COST–EFFECTIVE DATA WRANGLING IN DATA LAKES

A thesis submitted to The University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2020

By
Alex T. Bogatu
School of Computer Science

Contents

Word Count: 53610

List of Tables

List of Figures

Abstract

Data analytics stands to benefit from the increased availability of datasets that are held without their conceptual relationships being explicitly known. When collected, these datasets form a data lake from which, by processes like data preparation, also known as data wrangling, specific target datasets can be constructed that enable value–adding analytics. Given the potential vastness and heterogeneity of data lakes, obtaining value from such targets often requires significant prior effort in preparing the data for analysis. For example, data wrangling is reported to take as much as 80% of the time of data scientists. The issue then arises of how to decrease this cost.

This thesis investigates what makes data preparation costly and how data preparation can become more cost–effective through automation. Specifically, this thesis inquires into two challenges that have been insufficiently covered by the state–of–the–art, viz., how to automatically pull out of the data lake those datasets that might contribute to wrangling out a given target, and how to automatically homogenise the representation of their instance value. We refer to the former as the problem of dataset discovery and to the latter as the problem of format transformation. This thesis contributes effective and efficient solutions to both problems.

The work described in this thesis should be of interest to researchers and professionals in the areas of data analysis and data wrangling, who, in the process of preparing the data for analysis, confront themselves with heterogeneously represented data originating from many autonomous sources.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First and foremost, I would like to thank my supervisors, Prof. Norman W. Paton and Dr. Alvaro A. A. Fernandes, for their continuous support and invaluable advice. I would have never completed a PhD without them and I could not have asked for better supervisors.

I would like to thank Dr. Nikolaos Konstantinou for all the help he provided, and to my colleagues and friends in the VADA Project group for their companionship. My appreciation and recognition to my colleagues from the Information Management Group and from the School of Computer Science at the University of Manchester for providing a suitable environment to complete my research.

Finally, I would like to offer my sincere gratitude to my family for their invaluable support during my PhD studies, and to my future wife, Lacra, for her love and support, and for putting up with me while I finished this thing.

Chapter 1

Introduction

It is said that "information is power", but this is only partially true when it comes to information extracted from available data by organisations, an action aimed at improving their decision–making processes. The power is not conferred by the simple possession of data alone, but by the ability to manage and analyse it as well. Market research analysts predict the global Big Data software market to be worth $92 billions by 2026 [Wik]. This growth is driven by a data "deluge" that promises virtually inexhaustible resources for analysis that can inform business decision making processes or unveil new commercial opportunities. But the volume of available data tends to outpace the capabilities of traditional data management technologies: it has become increasingly expensive for businesses to manage the available data pools, with many of the tasks required to prepare the data for analysis being done manually, through *ad–hoc* scripts [Data]. However appealing the promises of data analysis and data–centric business intelligence, in practice, the data fuelling these opportunities is challenging to interpret and to curate before it can be useful to analytical algorithms [Datc].

The focus of this thesis is, therefore, on *cost-effective data preparation*. The general aim is to explore the extent to which the cost of preparing data for analysis, a process commonly known as *data wrangling* [RHH+17], can be reduced through *automation*, while preserving the opportunities that stem from increased availability of data. At the same time, we acknowledge that automatic solutions are unlikely to be able to match the reach or quality of outcomes obtainable by data scientists, but any level of automation implies the possibility of added value for minimal cost.

This chapter continues with the scene setting and discusses why data preparation is important. Then, it presents the motivation for automatic data preparation and the context in which it becomes necessary. Lastly, the objectives and research contributions are described, and an overview is offered on what is to follow in the remaining chapters.

1.1 The importance of data preparation

The need for, and interest in data preparation have become widespread with the advent of data warehouses, a notion first proposed by William Inmon in the 1970s. Broadly speaking, the approach taken to prepare the data with which to build the data warehouse was largely governed by three main processes: Extract, Transform, and Load (ETL) [Inm92]. The extraction process obtains raw data from different data sources. The transformation process performs data cleaning and integration in order to merge the data from multiple sources. The loading process moves the clean and merged data into the data warehouse where it is organised in terms of dimensions and facts [Inm92]. Users access the warehoused data primarily to perform analytical tasks on it. Therefore, preparing data for analysis largely meant cleaning and structuring the data according to the rules embodied in the ETL processes used to populate the warehouse. It is clear, then, that data preparation was (and still is) a vital step, one that ensures the availability of clean and organised data for answering analytical questions.

In 2010, James Dixon, came up with the notion of a *data lake* [Dix], arguing that the traditional warehouse is limited due to size restrictions and narrow search parameters, hindering the full potential for analysis by imposing a predefined, fixed schema on the stored datasets. This led to the proposal of data lakes as a new and more liberal type of data repository, one that stores datasets in their original raw format, postponing any concerns with structure or with existing inconsistencies until analysis time. Such a repository promises unrestricted and malleable data analysis by being open to future opportunities that might not be known at population time. Storing unprocessed data, however, implies that data preparation will have to be performed at analysis time, and, potentially, in ways that are specific to the intended analysis. Therefore, the need for data preparation is as present as ever in the case of data lakes, and comes with new challenges, some of which motivate this work, as we now describe.

1.2 Motivation: data preparation challenges and opportunities

Much analysis has been conducted to better understand the needs and opportunities of data management for analytics. This has quickly revealed that taming the data and extracting its secrets is no free lunch. For example, a 2016 report on data science [Data] revealed that data scientists spend up to 80% of their time preparing the data for analysis and only 20% of their time analysing it. Two years later, a similar report outlined similar figures, with 55% of the respondents citing that the quantity and quality of the training data was their biggest challenge that takes up most of their time [Datc]. Such results have become the *raison d'etre* for many research efforts, with many papers on data preparation citing the 80/20 rule of working with data, e.g., [KHP+11], [KPHH11], [DFA+17], [KKA+17].

There is a clear need for lowering the data preparation time but this is made more difficult by the ever increasing availability of potentially useful, heterogeneous data. Consider, for instance, the task of estimating customer satisfaction with a particular (or a range of) products. A company interested in such insights can have multiple sources of evidence at its disposal: social media posts, e.g., tweets about the product, feedback (in the form of comments) on the product from specialised reviewing platforms, internal feedback obtained from clients, etc.. Each source of evidence can provide data in different formats ranging from unstructured text to relational data, obeying different formatting rules. Furthermore, not all available data may be relevant for the product(s) in question or contain the targeted characteristics. These are only few of the challenges that have to be overcome when working with large volumes of heterogeneous data such as are the focus in this dissertation. Also, the studies cited above estimate that 71% of the interviewed data scientists work primarily with structured data [Datc], so the techniques proposed in this dissertation focus on relational data, e.g., relational database tables, CSV files, etc. Other challenges that deal with data governance, security, or contextual metadata, such as studied in [BBC+15] or [HSG+17], are acknowledged as contributing to the risk of transforming the data lake into a data swamp and making the process of data preparation expensive, but are not addressed in this thesis.

Having touched on the challenges of data preparation, this thesis argues that

data is not only valuable for informing business decisions but also for understanding its real world domain, its structure, and the relationships that hold in it, all of which can inform data preparation tasks. In other words, given a repository of data and an analytical goal, there is an opportunity for harnessing such data characteristics to answer questions such as *What data should be used for the current requirements?* and *What operations are needed to clean the input data?* Ultimately, the answers to such questions can inform the specification of which data preparation steps should be performed, thereby opening the way for automating those steps and decreasing the preparation cost. Specifically, in this thesis, we address questions related to (i) automated dataset discovery, which aims to enable a targeted run of preprocessing tasks onto a relevant subset of sources, with a view to decreasing the overall cost of preparing the data, and (ii) automated normalisation of textual value representation, which aims to offer cleaner, more homogeneous data, as required by most analysis tools and essential for effectively performing other preparation steps that extract evidence from textual values. We argue that our ethos of automated dataset discovery and format normalisation decreases the cost of data preparation (i) by reducing the size of the input through the discovery of the relevant datasets for the task at hand, and (ii) by relieving the user from having to write transformation scripts that normalise the format representation of the values. These cost–reduction opportunities remain mostly unexploited by the research community, and, when they have been exploited, the contributed approaches either do not scale well when applied to large and diverse repositories, or require expert–level manual user intervention. More details on how the state–of–the–art has exploited these opportunities, and on how our work relates to existing approaches, are given in the sections that now follow.

1.3 Data preparation through wrangling

The previous section laid out the motivation behind our vision, viz., to enable cost–effective data preparation, in the absence of expert–level user knowledge, through the automation of data discovery (which selects the input to a given analytical task), and through the automation of format transformation (which normalises that input). Before presenting how our work contributes to such a vision, and to have a more complete picture as to why such a vision would be useful, we now briefly introduce the application contexts that make rapid and

cost–effective data preparation imperative, i.e., data lakes, as well as the tasks that are commonly performed to prepare the data, i.e., data wrangling.

1.3.1 Data lakes

The ever increasing availability of data with potential for analysis has produced a shift from the purposely–designed enterprise data warehouse to a more liberal, unconstrained dataset repository where information can be stored in its original representation, without concern for its structure or cleanliness until analysis time. When asked what is the drive for such a change, businesses often invoke the opportunities for fast development cycles, exploration and innovation with significant gains to their competitive edge [HKN+16]. These opportunities become tangible when data engineers and scientists can generate and use data without the constraints imposed by the predetermined and largely immutable structure that is characteristic of traditional data warehouses. The downside of data lakes is that organisations have to overcome the common challenges that stem from the four *V*'s of Big Data [FGL+16]: *Volume* relating to the scale (the number of datasets and their sizes) of repositories; *Variety* relating to the diversity of data sources, including publicly available data, internally generated datasets, web extracted data, etc.; *Velocity* relating to the rate at which new data is ingested or existing data is changed; and *Veracity* relating to the unavoidable uncertainty inherent in such repositories. Research on data lake management by leaders in the market, e.g., [HKN+16], [TSRC15], suggests that organisations tend to employ different, often *ad–hoc*, and very specific tools and techniques to manage their data lakes. Such tools often require highly specialised user skills and the data preparation process is often governed by internally developed best practices and norms.

