n, v_i, t_{b_i}, t_{f_i}\rangle \wedge t_{f_1} < t_{f_2}\}$. The union of all the sets $S_{e\restriction n}$ is denoted by $S_D = \bigcup_{e \in D} \bigcup_{n \in N_e} S_{e\restriction n}$ where $N_e \subset \mathcal{N}$ is the finite set of attribute names of the entity e.

Note that an attribute name can take more than a value at a time. For example, in the Wikipedia infobox of the entity "Apple Inc." the attribute "Founders" takes 3 values (i.e., Steve Jobs, Steve Wozniak, and Ronald Wayne), and the attribute "Key People" takes as many as 10 different values. As a consequence, in the first method, the set $S_{e\restriction n}$ can contain multiple sequences, while in the second method, multiple pairs can share a common value v_1 or v_2.

Furthermore, we implemented a strategy, which can be embedded in both the two methods, in order to deal with two edge cases: constant value and disappeared value. The first case happens when an attribute name n of an entity e does not change its value v during the lifespan of e, while the second case happens when the value v of the attribute disappears without being replaced by a new value. In both cases, the set $S_{e\restriction n}$ would be empty, and thus v would not be considered in the next steps of the algorithm. However, both these two kinds of values can convey important information, and therefore should be somehow included in the set S_D. Therefore, we introduce the suffix $*$ to indicate that a value is constant, and the suffix # to indicate that a value disappeared without being replaced. Then, we define $S_{e\restriction n}$ as $\{[v, *v]\}$ in the first case, and as $\{[v, \#v]\}$ in the second case.

We claim that the changes at system level and the changes in close-domain attributes can be identified by means of the frequent itemsets that can be extracted from S_D. We formally denote an itemset c in S_D as the set $c = \{v_1, \ldots, v_l\}$ where $\exists S_{e\restriction n} \in S_D \,.\, \exists s \in$

$S_{e\restriction n} \wedge c \subseteq s$, and the support $sup(c)$ of c as $sup(c) = \{s|s \in S_D \wedge c \subseteq s\}/|S_D|$. Then, a frequent itemset is defined as follows:

Definition 3. Given a threshold τ, an itemset c is a *frequent itemset* if $sup(c) > \tau$.

In particular, we use the frequent itemsets to generate association rules that likely contain values representing one of the two changes we want to discard. Association rules are implications in the form of $X \rightarrow Y$, where $X, Y \subseteq \mathcal{V}$. The support of an association rule is simply the support of the union of the two sides of the implication, i.e., $sup(X \rightarrow Y) = sup(X \cup Y)$, while its confidence is defined as $conf(X \rightarrow Y) = sup(X \cup Y)/sup(X)$.

Note that both kinds of changes will appear in rules with high confidence. Indeed, changes at system level affect almost every entity in the dataset and therefore they will be included in a large number of sequences $s \in S_D$, while changes in close-domain attributes are likely very frequent due to the narrow set of possible values that that attribute can take. Therefore, we introduce two threshold α and β, and discard the rules $X \rightarrow Y$ with support $sup(X \rightarrow Y) \leq \alpha$ and confidence $conf(X \rightarrow Y) \leq \beta$. The excluded values are inserted into the set W_f. The values appearing in the rule kept will be excluded in the step of movement extraction described in the following section.

Example 7.1. *Consider a set the set of infoboxes related to organizations in Wikipedia. They consist of* 57400 *infoboxes that span over 13 years. In this dataset the number of possible movements is greater that* $57M$.

Since the candidates are greater than 57 M, we need the initial pruning phase. Note that we consider only values that are links to other entities. Table 7.1 shows the distribution of the movements with respect to the distinct values. With only 100 values we are able to discover more than 55 millions of candidate movements that correspond to more that 96% of the total number of possible movement with remain set of values. Table 7.2 shows the number of candidate movements generates per each value. Values are displayed in decreasing order with respect to the number of candidate movements generated. Only the value "privately held company" is able to generate more than 12 millions of possible movements.

As shown in the previous example, the number of movement is considerably high with respect to the number of entities that is quite slow. Given this enormous amount of

#Values	#Movements	Coverage (%)
100	55529952	0,9669
200	56420671	0,9825
300	56782902	0,9888
400	56972754	0,9921
500	57078997	0,9939
1000	57285782	0,9975
2000	57372584	0,9990
3000	57396333	0,9994
5000	57413333	0,9997
10000	57424724	0,9999

Table 7.1: Number of distinct attribute values with respect to the candidate movements generated. The values are ordered by the ones that are generating more movements.

Value	#Movements
privately held company	12035712
united states	10852849
public company	9875540
united states dollar	9006635
chief executive officer	2286292
united kingdom	2097759
subsidiary	1084830
chairman	607206
london	524468
financial services	497412
TOTAL	48868703

Table 7.2: Top 10 values that generate movements.

movement generated by this small amount of values, we first apply the removal process for systematic and frequent changes. This process will generate a set of remaining values that we denote with W_f. Given the rest of the values, the process of generating the candidate movement is described in Algorithm 16.

7.5.2 Candidate movement extraction

Given the set values excluded W_f obtained from the previous, we now extract the set of remaining candidate movements.

The algorithm takes as input the database D containing the set of entities and returns the movements related to those entities. As a first step, the algorithm retrieves the

Algorithm 16 Candidate Movement Extractor

Input: D: a database
Output: $\mathcal{M}_D$, the movement database generated from D

```
1: M_D ← {}
2: W ← retrieveValues(D)
3: Ŵ = W ∖ W_f
4: for each w ∈ Ŵ do
5:     E ← extractEntities(w)
6:     A_w^t ← getAttributes(E, w)
7:     sort(A_w^t)
8:     for each a_1 ∈ A_w^t do
9:         A_w^{t+} ← getFurtherAttributes(A_w^t, a_1)
10:        for each a_2 ∈ A_w^{t+} do
11:            if a_1.id! = a_2.id then
12:                if a_1.t_f < a_2.t_b then
13:                    M_D.add(⟨w, a_1.id, a_2.id, a_1.t_f, a_2.t_s⟩)
14:                else
15:                    break;
16:                end if
17:            end if
18:        end for
19:    end for
20: end for
21: return M_D
```

candidate values that can move between entities. Candidate values are values that appear in more than one entity and values have to disappear in order to be considered.