This thesis does not address organisation–specific data lake management rules and, therefore, we employ a broad definition of what a data lake is, and we argue that the proposed contributions are independent from organisation–specific characteristics and relevant for a broad range of data lake types. In other words, this thesis does not make any assumptions regarding the size of the repository, the domain of the datasets (i.e., we do not use domain–specific knowledge), the existence of metadata for the datasets, or, crucially, regarding interrelationships and similarities between datasets. We do assume that the data lake consists of tabular datasets, such as relational database tables or CSV files.

Definition 1.1. *A data lake is a centralised repository that allows the storage of datasets in various formats, with potential for future analysis, at any scale, from any domain, possibly without any explainable metadata beyond attribute names and their domain–independent attribute types (i.e., string, integer, etc.).*

There are a number of characteristics of a data lake stemming from Definition 1.1, some of which have been already introduced in Section 1.2, that define the need for cost reduction through automation in data preparation:

- A data lake can contain a large number of datasets from autonomous sources. This means that each new analytical task could require a different subset of datasets to be prepared and analysed. The challenge here is to identify relevant datasets, lest one is forced to do data preparation over the entire data lake.

- The datasets are stored with their original schema and each new analytical task might require only a subset/combination of the attributes of each dataset. The challenge here is to identify those attributes and how to, potentially, integrate them so that analysis can be conducted on a unified set of values. Note that, here, we do not include cases requiring the entire representation of the data to be transformed, e.g., semi–structured data transformed to a tabular representation, since we assume all datasets to be in tabular format.

- The datasets can exhibit inconsistent levels of available metadata, and even no metadata at all. The challenge here is to identify other sources of evidence that would allow the correlation of similar datasets/attributes relevant for the given analytical task and to reliably extract that evidence.

- The attribute values in each dataset can be represented in different formats. This means that any task, be it part of analysis or preparation, that relies on evidence extracted from such values would be impacted by inconsistent representations. The challenge here is to normalise the representation of values for the datasets that are to be analysed.

Note that the above list of challenges is not exhaustive, but rather representative with respect to Definition 1.1. With respect to the 4 *V*'s of Big Data mentioned before, the above challenges are mostly characteristic of *Volume* and *Variety*, and this is our focus in this thesis.

The orchestration of preparatory tasks that deal with challenges, such mentioned above, is known as *data wrangling* and the next section briefly introduces the notion.

1.3.2 Data wrangling

We have pointed, in Section 1.1, to the importance of data preparation for analysis over both data warehouses and data lakes. Just as data preparation in data warehouses is carried out by ETL processes, data preparation in data lakes uses data wrangling techniques, i.e., a collection of methods that can enable the extraction, preparation and integration of large volumes of data originating from diverse autonomous sources, for the use of different types of users, in a variety of use cases that often exceed the capabilities of state–of–the–art ETL tools [HHK18], [RHH+17].

Current data wrangling solutions aim to enable data preparation for users without deep software engineering skills, by orchestrating the execution of tasks ranging from data discovery, through data cleaning and transformation, to data integration [KHP+11], [FGL+16]. In this thesis, we follow the same path laid out by previous research on data wrangling and define the process as follows.

Definition 1.2. *Data wrangling consists of orchestrating a collection of interconnected, often user–guided, processes that select, transform, clean, match and merge the data according to some target specification, with the aim of creating a data product suitable for downstream analysis.*

Definition 1.2 introduces important characteristics of data wrangling: it is a multi–step process (e.g., [FGL+16], [KHP+11]), involving specialised techniques, most of them complex enough to sustain their own research areas, e.g., schema matching [RB01], data cleaning [ACD+16], etc. In line with Definition 1.2, in this thesis, we consider data wrangling as consisting of the following steps:

- **Data discovery**, which identifies the relevant datasets for the task in hand, which are then provided as input for remaining wrangling steps.
- **Data profiling**, which identifies structural relationships, e.g., inclusion of instance values, between the attributes of the relevant datasets.

- **Schema matching**, which postulates matching relationships between attributes of the relevant datasets.
- **Schema mapping**, which constructs an executable specification of how to combine/merge relevant datasets to generate a unified view of the data for analysis.
- **Format transformation**, which carries out changes to the representation of some of the attributes, with a view to reducing some of the existing inconsistencies.
- **Data repair**, which fixes and amends some of the empty/incorrect values that are present in the relevant datasets.

In this thesis, of the above wrangling sub–processes, we contribute automated solutions for data discovery, thus addressing the *Volume* dimension of the Big Data challenges, and for format transformation, thus (partly) addressing the *Variety* dimension of the Big Data challenges.

Thus, the two topics at the core of this thesis are data discovery and automatic format transformation. Both topics have been addressed in the literature before, albeit with different aims, as follows:

- Data discovery has been studied in terms of searching for related datasets from a given repository, e.g., [DSFG+12], [NZPM18], [FAK+18]. The proposed solutions assume high levels of consistency between the representations of similar instance values of datasets and make extensive use of Web knowledge–bases or external sources of information about the tables in the repository, neither of which are always available. Furthermore, techniques for assimilating external knowledge may involve overheads that negatively impact the performance of the discovery process, as shown in Chapter 3. Our contributions to the data discovery problem aim at closing these gaps by identifying relatedness features that can be mapped to a uniform distance space, are lenient with respect to heterogeneous representation of similar instance values, and do not rely on external ontologies.
- Format transformation has been studied when data is transformed based on user–provided examples [Gul11] and when the user manually writes the transforming operations based on suggestions from the system [KHP+11]. The former only

requires input–output examples to synthesise a transformation. Therefore, the user has to know what transformation is needed to be able to provide useful examples. The latter expects manual authoring of transformation scripts. Therefore, the user has to be familiar with the transformation needed and know how to express it using a domain specific language. Both approaches assume a small number of input sources of manageable size and limited levels of format heterogeneity, identifiable by the user. Our contributions, presented in Chapters 4 and 5, describe a scalable and automated solution to the problem of format transformation that requires reduced knowledge about the data and the required transformation, and reduced or no knowledge about how to express the latter.

These two topics are further discussed in Chapters 3, 4, and 5, where our contributions are described in full detail and contrasted with the relevant state–of–the–art for each topic.

1.4 Aim, objectives and research contributions

The overall hypothesis underlying this thesis is that the overall cost of performing data wrangling in data lakes can be reduced through automation. We consider the cost of wrangling to be dominated by the need for deploying human expertise and by the associated time–cost incurred in identifying, cleaning and integrating the data prior to analysis. Therefore, the aim of the research presented in this dissertation is:

> *To reduce the cost of data preparation by developing techniques that automate the identification of relevant data for a given wrangling task and the operations necessary to clean and transform the format representation of data values.*

In order to fulfil this aim, we define the following objectives:

- O_1: Focusing on data discovery, to identify the types of evidence that enable the identification those datasets in a data lake that are most relevant for the given task, and to develop techniques that use that evidence to perform cost–effective dataset discovery.

Research contributions:

(i) A distance–based framework that combines five types of evidence in order to measure the relatedness of two datasets.

(ii) An automated solution that, given a large repository of data and a target schema with representative instances, uses the techniques from (i) to identify what datasets should be input to downstream wrangling tasks.

Impact: This would decrease the cost (in terms of time and of user expertise) of running a significant number of data wrangling tasks on data lakes by focusing the process on datasets that are relevant for the task at hand.

Evaluation: Success is measured by the effectiveness and the efficiency achieved when applying the data discovery techniques on data lakes. We measure both aspects and comparatively analyse our proposal against the closest antagonists, viz., [NZPM18] and [FAK+18], using a real–world and a synthetic data lake.

- O_2: Focusing on data format transformation, to develop techniques that learn the operations necessary for modifying the representation of strings, efficiently and with minimal user intervention, with a view to normalising the instance values of a given attribute.

  Research contributions:

  (i) A transformation language that is complex enough to express the common string processing operations that are required to modify the format representation of data values, while being simple enough to be amenable to automatic synthesis from given input–output examples.

  (ii) An automated solution to format transformation generation, i.e., a synthesis algorithm, that, given a set of input–output examples, generates an executable transformation that maps each new instance consistent with the input examples to its corresponding value consistent with the output examples, and that scales to large sets of examples.

  Impact: Such a solution would offer the opportunity for performing format normalisation of data values in an efficient and automatic manner, without requiring advanced familiarity with the data values or programming knowledge.

  Evaluation: Success is measured by the effectiveness and the efficiency achieved when performing format transformation on different real–world scenarios. We

measure both aspects and comparatively analyse our proposal against the closest antagonist from the sate–of–the–art, viz., [Gul11].

- O_3: Focusing on format transformation, to develop techniques that automatically identify the input specifications needed by a format transformation synthesis algorithm, when it is applied on multiple datasets and active user supervision is impractical.

  Research contributions:

  (i) An automated solution to the production of example data, i.e., input specifications, from which synthesis algorithms can learn how to transform the textual values without user input.

  (ii) An automated solution to the pruning of the potentially very large space of input–output example candidates produced by (i).

  Impact: Such a solution has the potential for decreasing the cost (in terms of time and user expertise) of normalising the format representation of values at data lake scale by (i) automatically identifying situations in which transformations are required; and by (ii) automatically and reliably selecting the most relevant examples that can be used to inform the synthesis of transformation programs.

  Evaluation: Success is measured by the effectiveness and the efficiency of format transformations, synthesised from automatically generated examples, applied on real–world datasets of different sizes.

- O_4: Focusing on the combined impact of the techniques stemming from O_1, O_2 and O_3, to complement existing methods for performing downstream wrangling tasks, such as data profiling, with preprocessing steps for normalising the data.

  Research contributions: a preprocessing step for data profiling tasks that normalises data with heterogeneously represented values ahead of running data profiling specific algorithms.

  Impact: This would enable the successful use of state–of–the–art data profiling algorithms, specifically inclusion dependency discovery algorithms, on datasets that do not exhibit the characteristics expected by such attributes, e.g., homogeneous format representation of instance values.

Evaluation: Success is measured by the effectiveness gain achieved when performing data profiling on normalised data, as opposed to performing data profiling on potentially heterogeneous data.

1.5 Overview of thesis structure

The remainder of this dissertation is structured as follows:

- Chapter 2 presents the technical background of the thesis. We pick up the discussion introduced in Section 1.3 and extend it by examining the major Big Data repository types and presenting the vision and motivation for data wrangling. From this discussion, the need for cost–effectiveness emerges, demanded primarily by the challenges of applying data wrangling on data from a data lake. Two of these challenges, viz., the scalability challenge resulting from the potential size of the input, and the heterogeneity challenge resulting from the potential diversity of datasets, are analysed in detail. Before concluding the chapter, we explore the existing systems for end–to–end data wrangling and investigate how the scalability and heterogeneity challenges are addressed in each of them.

- Chapter 3 describes our contributions to the scalability challenge, viz., automatic dataset discovery. We define the term and discuss the notion of data relatedness and how it plays a role in dataset discovery, i.e., what it means for two datasets to be related or similar, and how our proposal (called D^3L for *dataset discovery in data lakes*) positions itself with respect to the state–of–the–art in dataset discovery. The contributions are then introduced and described in detail, before we empirically evaluate our method and compare it against the closest competitors. We end the chapter by revisiting some of the objectives introduced in this chapter and by indicating whether and how they are fulfilled by D^3L.

- Chapter 4 introduces our contributions to the heterogeneity challenge, viz., automatic format normalisation. The chapter sets the scene by providing a definition of the term and a discussion of the methodology for performing format normalisation used in the literature. We touch on how to express the transformations that normalise the format representations, and on how to learn such transformations automatically from input–output examples. We

describe in detail the contributions of the chapter and empirically evaluate the effectiveness and efficiency of our proposed synthesis algorithm (called *SynthEdit*) against the closest antagonist. The chapter concludes by revisiting the relevant research objectives introduced in this chapter and by indicating whether and how *SynthEdit* fulfils them.

- Chapter 5 continues the discussion on format normalisation by describing how transformation synthesis can be applied to scenarios with many attributes to transform and where user provision of examples is impractical. The chapter defines the main goals and presents the background and theoretical notions upon which the relevant contributions are built. We review the main proposals for generating input–output examples for synthesis algorithms, and describe how such techniques can be combined with *SynthEdit* to perform format normalisation on multiple datasets without active user intervention. Finally, we empirically evaluate the effectiveness and the efficiency of our proposal by automatically normalising data from various domains. The conclusion of the chapter uses evidence from the evaluation to discuss whether and how relevant research objectives mentioned above have been fulfilled.

- Chapter 6 proposes a use case for combining data relatedness discovery and automatic format transformation, with the goal of enabling the use of specific data profiling algorithms that assume certain characteristics of the input data, when such assumptions do not hold. The chapter proposes a preprocessing step to inclusion dependency discovery and details how data structures specific to dataset discovery and format normalisation of values can be combined to process the data ahead of profiling. Finally, the benefits of such pre–cleaning of the data values are evaluated against data with various characteristics from various domains. The conclusion of the chapter discusses whether and how the fulfilment of relevant research objectives from above has been met.

- Chapter 7 summarises the reported contributions and discusses their significance. We briefly revisit the state–of–the–art for both dataset discovery and format transformation, with a focus on gaps that our contributions fill. The thesis ends with a discussion on how the work done on the two wrangling tasks addressed can be taken further, i.e., what we see as the main lines of research for future work.

Chapter 2

Background: data preparation for Big Data

In the previous chapter we introduced the notions of data lakes and data wrangling. We now provide a detailed discussion of these areas that represent the research background upon which we build our contributions in subsequent chapters.

We have defined data wrangling (Definition 1.2) as a process of iterative data exploration, transformation and integration that provides an end–product that is suitable for analysis. Two of the data preparation challenges that were cited as motivating factors for data wrangling in Chapter 1 are the time–consuming nature of data preparation and the need for specialised technical knowledge. To draw a more comprehensive picture of the context in which these challenges arise, and of how state–of–the–art research has approached their treatment, we start from data warehouses and ETL processes, and then explore the circumstances in which the shift to the more liberal data lakes has taken place. To this end, we begin by revisiting the notion of data warehouses in Section 2.1. We explore the principles upon which such repositories are built and then focus on the operations on the data that take place in its path from source to analysis, i.e., the ETL process. This highlights some challenges faced in ETL tasks that demand a different approach to storage and preparation of data that give rise to the more recent notions of data lakes and data wrangling, respectively. In Section 2.2 we introduce data lakes as a type of repository embodying a different approach to the storage of data for later analysis. We provide examples of what a data lake is and explore how data wrangling is used to prepare the data for analysis. We then discuss the need for automating data wrangling tasks in Section 2.3. We finish the chapter

with an overview of representative data wrangling systems in Section 2.4 and conclude with a summary in Section 2.5. Note that detailed, technical discussion and contrast of our contributions with relevant state–of–the–art approaches is given in later chapters, at the point in which the said research contributions have been minutely described and evaluated.

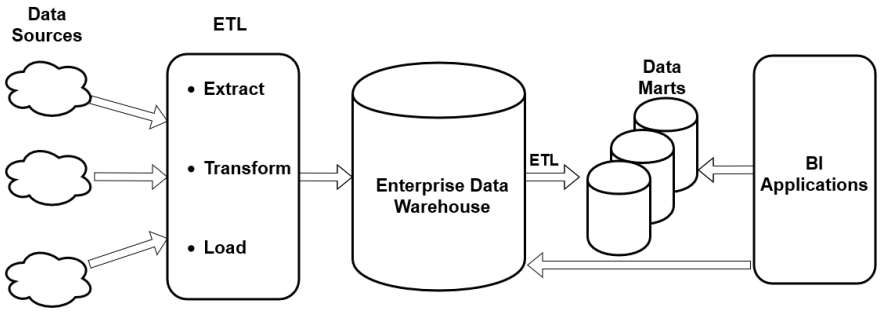
2.1 The data warehouse

Before delving into the details of data lakes and data wrangling, we first explore the notion of a data warehouse and of an ETL process, i.e., the longer–standing technology for enterprise analytics [KC04], so as to explain how the need for data lakes emerged, how ETL is different from data wrangling and why ETL does not always fulfil current data preparation requirements.

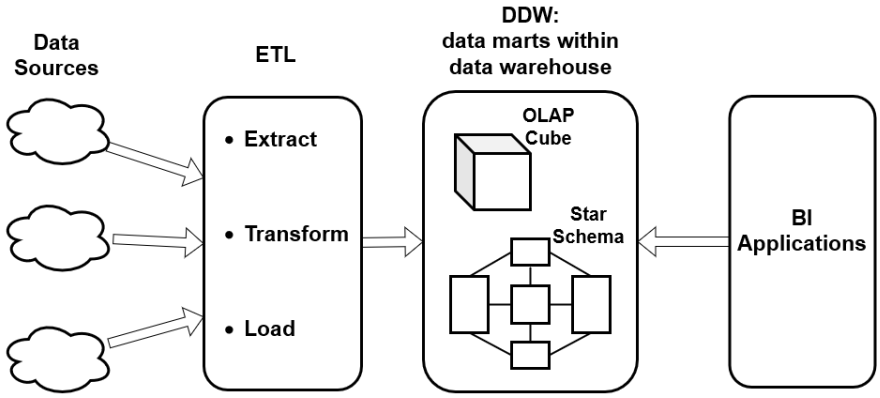
2.1.1 Building the data warehouse

Research and development of data warehouses [Inm92] has been conducted on two fronts. The first front is concerned with the design principles for building and using a data warehouse. Two proposals are prominent in this area: the Corporate Information Factory (CIF) [Inm92] and the Dimensional Data Warehouse (DDW) [KR02]. Figure 2.1 illustrates the two approaches to data warehouse design. The CIF, also known as the Enterprise Data Warehouse (EDW), is "a subject–oriented, integrated, non–volatile, time–variant collection of data for managerial decision making" that stores data in relational format [Inm92]. It is created by ETL processes that extract the data from sources, transform it according to a set of predefined rules, and load it into the repository. Once created, the EDW can be accessed by applications whose requirements are satisfied by the predefined EDW schema. Sub–repositories, known as *data marts*, can be built to accommodate analytic needs that require a different organisation for the data. In creating additional data marts, more ETL processing is normally required.

The creation of a DDW also relies on ETL processes but, in contrast to a CIF, data is mapped to a *dimensional data model* [KR02]. The fundamental concept in dimensional modelling is the star schema, which typically consists of a fact table, containing all the measures pertaining to a given data point that are relevant to the subject area. A fact table is linked to many dimension tables which provide different viewpoints on the facts, such as over time, over location,



(a) A simplified illustration of the Corporate Information Factory (a.k.a. Enterprise Data Warehouse) model, as popularised by Bill Inmon [Inm92].



(b) A simplified illustration of the Dimensional Data Warehouse model, as popularised by Ralph Kimball [KR02].