The function retrieveValues() is responsible for the extraction of candidate value. From the set of candidate value W, we remove the frequent values extracted in the previous phase producing the set $\hat{W}$. Therefore, for each of these selected values $\hat{W}$ we generate the temporal valid movements present in the database D. The extraction process is composed of several steps.

First, for each value $w \in \hat{W}$, we extract all the entities E that contains this specific value w with the function extractEntities(), from this set of entities E we collect all the attributes A_w^t that contains the value w using the function getAttributes(). This set of attributes A_w^t is ordered by the end validity time and for each element $a_1 \in A_w^t$, the function getFurtherAttributes() retrieves the set of attributes A_w^{t+} that has begin time greater the end time of the analyzed attribute a_1, so $A_w^{t+} = \{a_2 | a_2.t_b > a_1.t_f \wedge a_2 \in A_w^t\}$. For each $a_1 \in A_w^t$ we compare with the elements in the set A_w^{t+} and then we collect all the movements ($\mathcal{M}_D$) that are temporally valid.

7.5.3 Ranking of candidate movements

The set of candidate movements $\mathcal{M}_D$ identifies the set of candidate movements that remain for the previous filtering procedure. This candidate movements are not equally important and represent real movements of values between the entities. In order to identify significant movements we define a scoring function that identifies the important characteristics of a movement. Each candidate movement will be associated with a score named movability score that represents how important of a movement m be. Given the set of candidate movements $\mathcal{M}_D$ we score all of them and discard movements with a score lower than γ.

The movability score highlights different aspects of the movement that qualify which movements are more important than the one that is not identified as movements.

A movement is generated from the disappearance and the appearance of a value within a certain amount of time, this fact imposes that the number of disappearances and appearances of a value have to be balanced. To measure that we introduce a scoring function called *bal* that uses entropy in order to measure the diversity between appearances and disappearances of a value in the history of a database D.

$$bal(m(e_1, e_2, v)) = -\sum_{i=1}^{n} P(x_i) log(P(x_i))$$

where $e_1, e_2 \in \mathcal{E}$, $v \in \mathcal{V}$ and X is a discrete random variable that contains two values, for each couple of entities. The first represents the probability of appearance of a specific attribute value $w \in \{\mathcal{V} \cup \mathcal{O}\}$ in a certain amount of time Δt in which the value w disappears from the entity e_1, and it appears in the entity e_2.

Another characteristic that qualifies a movement are the entities in which the values are moving around, in general, the entities in which the values are floating around are similar. In order to qualify this aspect, we define a similarity function between two entities:

$$simE(e_1, e_2) = J(Names(e_1), Names(e_2))$$

	avg	std	max	min
# attribute per entity	12.84	5.64	253	1
# value per entity	25.35	23.67	503	1

Table 7.3: Number of attributes and values per entity

where $Names : \mathcal{E} \rightarrow \mathcal{N}$ return all the attribute names $N \subseteq \mathcal{N}$ of the entity e_1. Moreover, $J : \mathcal{E} \times \mathcal{E} \rightarrow [0, 1]$ is the Jaccard similarity.

The final score is computed by the mean of the two previous scores: *bal* and *simE*. Finally, when all the movements have been scored, we used a threshold τ in order to the most probable movements.

7.6 Experimental Evaluation

This section descibes first the data that we used to show the potential of our method and then it provides details and examples of the rest of the process.

Dataset description. The dataset used is composed by a collection of infoboxes that are related to the entities of type company. This collection includes each entity, its complete history between 2004 and 2018. The result is 57465 entities.

The number of attributes used in the infoboxes varies: it is not fixed per entity type and in the time. Table 7.3 provides some statistics about attributes and values. The average number of attributes and values per entity are 12.84 and 25.35 respectively. Moreover, the maximum number of attributes associated to an entity, in the considered period of time, is equal to 253, and the maximum number of different values is 503.

In Tables 7.4 and 7.5 the top 10 most frequent attributes and values are reported. The attribute "name" is the most used: it appears in 97% of the collected entities, while "company_name", appearing in 47% of the entities, is the 10th most used attribute. Concerning the values, "united states", "privately held companies" and "public companies" appear respectively in 31.78%, 31.63% and 23.47% of the entities and are the most frequently values used in the collection.

Given our input dataset, composed of more than $50k$ entities, we first need to build our sequences of values as described in the previous section. The number of sequences generated is 949970. These sequences are then processed in a different manner (pairwise

attr	freq	freq (%)
name	59017	97.13%
industry	51845	85.33%
foundation	49033	80.70%
homepage	47076	77.48%
type	46015	75.73%
logo	40102	66.00%
key_people	36176	59.54%
products	33388	54.95%
location	32490	53.47%
company_name	28565	47.01%

Table 7.4: Frequencies of attributes and values: Top 10 most frequent attributes

value	freq	freq (%)
united states	19310	31.78%
privately held company	19221	31.63%
public company	14259	23.47%
united states dollar	9692	15.95%
private	7983	13.14%
subsidiary	7144	11.76%
united kingdom	5986	9.85%
worldwide	5973	9.83%
yes	5289	8.70%
chief executive officer	4793	7.89%

Table 7.5: Frequencies of attributes and values: Top 10 most frequent values

splitting or constant/end case addition). The distribution of sequences is shown in Figure 7.1.