Figure 2.1: Simplified illustrations of the two main data warehouse implementation methodologies.

etc.. Multiple star schemas are typically built to satisfy different analytical and reporting requirements. This allows for the creation of distinct data marts for the same data warehouse.

Additional design models for the data warehouse have been proposed, e.g., [DM88], and comparative studies between different approaches have been reported in the literature, e.g., [Inm], [CG18], [JT12], [VS99]. Covering this research area in more detail than this brief introduction is beyond the scope of this thesis.

2.1.2 The ETL process

The second front for research and development in data warehousing is concerned with Extract, Transform, Load (ETL) processes [Vas09]. Figure 2.1 shows that

ETL processes are a fundamental step in creating a data warehouse. They are responsible for extracting the data from its native sources, transforming it according to the data warehouse schema, and loading it into the repository itself.

The extraction component interfaces with various data sources and identifies the data to be extracted. One challenge in this process is to build a logical data map that documents the relationship between the original source entities and the final destination entities expected in the data warehouse, be it an enterprise warehouse or a relational star schema [KC04]. Another challenge is extracting data from autonomously designed sources. In particular, the problem of extracting data from the Web has been the focus of numerous research endeavours that find their application in ETL, e.g., [Kus97], [CMM01], [LRNdST02], [AGM03], [FGG+14].

The transformation component is typically responsible for contending with the challenges relating to data quality, such as duplicate records, data inconsistencies or erroneous data, e.g., [KCH+03], [RD00], [ORRHG05], [BG05]. For example, AJAX [GFSS00] proposes a declarative language for performing different types of data transformations. Potter's Wheel [RH01] enables interactive data cleaning with the goal of reducing format inconsistencies. HumMer [BBN+05] is a system for dealing with data fusion problems [BN08].

The loading task typically takes place in bulk, either once–and–for–all or incrementally [Vas09]. Most research has been focused on the latter, e.g., on index maintenance using dwarfs [SDRK02] or tree–like structures [RKR97], [FKM+00], and on materialised view maintenance [Kot02], with [Rou98] being a useful summary still.

One widely–shared characteristic of the above proposals is that ETL procedures are designed as workflows, which are intended for use by specialised users and are governed by highly structured policies. As more and more sub–divisions of an organisation require access to the data warehouse, these ETL processes have to fulfil a variety of needs and the policies that govern the process have to be aligned with these needs. Over time, the organisational entities that build and maintain the data warehouse, and the beneficiaries of the information extracted from the data can diverge in terms of requirements, making disruptive redesign all but inevitable. In the next section, we briefly explore the reasons why this gap emerges.

2.1.3 Schema–on–write

In data warehouses data items are extracted from sources and transformed to conform to a schema at the time of storage. This schema is the predefined corporate data model in the case of CIF [Inm92], and the star schema in the case of DDW [KR02]. This approach is known as *schema–on–write* and is the main source of limitation leading to the proposal of data lakes. The main restrictions of the schema–on–write paradigm are:

- Although it does a good job of enforcing consistency rules on the data warehouse content, it is not flexible enough to accommodate the concomitant need for semi–structured or unstructured datasets, e.g., XML data, log files, social media generated data, etc., that are hard to coerce to the data warehouse schema.

- The road to data analysis is *structure* → *ingest* → *analyse*. Therefore, there needs to be a clear a priori analytical objective for which data is being stored and ingested (inclusive of cleaning, repair and preparation). As a result, many forms of exploratory data analysis are not supported because the data that is stored has a predefined scope. For example, e–commerce companies may mine website interaction logs for fraud and compliance. But the same data may prove valuable for analysing response times and user experience. In this case, different representations of the data are needed, however, it may be that not all use cases are known at the time the data is loaded into the data warehouse [Sha19].

- The structure of the data dictates the types of analysis that can be performed, and modifying the schema of the data to accommodate a new type of analysis may be hard. In such cases, a distinct ETL process may have to be run. Furthermore, the old representation of the data may still be of interest and, therefore, this can lead to high levels of redundancy and storage allocation.

For scenarios that require data with different structures to be widely accessible to different needs in different parts of an organisation, a new type of repository, and a new approach to preparing the data for analysis are required. The next section explores the notion of data lakes, in relation to the former, and of data wrangling, in relation to the latter.

2.2 The data lake

In this section, we discuss data lakes, as the notion is construed in this dissertation. We also describe the process of preparing data in a data lake through wrangling, which is one approach to minimising the shortcomings of the schema–on–write paradigm.

2.2.1 Building the data lake

The term data lake was defined in Definition 1.1, as a repository that stores the data in its original ("raw") format. The core idea of the data lake vision is that data with potential for analysis should be stored as–is until its value needs to be unlocked, instead of being coerced into data mart–like repositories that imply imposing a predefined structure on the data and, therefore, restricting the range of possible analyses that can be performed without significant restructuring [QH19], [Dix].

There is still no consensus among the key commercial players or leading research groups on a formal definition for data lakes, or on a standardised methodology for building a data lake [QH19]. Creating such a data repository tends to be a complex task that involves integration of numerous technologies for data ingestion and transformation. Research on the subject comes mainly in the form of data lake management systems, such as the ones described in [HGQ16], [HKN+16], [HSG+17], [TSRC15], or [WA15]. All these proposals aim to encompass more than relational data and to record the lifecycle of datasets through lineage information. To this end, they propose extensible metadata models and parsing frameworks for different data types (e.g., [QHV16]) that tend to be tailored to the challenges faced by the organisation that builds and uses the data lake. For example, Goods [HKN+16], is highly oriented towards rapidly changing data sets. Others, such as Personal Data Lake [WA15], propose metadata management systems that promise to give control to the user over managing their data and sharing it beyond organisational boundaries. As already mentioned, in this thesis we focus on tabular data and assume a more general notion of a data lake: one that does not include organisation–specific policies. Our contributions focus on the automation of the data wrangling process and, therefore, data lake management systems can be considered orthogonal to our work.

Figure 2.2 is a simplified illustration of the processes involved in building a

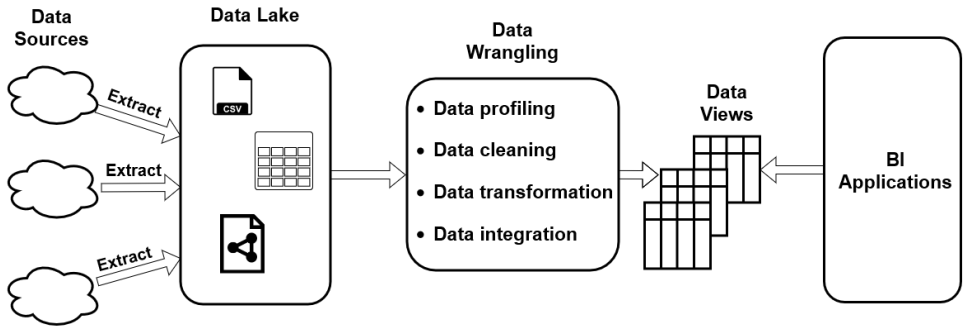


Figure 2.2: Simplified illustration of the approach to data analysis in data lakes with examples of possible data wrangling steps necessary for preparing the data.

data lake and preparing the data for analysis. Data can be extracted from its original sources using techniques similar to those used with ETL, e.g., [FGG+14], [AGM03], [LRNdST02]. The data lake can consist of such extracted data, or of data internally generated (such as logs or customer interaction statistics), including tabular data (e.g., CSV files), as well as semi–structured, or unstructured data. Data lake management systems, such as the ones mentioned above, tend to layer over this basic functionality by adding metadata, security, and governance–oriented capabilities.

Once the data is stored in the data lake, applications can access it through views that can be pregenerated, if the analytical/reporting task is known in advance, or generated on–demand, when the requirements for the input to the analysis tool are specified. In both cases, the views are generated by an orchestration of techniques including, but not limited to, profiling, cleaning, and integration, which are collectively referred to as data wrangling steps. Such views are usually specified as tables containing a unified, integrated version of all data relevant for the analysis task, since most analysis tools can consume inputs in tabular form.

Before delving into data wrangling tasks, and in order to have a clearer view as to what a data lake is and what challenges arise when managing and preparing datasets for analysis, consider repositories of publicly available open government data such as the UK data portal: `https://data.gov.uk/`, the US data portal: `https://www.data.gov/`, the New York data portal: `https://opendata.cityofnewyork.us/`, the UK historic data portal: `http://www.nationalarchives.gov.uk/`, etc. Figure 2.3 depicts some histograms extracted from tabular data in the UK data portal. Construed as a data lake, it contains approx. $25,000$

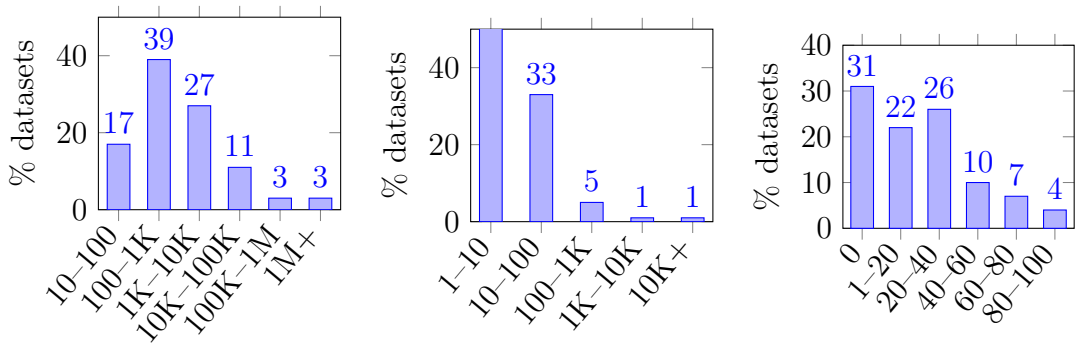
datasets with a total of approx. $1,000,000$ attributes and over 12 discernible domains, as shown in Table 2.1. Most datasets contain up to $10,000$ records and between 1 and 100 columns, while almost half have 20% or more numerical columns. Furthermore, columns often contain duplicate or missing values. Such statistics are evidence of the complexity faced by data scientists in selecting the relevant data for the task at hand, cleaning the selected data, and merging similar datasets. The orchestration of preparatory tasks that achieve these goals is known as *data wrangling*, which we discuss in more detail next.

Note that a data lake is not an alternative to a data warehouse: data lakes more likely complement data warehouses. For instance, there are reports from industry that advocate complementing data warehouses with data lakes, e.g., [DWa].

2.2.2 The data wrangling process

The path followed by data from sources to analysis tools, as illustrated in Figure 2.2, does not align well with the design principles of ETL procedures: ETL tasks would have to generate various types of views, on demand, from data with no fixed structure. This suggests that there is a need for a different type of data preparation. Data wrangling is the orchestration of profiling, cleaning, transformation, and integration tasks over datasets to ensure they are suitable for a given analysis task [HHK19]. Data wrangling tasks underpin tools accessible not only to IT professionals, as was the case with ETL, but to a broader range of users, such as data and business analysts, as well as managers. Research on the subject has gained traction with the realisation that, if data is not warehoused with a predefined analysis objective in mind, but stored in a data lake repository in anticipation of unplanned–for analysis, data preparation and cleaning can take up the majority of the time (up to 80%) of data scientists [DJ03], [KPHH12]. This high cost is determined by the limitations of preparation tasks, most of which are derived from ETL processes, in coping with the volume, variety, veracity, and velocity of the data. Specifically, research on data wrangling (e.g., [KHP+11], [KPHH12]) has identified the following sources of difficulties in data preparation:

Semi–structured data: Writing complex regular expressions to extract data into relevant fields proves to be a hard task. This prompted research into data extraction



(a) Cardinality of the constituent tables.

(b) Arity of the constituent tables.

(c) % of numeric attributes in the constituent tables.

Figure 2.3: Different statistics measured on the constituent tables of a real–world, open–government data lake: `data.gov.uk`

Domain	**Num. datasets**
Business & Economy	313
Crime & Justice	134
Defence	87
Education	716
Environment	1,963
Government	4,743
Government Spending	12,478
Health	1,578
Maps	465
Society	745
Towns & Cities	644
Transport	452

Table 2.1: `data.gov.uk` domains

from sources other than Web sites, e.g., [BGHZ15], [LG14], [RG17], [BLMT16].

Data cleaning: Data quality problems that were already driving research on data transformation in the context of ETL [KCH+03], [RD00], [ORRHG05], [BG05] are rife in data wrangling as well. According to [ACD+16], research in this area falls into one or more of the following four categories:

- Rule–based algorithms for detection and repair of integrity constraint violations, e.g., functional dependencies, inclusion dependencies, etc. [AAO+15], [CIP13], [FLM+12], [KIJ+15], [FG12]: These algorithms are embedded into data cleaning systems such as [DEE+13] and [RCIR17]. These proposals

are built around user input since there is no easy way to reliably automate the detection and repair of constraint violations.

- Pattern enforcement and transformation tools [KHP+11], [CMI+15], [AMI+16], [JACJ17], [HCG+18], [Gul11], [Sin16]: These tools discover format inconsistencies in data values starting from user–provided input–output examples or from other user interventions. In Chapter 4, we discuss this work in more detail since our contributions regarding format transformation belong to this category.

- Quantitative error–detection algorithms [PSC+15], [WT14], [AGN15]: The main tasks addressed by this line of work involve identifying outliers and dealing with missing values and other glitches in the data. Once again, the user has a central role in many of these research proposals, i.e., the correctness of a repair rule is ultimately decided by the user.

- Record linkage and de–duplication algorithms [ADKC18, NH10, EIV07]: The objective of these techniques is to detect records that contain similar data for the same real world entities. In the process, they employ techniques for indexing, comparison and classification of records to identify duplicates. Once identified, the candidates are presented to human domain experts to decide on their similarity [SBI+13].

Data integration: Integrating multiple data sources and presenting the user with a unified view of the data [DHI12] is a long–standing research problem, with most of the efforts centred on schema matching [RB01], schema mapping generation and selection [MPFK19], [Pap19], and data exchange [ABLM14], in the context of relational databases. Recent research (e.g., [AW18]) studied the problem of data integration in the context of data lakes, taking into account the potential scale of the repository and the potential level of heterogeneity in data models, e.g., structured/semi–structured/unstructured data.

Data discovery: The identification of datasets relevant for a given analysis task is, in comparison, a relatively recent concern. This problem has roots in the source selection challenge [DSS12] that appears in data integration scenarios and aims at balancing the quality of integrated data and integration cost. Source selection starts from a pool of relevant datasets for the given integration task, but with

different degrees of relevance. With the ever increasing availability of data, we find ourselves past the point where one could impose upon any collection of datasets any global, conceptually cohesive, model that captures their interrelationships and allows us to identify that initial pool of relevant sources for a given analytical task. This is the problem that data discovery aims to solve in [DSFG+12], [NZPM18], [FAK+18], [MNZ+18]. This area is further discussed in Chapter 3, since our contributions relating to dataset discovery belong to this category.

Motivated by the above list of challenges, research on data wrangling has inched towards a more holistic view of the process, with recent proposals considering data wrangling as a collection of interconnected processes, involving tasks from all fields mentioned above. For example, D^2WL [SAPS19] is a conceptual data wrangling language, focused primarily on traffic data, aiming to describe several wrangling tasks (e.g., data selection, data cleaning, data integration, etc.) at a high level of abstraction, and to bridge the gap between different platforms that can be used to perform different wrangling tasks.

Other, more general, approaches rely on the interrelationships between data instances from external data/metadata sources, and on user preferences to automate the constituent wrangling sub–processes and decrease human involvement [FGL+16]. This results in a *cost–effective* view of data wrangling, and represents the foundation upon which we build our contributions in this thesis. As such, Section 2.3 of this chapter is dedicated to a detailed exploration of the principles and particularities of this view.

2.2.3 Schema–on–read

We have seen that the clash between current needs and opportunities in data analysis and the data warehouse principles have brought forward the need for a different approach to data management: data processing should allow for the creation of multiple views on the same collection of datasets if this has the potential for uncovering new analysis opportunities. The data lake seems to be an important step towards achieving this ethos because of its pledge for low–cost data storage followed by flexible data processing bounded only by the specifications of the current task, i.e., the input requirements of the current task. This approach is called *schema–on–read* and it tackles the schema–on–write disadvantages presented in Section 2.1.3 as follows:

- Because data is stored without or with minimal processing, it is not uncommon for a data lake to contain semi–structured and unstructured data such as XML files or log files. The consequence is that preparing such data for analysis can be costly in terms of time and required expertise.

- The road to data analysis is *ingest* → *structure* → *analyse*. Therefore, the prescriptive approach imposed by the predefined schema required by the data warehouse is replaced by a descriptive approach, favourable for exploratory data analysis, one that is only governed by the analysis input requirements. This is how the promise of enabling multiple views on the same collection of datasets is fulfilled. The consequence is, of course, the need to perform data preparation *on–demand* - typically by employing data wrangling techniques.

- If on–demand data preparation is possible, then the multiple views on the same data do not generate duplicate data sets, since every new view that stems from the same original is created on–the–fly.

There are two important aspects emerging from the above discussion. Firstly, the need for schema–on–read is not absolute, i.e., schema–on–write may still be the appropriate model to follow if the use case does not require the flexibility of the former. For example, it is worth mentioning that preprocessed data stored in a data warehouse leads to much faster analysis since most of the preparation has already been done by ETL. Secondly, the success of schema–on–read depends on successful on–demand data preparation. Furthermore, it can be argued that a costly data preparation step can undermine the benefits of schema–on–read. When preparing and managing the data stored in a data lake becomes too difficult, the repository is commonly considered a *data swamp*. The next section presents research endeavours that aim to fulfil the promise of cost–effective data wrangling, i.e., reliable and semi–automated data wrangling that dispenses with deep technical knowledge from the user or with too granular user interactions, and, thus, help preventing data lakes from becoming data swamps.

2.3 Cost–effective data wrangling

Data wrangling appears to be both a problem and an opportunity in the field of data preparation. A problem, because, with the advent of data repository types

such as data lakes, data preparation needs to be accessible to different types of users, often unskilled in programming preparation scripts, while still coping with the four V's of Big Data. An opportunity, because the ever increasing availability of data can inform the preparation process and allow for the automation of certain wrangling tasks, thus making the process more cost–effective.

Recent research on the subject (e.g., [FGL+16]), has exposed the double nature of data wrangling and championed research that aims at decreasing the wrangling cost by considering the process as a collection of sub–tasks that can, individually, be automated [Pat19] and, collectively, contribute to making the entire process cost–effective. Specifically, research guided by the vision of data wrangling proposed in [FGL+16], to which we adhere in this thesis, proposes an approach to data wrangling that (i) involves tasks from all fields mentioned in Section 2.2.2, e.g., data discovery, data integration, data cleaning, etc. [KKA+17]; (ii) adopts a best–effort ethos and, therefore, embraces compromise guided by user priorities [AKP+18]; (iii) makes full use of all the available information, including internally curated reference data and knowledge–bases, that can inform the process [KBC+17]; and (iv) adopts an incremental, pay–as–you–go approach [SDH08], [PBE+16] that is guided by the user, thus addressing the needs highlighted in Section 2.2.2, while being cost-effective, i.e., user guidance plays a role in informing automated tasks but does not require deep technical knowledge or too granular interaction [BPE+13].