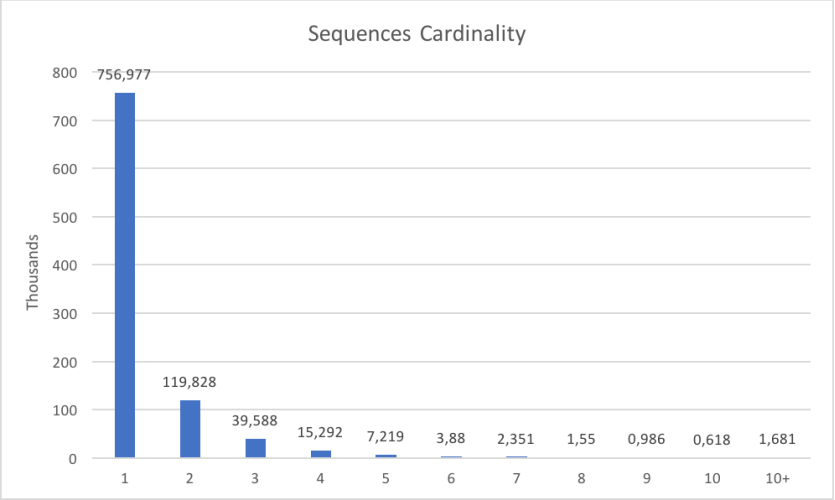


Figure 7.1: Distribution of sequence cardinality

Table 7.6 presents some results about the pruning phase and the candidate movements

extraction process. Given the different type of sequence creation process, we show the effectiveness and efficiency of the pruning approach. The column *Sequence Type* indicates the different sequence construction used: "Normal" indicates the sequences built without any addition the constant and ending case. Instead, *Pair* indicates the other sequential strategy that is based on pairs, and the suffix E, C indicates the usage of the strategy that includes extra cases of constant and ending cases. Support and confidence values are reports and for each of them the number of values excluded by frequent itemset, the values excluded by association rules and the number of candidate movements that have been generated with these settings.

Sequence Type	Support	Confidence	#Vs excluded by FI	#Vs excluded by AR	Candidate Movements
$Normal$	0,001	0,001	59	0	3735829
$Normal$	0,0001	0,001	862	46	299838
$Normal$	0,00005	0,001	1554	93	144257
$Normal$	0,00004	0,001	1900	128	112423
$Normal$	0,00003	0,001	2406	182	83542
$Normal$	0,00002	0,001	3497	327	54251
$Normal$	0,00001	0,001	6700	740	25934
$Normal_{E,C}$	0,001	0,001	96	71	3735829
$Normal_{E,C}$	0,0001	0,001	1490	1196	299838
$Normal_{E,C}$	0,00005	0,001	2712	2179	144257
$Normal_{E,C}$	0,00004	0,001	3316	2656	112423
$Normal_{E,C}$	0,00003	0,001	4219	3376	83542
$Normal_{E,C}$	0,00002	0,001	6154	4916	54251
$Normal_{E,C}$	0,00001	0,001	11432	8747	25934
$Pair_{E,C}$	0,001	0,001	84	61	3997222
$Pair_{E,C}$	0,0001	0,001	1329	1008	345946
$Pair_{E,C}$	0,00005	0,001	2480	1892	163802
$Pair_{E,C}$	0,00004	0,001	3026	2297	125895
$Pair_{E,C}$	0,00003	0,001	3873	2921	94277
$Pair_{E,C}$	0,00002	0,001	5494	4136	61590
$Pair_{E,C}$	0,00001	0,001	10271	7303	29644

Table 7.6: Effectiveness of pruning of Systematic and frequent changes removal.

7.6.1 Quantitative evaluation of the effectiveness

The quantitative evaluation wants to analyze that our approach is able to identify the movements that happen in the data. To perform this evaluation, we first manually create a ground truth. The generated ground truth includes the movements of 16672 values, and a set of labelled movements greater than $40M$.

To provide a measure of the quality of the clusters generated by our approach with respect to the ground truth, we adopted three measures: precision, recall, and f1 score.

We calculate precision as

$$\frac{\#\ find\ movements}{\#\ total\ movements}$$

and recall as

$$\frac{\#\ true\ synonyms\ in\ cluster}{\#\ total\ attributes\ in\ real\ cluster}$$

. The f1 score is a combination of precision and recall defined as

$$2 * \frac{precision * recall}{precision + recall}$$

.

The experiment aims, in particular, to evaluate the contribution of each measure in obtaining the final result. In order to find our final set of movements we used support = 0.00017, confidence = 0.05 and γ = 0.3. In this setting, we obtain a set of candidate movements of 15149, and we score them with the scoring function introduced in previous settings. After selecting candidate movements with a score greater than $\gamma > 0.45$, we obtain a set of candidate movements composed of 6529 movements.

After comparing it with the groundtruth we obain a precision = 0.83 and recall = 0.99. These results are part of a preliminary work that will surely be improved with future extensions.

7.7 Summary

In this chapter, we present our approach that discovers movements of attributes between evolving entities. We first formalized the problem, then propose an approach that can identify the correct movements with good quality. This work is at an initial state but it shows the potential of our idea.

Chapter 8

Conclusions

Big data analysis is the process of extracting knowledge from a huge amount of data that is constantly increasing. Since the number of users accessing data in any format, from different sources, is exploding, the process of extracting information from those data has to adapt to this emerging need. This is becoming even critical when novice users are involved in this process because they do not have any experience in the treatment of such amount of data. The problems start from the acquisition of the data, if users are not completely aware of the structure, the content and the information present in their input dataset, they are not able to proceed with their analysis.

This dissertation aims at constructing an initial study. More precisely we proposed methods to cope with very common problems in data analysis: (1) the generation of data description, in which the user provides some parameters and system provides the top-k description to the user; (2) the order dependency discovery, which deals with the problem of finding list of columns that when ordered they order other columns; and with (3) the extraction important information from textual data, in particular the extraction of historical events from annotated documents. Furthermore, we also analyze the dynamics of the evolving datasets: (4) with the extraction of synonymous attributes of Wikipedia Infoboxes and (5) the extraction of movements of attributes values between entities over time.

8.1 Key Contributions

In this dissertation, we study the problem of information extraction with particular attention on the discovery of important types of information that could help the user understanding the characteristic of a dataset. This dissertation provides five significant contributions in this area by proposing a data explanation system that provides descriptions of an input dataset [PSM+19a], a new method for the discovery of order dependencies on relational data [CSMV19]; the extraction of historical events from text corpora [SGNV19] and the identification of synonymous of attributes [SPGV19a] and movements from evolving Wikipedia Infoboxes. For each problem, we provide algorithmic solutions and experimental evaluation. We also build a prototype to show the efficiency of our explanation system [PSM+19b].

8.1.1 Data Description

We started our investigation from the problem of data description. We introduce the problem of generating data descriptions: a set of compact, readable and insightful formulas of boolean predicates that represents a set of data records. We demonstrate that is a hard problem, so we introduce a dynamic programming approach which, in concert with a set of heuristics, allow us not only to generate descriptions at interactive speed but also to accommodate diverse user needs — from anomaly detection to data exploration. Using real datasets, we evaluate our approach both quantitatively and qualitatively and prove that descriptions are indeed a viable and powerful tool for supporting data enthusiasts and practitioners in gaining insights from data.

8.1.2 Order dependencies discovery

We continue with a relevant task in the exploration and understanding of large datasets: the discovery of hidden relationships in the data. In particular, functional dependencies have received considerable attention in the past. However, other kinds of relationships are significant both for understanding the data and for performing query optimization. Order dependencies belong to this category. An order dependency states that if a table is ordered on a list of attributes, then it is also ordered on another list of attributes. Our

approach leverages the observation that discovering order dependencies can be guided by the discovery of a more specific form of dependencies called order compatibility dependencies. We show that our algorithm outperforms existing approaches on real datasets.

8.1.3 Historical event discovery

We presented our third contribution that aims to extract information from textual data. To allow this data to reach its full potential, there has been an increasing interest in extracting structural information from it. A well-studied topic towards that direction is entity extraction, that aims at identifying references and information about real world entities. An equally important, but relatively understudied, the topic is event extraction, i.e., the identification of real world events that have occurred at some point in time. In this direction, we provide a novel approach to event extraction that is based on temporal co-reference among entities. We consider an event to be a set of entities that collectively experience some relationships between them in a specific period of time. This translates to mutual reference across such entities for some specific time period. We formally define the problem, describe three different solution implementations. We also show our finding with an exhaustive experimental evaluation.

8.1.4 Finding Synonymous Attributes in Evolving Wikipedia Infoboxes

Our fourth contribution, the identification of synonymous attributes of evolving Wikipedia Infoboxes, deals with the evolving nature of the data. Despite a lot of attention has been devoted in the case of static datasets, very few attention has been spent in the context of evolving data. However, since our storage capacity is increasing, there is the possibility to store and preserve the data that has change over time. In this direction, we study the attribute alignment or schema matching problem in the context of the Wikipedia Infoboxes. Our approach leverages two different observation that we map into an instance of the correlational clustering problem. We provide an approximate solution that shows the potential of our approach in the reconciliation of attributes that represent the same concept.

8.1.5 Finding Movements of Values

Finally, our last contribution, the discovery of movement attributes in evolving datasets, tackles the evolving nature of the datasets, and tries to find complex evolving patterns in the data. While other work has been done in order to find insight from evolving data, the problem of discovery movements has never been explored. We provide a formal definition of a movement and present an experimental approach that tries to leverage different observations to identify the values that really transfer to other entities.

8.2 Extensions and open problems in Evolving Wikipedia Infoboxes

The work done in this dissertation, the studied problems and the proposed solutions provide the need for some additional direct extensions classified with respect to the different chapters presented in this dissertation.

8.2.1 Data Description

Incremental Generation of Descriptions. We demonstrate that our data description approach is able to provide descriptions in a very short time period also with large datasets. Suppose that the user is using our framework in an interactive fashion, in such a scenario, our approach will re-compute all the d-formulas and description at each iteration of the user. This is extremely inefficient and time consuming, so it will negatively impact on the usability. For these reasons, as an immediate extension, we would like to integrate a caching strategy that is able to dramatically reduce the computational time. Also, dependencies between data may improve the performance since we can exclude some trivial descriptions based on the actual dependencies that hold in the data.

Descriptions over multiple Datasets. So far we have studied data descriptions over single tables and datasets. However, several scenarios require to deal with multiple tables or datasets. For example, this is the case of the relational databases, due to normalization, where data is spread in different tables. In such as scenario, our approach needs to be adapted in order to deal with this data separation, for example,

we can use other dependencies such as inclusion dependencies, functional or order dependencies to help the description process.

8.2.2 Order dependency discovery

Approximate order dependency discovery. In Chapter 4, we presented our approach for finding order dependencies. This approach discovers all the exact dependencies holding in a specified dataset. However, in practice, many times order dependencies may not hold exactly in the data, this is due to errors in the data. In order to identify these errors, as a future extension, we plan to extend our algorithm for the discovery of approximate order dependencies that hold in the data. The identification of the pieces of the data that invalidate the exact order dependencies may be corrected in order to improve the quality of the data.

Distributed order dependency discovery. Regarding order dependency discovery, we to extend our work with a distributed version of our algorithm. Despite our work outperforms existing work on the order dependency discovery, our experimental evaluation shows an explosion of candidate dependencies when increasing the number of columns. In order to make it scale, we could study and implement a distributed version of the algorithm with a distributed framework like Hadoop[1] or Spark[2].

8.2.3 Event Extraction

Multilingual event extraction. In our experiments related to event extraction approach presented in Chapter 5, the English Wikipedia was used. However, the proposed approach is language-independent. As long as tools for sentence splitting and time extraction are available, events from each Wikipedia language editions could be extracted. While this could lead to a single set of events per language, we also consider to establish co-references across languages – e.g., a reference to Barack Obama on the English and French Wikipedia page of Donald Trump in an overlapping time span could

[1] https://hadoop.apache.org/
[2] http://spark.apache.org/

be considered a temporal co-reference.

NLP for event extraction. To give precise results, our approach is solely based on sentences, so that less wrong assignments between entity references and time references are inferred. Additionally, for recall, we do not put any constraints on the semantics, e.g. by using NLP dependency parsing. In future work, dependency parsing could be used to establish more fine-grained events. For instance, the consideration of negations could increase precision, while we could potentially also increase recall by applying an approach that exceeds sentence borders.

Event extraction on different corpora. Document resources such as news collections or websites usually do not follow Wikipedia's model where each page is constantly referring to the same entity. Therefore, our methods need to be slightly updated in order to extract entity relationships and events from other resources. For example, we can establish new entity relationships by exploiting the closeness of entity mentions. As soon as the co-reference graph is constructed with such data, our methods can be used in the same manner as for the graph induced by Wikipedia.