Inspired by the above approach, we adopt the following definition for cost–effective data wrangling:

Definition 2.1. *A cost–effective data wrangling process combines a collection of wrangling components that between them cover most of the data preparation lifecycle, uses interrelationships between data entities, e.g., datasets, attributes, and external sources of data/metadata to automate each wrangling component, wherever possible, with the aim of decreasing the amount of user involvement and the level of user expertise needed to perform the wrangling tasks.*

In line with Definition 2.1, Figure 2.4 illustrates the elements of such a view of data wrangling. Firstly, data can originate from a data warehouse, or from a data lake, or from any other type of repository, i.e., cost–effective data wrangling is not restricted by the type of backend, rather it is driven by the need for on–demand

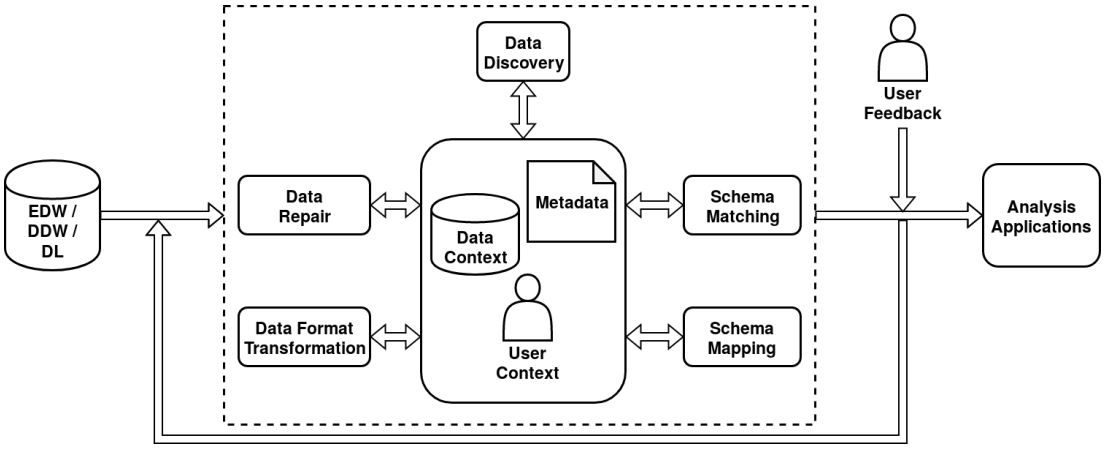


Figure 2.4: Simplified illustration of the interaction between possible steps undertaken when performing cost–effective data wrangling.

data preparation performed by users that are not specialists in data preparation processes. Secondly, data wrangling consists of multiple sub–processes, including, but not limited to:

- Data discovery: for identifying the relevant sources for the current analytical task. This step is necessary when data originates in a data lake or on the web, e.g., [MNZ+18].
- Schema matching: for identifying similar entities that, often, are represented differently, e.g., [RB01].
- Schema mapping: for merging datasets into a unified view upon which analysis is then conducted, e.g., [MPFK19].
- Data format transformation: for normalising the representation of values with the aim of reducing inconsistencies, e.g., [HCG+18].
- Data repair: for identifying missing/incorrect values in data and attempting to correct them, e.g., [FG12].

Section 2.2.2 briefly describes some examples of state–of–the–art approaches for performing each of the above wrangling sub–tasks.

Each of the above processes can be informed by additional sources of information:

- Data context [KBC+17], in the form of reference data, master data and examples data. Reference data is defined as a collection of values that

stipulate the valid domain of a set of specific attributes. Thus, reference data is complete, in that there are few/no missing values, and accurate, in that it provides correct values to be occurring in the final view presented for analysis. Master data is defined as a consistent view on the core entities in an organisation. Although it has the same veracity properties as reference data, master data is an internal source of information to the organisation and includes interrelationships between entities. Example data is defined as a collection of data values that exemplify what is expected to appear in the view presented for analysis, including the format in which the values should be represented. For example, in a real estate scenario, reference data can be represented by a complete list of postcodes from a particular region that are accessible through an open data portal, master data would comprise information about the properties for sale/rent advertised by a particular organisation, including details such as address or cost, and example data could include information about past values for particular attributes such as cost, which may be available from open government statistics.

- User context [AKP+18], in the form of a set of predefined quality criteria for the resulting view passed to the analysis applications. Such criteria could reflect a preference for completeness over correctness, or vice–versa, or could require the presence of certain values and/or attributes in the final result.

- Other data/metadata, including information that results from running individual wrangling sub–tasks and that can inform other sub–tasks, e.g., identifying that two or more datasets are unionable/joinable in the discovery step can be valuable for the schema mapping generation step.

The potential for cost–effectiveness relies on the above sources of information to decrease the level of user involvement in performing tasks, such as in Figure 2.4. In the next two sections, we introduce two examples of challenges faced when performing data wrangling on data lakes, that are amenable to automation and, therefore, have the potential for decreasing the wrangling cost. In our discussion, we focus on two wrangling components: data discovery and data format transformation, and we assume that data wrangling is applied on a data lake. Other examples of tasks that can benefit from the use fo data/user context are discussed in [KBC+17] and [AKP+18].

2.3.1 The scalability challenge

The first source of increased cost when applying data wrangling on data lakes is the potential *scale of the input*. Consider the following example, inspired by [KP19]: a real estate company intends to perform an estimation of the rent/sale prices for a particular geographic area. To this end, it uses data from an open government data lake with information about past property sales, quality–of–life statistics, crime statistics, tax data, etc. The first step in reducing the cost of wrangling is to apply it only on relevant sources, e.g., datasets with properties from the different geographic locations of interest. This is the motivation for the data discovery step, which we define as follows:

Definition 2.2. *Given the target schema of a dataset to be subject to an analytical task (ideally including exemplar tuples and expected attribute names), the data discovery component finds which datasets in the data lake contain relevant information to populate the target (and, therefore, are useful as inputs for data wrangling).*

For the discovery component to reduce the cost implied by the scalability challenge, it must be automated, since manually identifying the relevant datasets is impractical due to the potential size of the repository. In our example, the exemplar tuples mentioned in Definition 2.2, can come from the data context and, once exemplar records are available, the most similar datasets from the data lake can be identified automatically and passed as input to the data wrangling process. Chapter 3 presents our contributions to data discovery and shows how the existence of exemplar tuples can enable the automation of data discovery.

2.3.2 The heterogeneity challenge

Another source of increased wrangling cost is the *inconsistency (or heterogeneity) in format representation* of values. Consider again our real estate example. In such a scenario, addresses are often represented following heterogeneous conventions, e.g., street name followed by street number or vice–versa, using abbreviations such as "*St*" instead of "*Street*", etc. These variations in format require a normalisation step before data is subject to analysis since, often, analytical algorithms assume the input has been normalised [LRG+18]. This is the motivation for the format transformation step, which we define as follows:

Definition 2.3. *Given the target schema of a dataset to be subject to an analytical task (ideally including exemplar tuples and expected attribute names), the data format transformation component carries out changes with a view to normalising the representation of instance values of the same attribute.*

For the format transformation component to reduce the cost implied by the heterogeneity challenge, it must be automated, since manually identifying values to be transformed and manually writing transformation rules is impractical due to the programming steps required and the potential size of the repository. This challenge also impacts on the quality of the outcome of internal components in the wrangling process itself. For example, schema matching relies on information extracted from instance values and would have to take format inconsistencies into account in the absence of a format normalisation step. State–of–the–art schema matching tools (e.g., COMA++ [ADMR05]) normally do not do so.

In Definition 2.3, the exemplar tuples can come, as previously, from data context. Chapter 4 presents our contributions to format heterogeneity and describes how the existence of exemplar tuples can enable the automation of data format transformation.

In the next section, we explore the state–of–the–art in data wrangling, with a focus on how these tackled the two problems, and on how much user involvement is assumed.

2.4 Representative data wrangling systems

In this dissertation, we describe research contributions that can underpin functional components to support dataset discovery and a specific form of format transformations, with the goal of addressing the scalability and heterogeneity challenges of data wrangling in data lakes. In this section, we review the most important wrangling systems from a research perspective. Later chapters, i.e., Chapters 3 and 4, analyse the state-of-the-art in the narrower research areas of dataset discovery and format transformations.

2.4.1 Wrangler

Wrangler [KPHH11] is a descendant of the Potter's Wheel data cleaning system [RH01] and provides interactive mechanisms for transforming data, correcting

erroneous or missing values, and integrating multiple data sources.
Goal: to assist the user in authoring expressive data transformation rules.
Methodology: to achieve its goal by combining the benefits of a declarative transformation language, which is able to express various transformation rules, with a predictive user interaction model [HHK15], which aims to relieve the user from the burden of technical specification.
Operation mode: the users select some data value that is to be subject to a transformation and the system uses predictive methods to suggest a variety of possible next steps that transform the selection (or else the user can author rules directly). Suggestions are presented in the form of rules expressed using a domain–specific language. Rules can be organised as a script for later use on similar datasets. Transformations include operations such as *aggregation*, *mathematical functions*, *string manipulation*, or *merging* of two or more columns.
Cost–effectiveness: the user is relieved from having to cope with the potential size of input datasets by being presented with a subset of the records on which transformations are being designed and tested. If proved useful, the suggestions made by the system can relieve the user from writing the rules as well. However, the user still has to know what transformation is required and how to express the transformation, i.e., the user needs to understand the suggestions. Furthermore, there is no discovery component, so the user has to know what datasets to consider for the given task.