8.2.4 Finding Synonymous and Movements in Wikipedia Infoboxes

Find synonymous attribute values. Our approach that discovers synonymous attributes in evolving attributes is able to cluster attribute names in a very precise and effective way. However, the same issue that affects attribute names also affects the set of values. To this end, in future work, we may integrate into our approach a second process that identifies synonymous attribute values. For example, we can use evidence from both attributes and value synonymous in order to improve the effectiveness of our approach.

Find movements in other contexts. The movement identification is only an initial step in the direction of the discovery of new insights from evolving dataset. Our approach provides initial work that could be extended in many directions. For example, we may apply machine learning techniques to classify candidate movements or we can discover

movements in other context such as free text (Wikipedia pages or news articles) or on tabular data.

List of Figures

List of Tables

Bibliography

[AB15] Abdalghani Abujabal and Klaus Berberich. Important events in the past, present, and future. In *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, pages 1315–1320, 2015. 8, 28, 29, 133

[ABD+15] Eleanor Ainy, Pierre Bourhis, Susan B. Davidson, Daniel Deutch, and Tova Milo. Approximated summarization of data provenance. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 483–492, 2015. 20

[ABK+07a] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007. 8, 120

[ABK+07b] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 722–735, 2007. 30

[ACD+98] James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, Yiming Yang, et al. Topic detection and tracking pilot study: Final report. In *Proceedings of the DARPA broadcast news transcription and understanding workshop*, volume 1998, pages 194–218. Citeseer, 1998. 8, 28

[ADD+11] Yael Amsterdamer, Susan B. Davidson, Daniel Deutch, Tova Milo, Julia Stoyanovich, and Val Tannen. Putting lipstick on pig: Enabling database-style workflow provenance. *PVLDB*, 5(4):346–357, 2011. 20

[ADM+15a] Tim Althoff, Xin Luna Dong, Kevin Murphy, Safa Alai, Van Dang, and Wei Zhang. Timemachine: Timeline generation for knowledge-base entities. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 19–28, 2015. 15

[ADM+15b] Tim Althoff, Xin Luna Dong, Kevin Murphy, Safa Alai, Van Dang, and Wei Zhang. Timemachine: Timeline generation for knowledge-base entities. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 19–28, 2015. 27, 128

[AGN15] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015. 6

[AGNP18] Z. Abedjan, L. Golab, F. Naumann, and T. Papenbrock. *Data Profiling*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018. 2

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216. ACM Press, 1993. 63

[AKSS12] Albert Angel, Nick Koudas, Nikos Sarkas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6):574–585, 2012. 28

[ALNZ13] Sören Auer, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, and Amrapali Zaveri. Introduction to linked data and its lifecycle on the web. In *Reasoning*

Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings, pages 1–90, 2013. 6

[Arm74] William Ward Armstrong. Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583, 1974. 25, 71, 75

[AS17] Prabal Agarwal and Jannik Strötgen. Tiwiki: Searching wikipedia with temporal constraints. In *Proceedings of WWW*, pages 1595–1600, 2017. 30

[ASW09] Eytan Adar, Michael Skinner, and Daniel S. Weld. Information arbitrage across multi-lingual wikipedia. In *Proceedings of WSDM*, pages 94–103, 2009. 31

[BBC04] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004. 143, 146

[BBJ+18] Tobias Bleifuß, Leon Bornemann, Theodore Johnson, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. Exploring change - A new dimension of data analytics. *PVLDB*, 12(2):85–98, 2018. 13, 31

[BBK+18] Leon Bornemann, Tobias Bleifuß, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. Data change exploration using time series clustering. *Datenbank-Spektrum*, 18(2):79–87, 2018. 13, 32

[BBK+19] Tobias Bleifuß, Leon Bornemann, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. Dbchex: Interactive exploration of data and schema change. In *Proceedings of CIDR*, 2019. 13, 31

[BBR11] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011. 30

[BDI09] Gosse Bouma, Sergio Duarte, and Zahurul Islam. Cross-lingual alignment and completion of wikipedia templates. In *Proceedings of the Workshop on Cross Lingual Information Access*, pages 21–29. Association for Computational Linguistics, 2009. 31

[BFG+07] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning.

In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 746–755, 2007. 25

[BK73] Coenraad Bron and Joep Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973. 116

[BKG14] Gabriel Bender, Lucja Kot, and Johannes Gehrke. Explainable security for relational databases. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1411–1422, 2014. 20

[BKT01] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, pages 316–330, 2001. 20

[BMQ+15] Marcello Buoncristiano, Giansalvatore Mecca, Elisa Quintarelli, Manuel Roveri, Donatello Santoro, and Letizia Tanca. Database challenges for exploratory computing. *SIGMOD Record*, 44(2):17–22, 2015. 4, 24

[BMR11] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011. 30

[BOZ12] Nurzhan Bakibayev, Dan Olteanu, and Jakub Zavodny. FDB: A query engine for factorised relational databases. *PVLDB*, 5(11):1232–1243, 2012. 23

[CIP13] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 458–469, 2013. 6, 25

[CLMR16] Zaheer Chothia, John Liagouris, Frank McSherry, and Timothy Roscoe. Explaining outputs in modern data analytics. *PVLDB*, 9(12):1137–1148, 2016. 20

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009. 38

[CM12] Angel X. Chang and Christopher D. Manning. Sutime: A library for recognizing and normalizing time expressions. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, pages 3735–3740, 2012. 29, 126

[CSMV19] Cristian Consonni, Paolo Sottovia, Alberto Montresor, and Yannis Velegrakis. Discovering order dependencies through order compatibility. In *International Conference on Extending Database Technology (EDBT)*, 2019. 7, 170

[CW03] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12(1):41–58, May 2003. 20

[CWR14] Lei Cao, Qingyang Wang, and Elke A. Rundensteiner. Interactive outlier exploration in big data streams. *PVLDB*, 7(13):1621–1624, 2014. 24

[DD89] Hugh Darwen and C Date. The role of functional dependencies in query decomposition. *Relational Database Writings*, 1991:133–154, 1989. 25

[DEFI06] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006. 11, 146, 147

[DFA+17] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. The data civilizer system. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017. 2

[DG15] Benjamin Dietrich and Torsten Grust. A SQL debugger built from spare parts: Turning a SQL: 1999 database system into its own debugger. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 865–870, 2015. 4, 22

[DH82] Jirun Dong and Richard Hull. Applying approximate order dependency to reduce indexing space. In *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data, Orlando, Florida, USA, June 2-4, 1982.*, pages 119–127, 1982. 6, 15, 26

[DHKS06] Gautam Das, Vagelis Hristidis, Nishant Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 395–406, 2006. 24

[DJHM13] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *I-SEMANTICS 2013 - 9th International Conference on Semantic Systems, ISEM '13, Graz, Austria, September 4-6, 2013*, pages 121–124, 2013. 126

[DMT14] Daniel Deutch, Yuval Moskovitch, and Val Tannen. A provenance framework for data-dependent process analysis. *PVLDB*, 7(6):457–468, 2014. 20

[DZS13] Ankur Dave, Matei Zaharia, and I Stoica. Arthur: Rich post-facto debugging for production analytics applications. Technical report, UC Berkeley, 2013. 22

[Ebb13] Hermann Ebbinghaus. *Memory: A Contribution to Experimental Psychology*. 1913. 109

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007. 8

[ES07] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 2007. 30

[FAK+18] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. Aurum: A data discovery system. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 1001–1012, 2018. 14

[FG12] Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012. 6, 25

[FL11] Daniel Fabbri and Kristen LeFevre. Explanation-based auditing. *PVLDB*, 5(1):1–12, 2011. 20

[For73] Jr. Forney, G.D. The Viterbi algorithm. *Proceedings of the IEEE*, 1973. 48

[GH81] Seymour Ginsburg and Richard Hull. Ordered attribute domains in the relational model. In *XP2 Workshop on Relational Database Theory, June 22-24 1981, The Pennsylvania State University, PA, USA*, 1981. 6, 25

[GH83] Seymour Ginsburg and Richard Hull. Order dependency in the relational model. *Theor. Comput. Sci.*, 26:149–195, 1983. 6, 25, 26

[GH86] Seymour Ginsburg and Richard Hull. Sort sets in the relational model. *J. ACM*, 33(3):465–488, 1986. 6, 25

[GIY+16] Muhammad Ali Gulzar, Matteo Interlandi, Seunghyun Yoo, Sai Deep Tetali, Tyson Condie, Todd D. Millstein, and Miryung Kim. Bigdebug: debugging primitives for interactive big data processing in spark. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, pages 784–795, 2016. 22

[GKS11] Lukasz Golab, Flip Korn, and Divesh Srivastava. Efficient and effective analysis of data quality using pattern tableaux. *IEEE Data Eng. Bull.*, 34(3):26–33, 2011. 15

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40, 2007. 20

[GLS+13] Salvador García, Julián Luengo, José Antonio Sáez, Victoria López, and Francisco Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Trans. Knowl. Data Eng.*, 25(4):734–750, 2013. 43

[GR13] Torsten Grust and Jan Rittinger. Observing SQL queries in their natural habitat. *ACM Trans. Database Syst.*, 38(1):3:1–3:33, 2013. 22

[HCCT16] Yeye He, Kaushik Chakrabarti, Tao Cheng, and Tomasz Tylenda. Automatic discovery of attribute synonyms using query logs and table corpora. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 1429–1439, 2016. 11, 30, 143

[HKPT99] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999. 7, 25

[HL12] Daniel Hienert and Francesco Luciano. Extraction of historical events from wikipedia. In *The Semantic Web: ESWC 2012 Satellite Events - ESWC 2012 Satellite Events, Heraklion, Crete, Greece, May 27-31, 2012. Revised Selected Papers*, pages 16–28, 2012. 8, 28, 29, 133

[HSBW13] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013. 30

[IC15] Ihab F. Ilyas and Xu Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015. 15

[IPC15] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 277–281, 2015. 14

[IPW11] Robert Ikeda, Hyunjung Park, and Jennifer Widom. Provenance for generalized map and reduce workflows. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 273–283, 2011. 20

[IST+15] Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, Seunghyun Yoo, Miryung Kim, Todd D. Millstein, and Tyson Condie. Titian: Data provenance support in spark. *PVLDB*, 9(3):216–227, 2015. 4, 20

[JGP16] Manas Joglekar, Hector Garcia-Molina, and Aditya G. Parameswaran. Interactive data exploration with smart drill-down. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 906–917, 2016. 22, 42

[KBS12] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Perfxplain: Debugging mapreduce job performance. *PVLDB*, 5(7):598–609, 2012. 4, 22

[KHM+06] Holger Kache, Wook-Shin Han, Volker Markl, Vijayshankar Raman, and Stephan Ewen. POP/FED: progressive query optimization for federated queries in DB2. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 1175–1178, 2006. 15

[KL08] Chunyu Kit and Xiaoyue Liu. Measuring Mono-word Termhood by Rank Difference via Corpus Comparison. *Terminology. International Journal of Theoretical and Applied Issues in Specialized Communication*, 14(2):204–229, 2008. 121

[KLD11] Bhargav Kanagal, Jian Li, and Amol Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 841–852, 2011. 21

[KM03] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan.*, pages 423–430, 2003. 126

[KM10] Carl-Christian Kanne and Guido Moerkotte. Histograms reloaded: the merits of bucket diversity. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 663–674, 2010. 23

[KNPZ16] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, and Xiaojin Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 19–34, 2016. 24

[KPP+12] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Profiler: integrated statistical analysis and visualization for data quality assessment. In *International Working Conference on Advanced*

Visual Interfaces, AVI 2012, Capri Island, Naples, Italy, May 22-25, 2012, Proceedings, pages 547–554, 2012. 24

[KVW14] Erdal Kuzey, Jilles Vreeken, and Gerhard Weikum. A fresh look on knowledge bases: Distilling named events from news. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1689–1698, 2014. 28