2.4.2 Data Tamer

The Data Tamer system [SBI+13], [SI18] offers user–guided methods for data transformation, schema matching and mapping, and data deduplication.
Goal: to provide scalable data preparation through automation, informed by training data, i.e., it does not require programming knowledge to perform data preparation.
Methodology: to achieve its goal by combining string comparisons based on known string similarity measures (such as Jaccard and cosine) with both distribution–based similarity measures (such as Welch t-test), and user expertise to identify similar attributes and merge them. It also combines clustering algorithms (such as K–means [AV07]) with classification techniques to perform record linkage.
Operation mode: Data Tamer automatically runs several algorithms for each task it performs and appeals to the user for decision making. The system is guided

by an administrator that specifies the collection of data sources for Data Tamer to curate. Then, one ore more domain experts supervise the process by answering questions raised by the automated algorithms that perform the wrangling.
Cost–effectiveness: although the system can perform most of the tasks automatically, user input is still crucial to Data Tamer. The system assumes the domain experts to be able to decide on problematic cases that may require advanced technical knowledge. The processing of large datasets is tackled by employing sampling techniques, but the input itself is still assumed to be provided by the user, i.e., there is no discovery component. The challenge of inconsistent format representations is not considered by this system.

2.4.3 Data Civilizer

The Data Civilizer system [DFA+17] aims to facilitate the analysis of data in the wild (e.g., from data lakes), by offering automated data profiling and discovery, join path selection, and on–demand data cleaning.
Goal: to offer efficient methods for (i) the discovery of datasets of interest from large numbers of tables; (ii) linking relevant datasets; (iii) cleaning the desired data; and (iv) integrating the discovered datasets;
Methodology: to achieve its goal by combining a linkage graph that describes primary key–foreign key and similarity relationships between table attributes, for discovery and linkage, with integrity constraints–based data cleaning for ensuring the quality of the data values.
Operation mode: the discovery task is based on keyword search among the attribute names/instance values. The search for tables is performed through their names, attributes, or content, as well as their relationships with other attributes and tables, as captured in the linkage graph. The data cleaning task is performed with guidance from the user. More specifically, the actual cleaning of a dataset requires feedback from the user on how to change the values of cells that are estimated to be erroneous.
Cost–effectiveness: the discovery process is a central component in Data Cvilizer, therefore, addressing the scalability issue thereby. As for the potential inconsistency of data value representations, this can only be addressed through constraint–based cleaning, which is not expressive enough to catch many inconsistency cases. For example, using such an approach, data values have to be transformed with direct editing from the user. The user is, therefore, actively

involved in the data cleaning process, although the task itself does not require expert–level knowledge.

2.4.4 Value Added Data Systems (VADA)

The VADA system [KKA+17, KAB+19] aims to implement the cost–effective data wrangling paradigm introduced in Section 2.3. It supports most of the data wrangling sub–tasks discussed in this chapter, including data discovery, schema matching and mapping generation, data format transformation and data repair. The methods claimed as contributions in this thesis and described in detail in the next two chapters have been integrated into the VADA system.
Goal: to offer cost–effective data wrangling as defined in Definition 2.2.
Methodology: the wrangling workflow is similar to the one in Figure 2.4, and makes use of several components that employ state–of–the–art methods for Web data extraction [FGG+13], schema matching [DR02], schema mapping [MMP+11], data repair using conditional functional dependencies [FG12], and data format transformations [BPFK18], [BFPK19]. Components are dynamically orchestrated using a Vadalog–based reasoner [BSG18].
Operation mode: with reference to Figure 2.4, given a target schema to be populated with data, each relevant input dataset is automatically cleaned, transformed and merged. Throughout, data/user context information is used to inform the decision–making algorithms [KBC+17], and the user is only asked for feedback in the form of simple questions that do not require expert–level knowledge [KP19].
Cost–effectiveness: the user is relieved from having to manually extract data, write transformation and repair rules, and specify how to merge the relevant datasets by employing automatic algorithms informed by data and user context [KBC+17]. One consequence of simple question–based feedback and of automation is that users have less control over the wrangling process, i.e., they are limited to the capabilities of the automated techniques as informed by user feedback.

2.5 Summary and conclusions

Data preparation has been a significant research concern in Big Data management in recent years [HHK18], with many research proposals on the subject evolving into commercial solutions (e.g., Trifacta (`www.trifacta.com/`, Tamr (`www.tamr.com/`)). These research endeavours and their commercial materialisation are

evidence that there is demand for innovative data preparation proposals. This chapter has briefly explored the research landscape in this area, starting from traditional data warehouses and ETL–based data preparation and continuing with the more liberal data lake repositories that, by adopting the schema–on–read paradigm over the traditional schema–on–write, fuelled the development of new data preparation approaches that enable on–demand data access to a broader range of users. The success of systems such as Wrangler (now commercialised by Trifacta) has demonstrated the viability of data wrangling as the deliverer of self–service data preparation, i.e, data preparation performed by users who know the data best, even if they are not IT experts. But the ever increasing availability of data and analysis opportunities asks for cost–effectiveness, where, among other desiderata, the user should not need to be highly skilled in manually conducting/guiding wrangling tasks, nor need to manually identify the datasets relevant for the task at hand (when applied on data from a data lake).

In the next two chapters, we discuss in detail the techniques devised to automate data discovery and data format transformation cost–effectively.

Chapter 3

Dataset discovery in data lakes

This chapter focuses on techniques employed to perform the dataset discovery task in data wrangling. We have defined the process of dataset discovery in Definition 2.2 as a functional component of data wrangling that identifies the relevant input that can lead to the the construction of specific target datasets used for value–adding analytics in data lakes.

The primary goal of dataset discovery is to select for relevance from an (otherwise unmanageable) collection of data sources those datasets on which to run tasks such as schema matching (e.g., [RB01]), format transformation (e.g., [BFPK19], [Gul11], [KHP+11])), or schema mapping generation (e.g., [MPR09], [MPFK19]). So, given a target (which ideally includes exemplar tuples and expected attribute names), the dataset discovery process aims to find which datasets are most useful as inputs for data wrangling. By most useful, we mean datasets (or possibly projections thereof) that are *unionable* with the target, and, desirably, *joinable* with each other. So, given a target, a solution to this dataset discovery problem, returns a set of datasets, some of which are potentially horizontal fragments, and, when possible, others that are potentially joinable with the horizontal fragments to contribute additional attributes to the target. We refer to such datasets as *related* to the given target. Thus, the objective of dataset discovery is:

> *To identify related datasets from a data lake that are relevant for populating as many target attributes as possible.*

In Section 3.1, we motivate our desiderata for the proposed dataset discovery approach. Then, in Section 3.2, we set the scene for this chapter by introducing

the theoretical concepts that underpin our approach, together with the state–of–the–art on dataset discovery. Next, we state the main contributions of this chapter and introduce an overview of the approach in Section 3.3. Section 3.4 discusses relatedness at attribute level and presents what types of similarity evidence are employed, the associated distance measures, and how we rely on specialised search techniques to efficiently compute the distance values and to find related attributes. Section 3.5 moves the discussion from attribute–level to dataset–level by describing how multiple attributes from the same dataset and five types of distances contribute to a dataset–level distance measure that quantifies the relatedness between the target dataset and other datasets relevant for populating the target. Relatedness–by–joinability is then covered in Section 3.6, where we show how the same data structures used to identify related datasets can also be used to derive postulated join paths between datasets, with the potential for increasing the coverage of target attributes. Finally, we empirically evaluate our approach, with a focus on both efficiency and effectiveness, against the closest state–of–the–art competitors.

3.1 Motivation and desiderata

To achieve a better understanding of the ramifications of the problem addressed in this chapter, we introduce a set of desiderata, motivated by specific characteristics of data lakes:

Missing metadata. A data lake often provides limited metadata and it is often hard to link its contents to external sources of information such as domain ontologies that could define the entities described in the datasets. Therefore, we need to induce characteristics from the data itself that can be reliably used as sources of evidence for the discovery of relatedness relationships between datasets. This evidence should be quantifiable, minimally dependent on supplementary data that may not always be available, and tolerant of different levels of metadata and schema–level information associated with the sources.

Schema heterogeneity. A data lake often contains datasets from many different sources that obey different rules, so some types of relatedness evidence may only hold for a few datasets, but not all. For example, attribute names may clearly describe the domain of attributes for some datasets, while being

abstract or even missing altogether for other datasets. Therefore, we need relatedness evidence types that rely on different characteristics of the data, and a balanced aggregation method that balances the weight given to each evidence type.

Format representation heterogeneity. A data lake often contains datasets with inconsistent information about the same real–world entities in terms of how data values are represented. Therefore, we need relatedness evidence types that are lenient in identifying related datasets with respect to format inconsistencies, thereby reducing the negative impact of format heterogeneity.

Scale. A data lake often contains a large number of datasets. For example, the open government data lakes mentioned in Chapter 2 contains tens of thousands of datasets. Therefore, we need a scalable technique that, while being able to benefit from multiple types of relatedness evidence, avoids all–against–all comparison methods that are quadratic on the number of items compared.

3.2 Background and related work

Before describing how we meet the above desiderata, we define the central notion of this chapter, viz., *dataset relatedness*, and discuss the technical background upon which our approach builds. Firstly, we introduce the concepts that ground our approach, and, secondly, we introduce the state–of–the–art methods for finding related/similar/unionable datasets in data lakes.

3.2.1 Dataset relatedness

We formally define the *relatedness* of a dataset S with respect to a target T as follows:

Definition 3.1. *Given a dataset S with attributes $a_1, \ldots, a_n$ and a target dataset T with attributes $a'_1, \ldots, a'_m$, we say that S and T are related iff $\exists a_i \in \{a_1, \ldots, a_n\}$, such that a_i contains values drawn from the same domain represented by some attribute $a'_j \in \{a'_1, \ldots, a'_m\}$, and, therefore, is relevant for populating a'_j, i.e., a_i and a'_j are attribute–level related.*

For simplicity, we only consider one–to–one relatedness between attributes. In a many–to–one (or one–to–many) case, i.e., when an attribute from S is related to more than one attribute from T (or vice–versa), we choose the most related attribute pair.