[KW12] Erdal Kuzey and Gerhard Weikum. Extraction of temporal facts and events from wikipedia. In *2nd Temporal Web Analytics Workshop, TempWeb '12, Lyon, France, April 16-17, 2012*, pages 25–32, 2012. 8, 27, 28, 29, 128

[LBL16] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1675–1684, 2016. 4, 22, 42, 60, 61, 63

[LDY13] Dionysios Logothetis, Soumyarupa De, and Kenneth Yocum. Scalable lineage capture for debugging DISC analytics. In *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 17:1–17:15, 2013. 20

[LJ12] Alexandros Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. *PVLDB*, 5(12):2032–2033, 2012. 1, 2

[LLL+12] Chen Lin, Chun Lin, Jingxuan Li, Dingding Wang, Yang Chen, and Tao Li. Generating event storylines from microblogs. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 175–184, 2012. 8, 28, 128

[LLLC12] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. Discover dependencies from data - A review. *IEEE Trans. Knowl. Data Eng.*, 24(2):251–264, 2012. 5, 6, 25

[LN16] Philipp Langer and Felix Naumann. Efficient order dependency detection. *VLDB J.*, 25(2):223–241, 2016. ix, 6, 7, 27, 79, 88, 92, 94, 95, 96, 102, 179

[MBS15] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015. 8

[MDM+14] Guido Moerkotte, David DeHaan, Norman May, Anisoara Nica, and Alexander Böhm. Exploiting ordered dictionaries to efficiently construct histograms with q-error guarantees in SAP HANA. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 361–372, 2014. 23

[MGMS10] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010. 21

[ML08] Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*, pages 236–244, 2008. 118

[MR71] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 12 1971. 152

[MRS14] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. Causality and explanations in databases. *PVLDB*, 7(13):1715–1716, 2014. 20

[NHTW14] Dat Ba Nguyen, Johannes Hoffart, Martin Theobald, and Gerhard Weikum. Aida-light: High-throughput named-entity disambiguation. In *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014.*, 2014. 126

[NMN+11] Thanh Nguyen, Viviane Moreira, Huong Nguyen, Hoa Nguyen, and Juliana Freire. Multilingual schema matching for wikipedia infoboxes. *PVLDB*, 5(2):133–144, 2011. 31

[OR11] Christopher Olston and Benjamin Reed. Inspector gadget: A framework for custom monitoring and debugging of distributed dataflows. *PVLDB*, 4(12):1237–1248, 2011. 22

[OZ15] Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015. 23

[PB13] Heiko Paulheim and Christian Bizer. Type inference on noisy RDF data. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 510–525, 2013. 120

[PBF+15] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. Data profiling with metanome. *PVLDB*, 8(12):1860–1863, 2015. 25, 93

[PBGS19] Matteo Paganelli, Domenico Beneventano, Francesco Guerra, and Paolo Sottovia. Parallelizing computations of full disjunctions. *Big Data Research*, 17:18–31, 2019. 14

[PEM+15] 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. *PVLDB*, 8(10):1082–1093, 2015. 25

[pr11] Gartner Group press release. Pattern-based strategy: Getting value from big data. http://www.gartner.com/it/page.jsp?id=1731916, July 2011. 1

[Pre16] Gil Press. Cleaning data: Most time-consuming, least enjoyable data science task. http://www.gartner.com/it/page.jsp?id=1731916, March 2016. 2

[PSGV19] Matteo Paganelli, Paolo Sottovia, Francesco Guerra, and Yannis Velegrakis. Tuner: Fine tuning of rule-based entity matchers. In *The 28th ACM International Conference on Information and Knowledge ManagementNovember 3–7, 2019Beijing, China*, 2019. 14

[PSM+19a] Matteo Paganelli, Paolo Sottovia, Antonio Maccioni, Matteo Interlandi, and Francesco Guerra. Explaining datasets with descriptions. In *Under Submission*, 2019. 5, 170

[PSM+19b] Matteo Paganelli, Paolo Sottovia, Antonio Maccioni, Matteo Interlandi, and Francesco Guerra. Understanding data in the blink of an eye. In *The 28th ACM International Conference on Information and Knowledge ManagementNovember 3–7, 2019Beijing, China*, 2019. 5, 170

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001. 30

[RLN13] Daniel Rinser, Dustin Lange, and Felix Naumann. Cross-lingual entity matching and infobox alignment in wikipedia. *Inf. Syst.*, 38(6):887–907, 2013. 31

[ROS15] Sudeepa Roy, Laurel Orr, and Dan Suciu. Explaining query answers with explanation-ready databases. *PVLDB*, 9(4):348–359, 2015. 4, 21

[RS14] Sudeepa Roy and Dan Suciu. A formal approach to finding explanations for database queries. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1579–1590, 2014. 21

[RWD+08] Senjuti Basu Roy, Haidong Wang, Gautam Das, Ullas Nambiar, and Mukesh K. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 13–22, 2008. 4, 24

[SA00] Russell C. Swan and James Allan. Automatic generation of overview timelines. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 24-28, 2000, Athens, Greece*, pages 49–56, 2000. 128

[SAMA17] Vinay Setty, Abhijit Anand, Arunav Mishra, and Avishek Anand. Modeling event importance for ranking daily news events. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM*

2017, Cambridge, United Kingdom, February 6-10, 2017, pages 231–240, 2017. 123

[SG16] Andreas Spitz and Michael Gertz. Terms over LOAD: leveraging named entities for cross-document extraction and summarization of events. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 503–512, 2016. 8, 28, 30, 133

[SGG12a] Jaroslaw Szlichta, Parke Godfrey, and Jarek Gryz. Chasing polarized order dependencies. In *Proceedings of the 6th Alberto Mendelzon International Workshop on Foundations of Data Management, Ouro Preto, Brazil, June 27-30, 2012*, pages 168–179, 2012. 25

[SGG12b] Jaroslaw Szlichta, Parke Godfrey, and Jarek Gryz. Fundamentals of order dependencies. *PVLDB*, 5(11):1220–1231, 2012. 6, 25, 70, 71, 73, 74, 75, 76, 78, 81, 82, 83, 84

[SGG+14] Jaroslaw Szlichta, Parke Godfrey, Jarek Gryz, Wenbin Ma, Weinan Qiu, and Calisto Zuzarte. Business-intelligence queries with order dependencies in DB2. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, pages 750–761, 2014. 26, 69

[SGG+17] Jaroslaw Szlichta, Parke Godfrey, Lukasz Golab, Mehdi Kargar, and Divesh Srivastava. Effective and complete discovery of order dependencies via set-based axiomatization. *PVLDB*, 10(7):721–732, 2017. ix, 7, 27, 88, 92, 97, 102, 179

[SGGZ13a] Jaroslaw Szlichta, Parke Godfrey, Jarek Gryz, and Calisto Zuzarte. Axiomatic system for order dependencies. In *Proceedings of the 7th Alberto Mendelzon International Workshop on Foundations of Data Management, Puebla/Cholula, Mexico, May 21-23, 2013.*, 2013. 25

[SGGZ13b] Jaroslaw Szlichta, Parke Godfrey, Jarek Gryz, and Calisto Zuzarte. Expressiveness and complexity of order dependencies. *PVLDB*, 6(14):1858–1869, 2013. 26

[SGNV19] Paolo Sottovia, Simon Gottshalk, Wolfgang Nejdl, and Yannis Velegrakis. Extracting historical and contemporary events from wikipedia by exploiting temporal entity co-references. In *To be submitted*, 2019. 9, 170

[SJY11] Anish Das Sarma, Alpa Jain, and Cong Yu. Dynamic relationship and event discovery. In *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pages 207–216, 2011. 8, 28

[SKAZ15] Stefan Siersdorfer, Philipp Kemkes, Hanno Ackermann, and Sergej Zerr. Who with whom and how?: Extracting large social networks using search engines. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 1491–1500, 2015. 8, 29

[SKW08] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A large ontology from wikipedia and wordnet. *J. Web Semant.*, 6(3):203–217, 2008. 30

[SM07] Lucia Specia and Enrico Motta. Integrating folksonomies with the semantic web. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, pages 624–639, 2007. 120

[Spa16] Apache Spark. Apache Spark: Lightning-fast Cluster Computing, 2016. 127

[SPGV19a] Paolo Sottovia, Matteo Paganelli, Francesco Guerra, and Yannis Velegrakis. Finding synonymous attributes in evolving wikipedia infoboxes. In *23nd European Conference on Advances in Databases and Information Systems, ADBIS 2019, Bled, Slovenia, September 8–11, 2019*, 2019. 12, 170

[SPGV19b] Paolo Sottovia, Matteo Paganelli, Francesco Guerra, and Maurizio Vincini. Big data integration of heterogeneous data sources: The re-search alps case study. In *2019 IEEE International Congress on Big Data (BigData Congress)July 8–13, 2019Milan, Italy*, 2019. 14

[SS94] Nambirajan Seshadri and Carl-Erik W. Sundberg. List viterbi decoding algorithms with applications. *IEEE Transactions on Communications*, 1994. 43, 48

[SSM96] David E. Simmen, Eugene J. Shekita, and Timothy Malkemus. Fundamental techniques for order optimization. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.*, pages 57–67, 1996. 15, 26, 69

[ST17] Rachele Sprugnoli and Sara Tonelli. One, no one and one hundred thousand events: Defining and processing events in an inter-disciplinary perspective. *Natural Language Engineering*, 23(4):485–506, 2017. 29

[SZK+16] Miroslav Shaltev, Jan-Hendrik Zab, Philipp Kemkes, Stefan Siersdorfer, and Sergej Zerr. Cobwebs from the past and present: Extracting large social networks using internet archive data. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 1093–1096, 2016. 30

[TA14] Giang Binh Tran and Mohammad Alrifai. Indexing and analyzing wikipedia's current events portal, the daily news summaries by the crowd. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 511–516, 2014. 29, 128, 133

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. 84

[VK14] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014. 133

[WDM15] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1231–1245, 2015. 20

[WGR01] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7, 2001, Proceedings*, pages 101–110, 2001. 92, 94

[WM08] Ian H Witten and David N Milne. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. 2008. 8, 29

[WM13] Eugene Wu and Samuel Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013. 4, 21, 35

[WNRR16] Wei Wang, Yue Ning, Huzefa Rangwala, and Naren Ramakrishnan. A multiple instance learning framework for identifying key sentences and detecting events. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 509–518, 2016. 28

[WZQ+10] Yafang Wang, Mingjie Zhu, Lizhen Qu, Marc Spaniol, and Gerhard Weikum. Timely YAGO: harvesting, querying, and visualizing temporal knowledge from wikipedia. In *EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings*, pages 697–700, 2010. 127, 134

[YBRW13] Mohamed Yahya, Klaus Berberich, Maya Ramanath, and Gerhard Weikum. On the spot: Question answering over temporally enhanced structured data. In *Proceedings of Workshop on Time-aware Information Access, TAIA2013. Dublin, Ireland*, 2013. 27

[YNM16] Dong Young Yoon, Ning Niu, and Barzan Mozafari. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1599–1614, 2016. 4, 22

[YPS09] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Summarizing relational databases. *PVLDB*, 2(1):634–645, 2009. 23

[YPS11] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Summary graphs for relational database schemas. *PVLDB*, 4(11):899–910, 2011. 23

[YZH+15] Ying Yan, Jiaxing Zhang, Bojun Huang, Xuzhan Sun, Jiaqi Mu, Zheng Zhang, and Thomas Moscibroda. Distributed outlier detection using compressive sensing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 3–16, 2015. 4

[ZDFB12] Jian Zhao, Steven M. Drucker, Danyel Fisher, and Donald Brinkman. Timeslice: interactive faceted browsing of timeline data. In *International Working Conference on Advanced Visual Interfaces, AVI 2012, Capri Island, Naples, Italy, May 22-25, 2012, Proceedings*, pages 433–436, 2012. 128

[ZSW15] Congle Zhang, Stephen Soderland, and Daniel S. Weld. Exploiting parallel news streams for unsupervised event extraction. *TACL*, 3:117–129, 2015. 28, 127, 128