Given two datasets S_1 and S_2 related with respect to a target T, the following properties follow from Definition 3.1:

- S_1 and S_2 can have different degrees of relatedness to T, subject to how many of their attributes are related to some target attribute and to how strongly related those attributes are.

- S_1 and S_2 are *unionable* on the attributes related to the same target attribute and each is unionable with the target itself. We focus on *relatedness–by–unionability* in Section 3.4.

- If S_1 and S_2 are *joinable* as well, then the projection of the attributes related to some target attribute over the result of joining them on those attributes is, potentially, more related to T than S_1 and S_2 individually. We explore this property in Section 3.6.

We consider attribute–level relatedness to also imply attribute–level similarity, and quantify the former using distance measures, often used to also quantify the latter: the closer, the more similar, and the more similar, the more related.

3.2.2 Sources of relatedness evidence

In performing dataset discovery, we employ the following sources of similarity evidence, some of which originate from research on schema and ontology matching (e.g., [DR02], [ADMR05]) and on distributional semantics (e.g., [TP10]).

Instance values set-similarity between two attributes a and a' is defined as the ratio of shared items between their value sets. When quantified, the *similarity* $sim(a, a')$, is a number between 0 and 1, such that the closer $sim(a, a')$ is to 1, the more similar the two sets are. The similarity has the additional property that it can be seen as a distance measure $dist(a, a') = 1 - sim(a, a')$, often used in algorithms that aim to cluster a collection of sets into groups of closely resembling sets [Bro97]. Examples of systems that rely on instance–based similarity measures include schema and ontology matching systems such as COMA++ [ADMR05]

and DUMAS [BN05]. In our approach we employ instance values similarities for identifying related attributes with overlapping instance values.

Attribute name similarity is defined as the similarity between the strings representing the attribute names of two attributes. Three types of string similarity measures are prominent in the literature: *token–based*, *character–based*, and *hybrid* [YLDF16]. *Token–based* measures model each string as a set of tokens and quantifies the similarity between two strings based on the common tokens shared by their corresponding token sets, e.g., Jaccard similarity, cosine similarity. *Character–based* measures model each string as a sequence of characters and quantify the similarity between two strings based on the number of different characters in the two sequences, e.g., edit or Levenshtein distance calculates the minimum number of single–character insertions, deletions and substitutions that transforms one string into another. Lastly, *hybrid* measures combine similarities at token and character levels, e.g., [WLF11], [WLF14]. Systems such as COMA++ [ADMR05] use attribute name similarity when deciding on the equivalence of two attributes for schema matching purposes. We also use attribute name similarity, in addition to instance values overlap, as evidence in our approach.

Contextual similarity is used as evidence in the case of attributes whose names and extents are expressed in natural language, e.g., person names, addresses, product descriptions, etc.. This means that such values can often be found in textual corpora such as news articles, books, Wikipedia articles, etc. The existence of such corpora has been exploited in areas such as distributional semantics based on vector space models (VSM) to quantify the similarity of two strings using their context (roughly, their surrounding terms), as defined by the given text corpora [TP10]. The hypothesis upon which such approaches are grounded and which justifies the application of VSM in measuring word similarities, is that words that occur in similar contexts tend to have similar meanings. Examples of systems that rely on VSM to measure the similarity of words include INDRA [SSB+18] and GenSim [RS10]. We use contextual similarity as evidence in deciding on attribute relatedness when instance–based and attribute name–based techniques are not effective.

Ontologies and knowledge–bases are also sources of similarity signals that do not rely on the resemblance of data values. Examples include YAGO [SKW07], Wikipedia and other Web table corpora [CHW+08], WordNet [Mil95], and DBpedia [ABK+07]. Such sources of evidence are used to map individual

terms in the attribute names/values to entities in an ontology, or to identify relationships between attributes, that can lead to a decision on their similarity. Proposals that do this include [VHM+11] and [LSC10]. We do not use such external sources of similarity evidence in our approach because we found that only very seldom can data values or attribute names in a data lake be mapped to an external ontology or knowledge–base.

Statistical similarity is used as evidence when the attributes that are compared have numerical domains. When this is the case, evidence types that rely on overlap or annotation of instance values can only emit very weak similarity signals. This is because numbers cannot be analysed in terms of their individual values as usefully as text can. Alternatively, numbers can be meaningful when considered as part of a probability distribution. For instance, given two attributes, one denoting age of persons, and another denoting the weight of persons, the existence of the same individual value, say 68, in each attribute extent does not stand as evidence of similarity. However, if we consider the entire distribution of values for each attribute, more evidence about their dissimilarity can be gleaned. A number of statistical measures that quantify the distance (or similarity) between two statistical objects, such as two probability distributions, have been proposed. Examples include the Kullback–Leibler (KL) divergence [KL51], the Kolmogorov–Smirnov (KS) statistic [Con99a], or the Jensen–Shannon (JS) divergence [DLP02]. We employ the KS statistic to decide on the relatedness of numerical attributes.

3.2.3 Scalable relatedness discovery through Locality Sensitive Hashing

The problem of identifying similar/duplicate elements has been studied, in areas such as Information Retrieval, in the context of document clustering (where similar text objects, e.g., sentences, paragraphs, or documents, are grouped together (e.g., [AZ12])), or in the context of Web pages (where near–duplicate pages are identified to increase the quality of a web crawler, e.g., [MJDS07], [MKK+08])). The database research area has had a long–standing interest in the subject as well, in the context of record linkage and deduplication (where similar techniques are used to match records from several databases that refer to the same entities (e.g., [Chr12])). In all these research endeavours, besides the features that describe the objects being compared, which often rely on the types of similarity

evidence mentioned above, the crux of the matter lies in the comparison itself, i.e., how to perform efficient similarity grouping on many objects so as to avoid all–against–all comparisons which can be very computationally expensive. Since it is often the case that most comparisons are between objects that are dissimilar, an important technique known as blocking [BCFE03], can be used to reduce the large number of potential comparisons by removing as many dissimilar candidate pairs as possible. One prominent idea in this direction is to use hashing for similarity search [WTSSJ14], with proposals that perform nearest–neighbour search, (e.g. [AI08], [IM98]), proving to be effective in reducing the number of potentially similar objects [AAK10] before more computationally expensive comparisons are performed.

Locality Sensitive Hashing (LSH) [IM98] is a randomised hashing framework for efficient approximate nearest–neighbour search in high–dimensional spaces. It is based on a family of hash functions that map similar input items to the same hash code with higher probability than dissimilar ones. Before formally defining LSH, we introduce necessary definitions and notation.

The nearest–neighbour search problem is defined as: given a query item q, find an item $NN(q)$, called a nearest–neighbour, from a set of items $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, so that $NN(q) = \arg\min_{x \in \mathcal{X}} dist(q, x)$, where $dist(q, x)$ is a given distance function that measures the similarity between q and x. The K–NN search problem is a generalisation where the K–nearest neighbours are searched.

The fixed–radius near neighbour (R–near neighbour) problem, an alternative to the nearest neighbour search, is defined as: given a query item q, find the items $\mathcal{R}$ that are within a distance R from q, $\mathcal{R} = \{x | dist(q, x) \leq R, x \in \mathcal{X}\}$.

Lastly, the c–approximate R–nearest neighbour search problem is defined as: given a query q, find some item x, called (R, c)–near neighbour, so that $dist(q, x) \leq cR$, for some constant $c > 1$, called the approximation factor, such that $dist(q, x)$ is no more than c times the distance of the nearest point to q.

LSH was formally introduced in [IM98] and [GIM99] as a probabilistic approach to the (R, c)–near neighbour problem in d–dimensional spaces, based on a family of hash functions $\mathcal{H}$, with the property that *the collision probability of such hash functions is high for inputs that are similar and low for those that are not.* Formally, an LSH family of hash functions is defined as follows:

Definition 3.2. *A family $\mathcal{H}$ is called (R, cR, P_1, P_2)–sensitive if, for all $h \in \mathcal{H}$ and for any two items p and q, the following conditions hold:*

- *if $dist(p, q) \leq R$, then $Prob[h(p) = h(q)] \geq P_1$,*
- *if $dist(p, q) \geq cR$, then $Prob[h(p) = h(q)] \leq P_2$*

where $c > 1$ and $P_1 > P_2$.

Intuitively, a distance of at most R between points gives a higher collision probability between their hash codes than the probability given by more distant points. It follows that, given two near neighbours p' and q', $P_2 < Prob[h(p') = h(q')] < P_1$.

According to [IM98], in practice, given an LSH family $\mathcal{H}$, in order to increase the chance of collisions between hash codes of near similar candidates, the LSH scheme amplifies the gap between the upper (i.e., P_1) and lower (i.e., P_2) probability bounds by concatenating several hash functions from $\mathcal{H}$. In particular, K functions $h_1, \ldots, h_K$ are chosen independently and uniformly at random from $\mathcal{H}$ to form a compound hash function g. Given a new instance x, the output of $g(x) = (h_1(x)|| \ldots ||h_K(x))$ is a bit pattern that identifies a bucket in a hash table where x is to be stored ($||$ denotes concatenation between the bit patterns returned by each $h_i(x)$). However, increasing the gap between P_1 and P_2 reduces the chance of hash code collisions between similar items. To improve the recall, L compound functions $g_1, \ldots, g_L$, are chosen, each of which corresponding to a different hash table. These functions are used to hash each item x into L hash codes, and L hash tables are constructed to index the buckets that correspond to these hash codes, respectively. Each bucket in the hash table is identified by a hash code computed using the concatenation of the results of the K hash functions from $\mathcal{H}$ that aim to maximise the probability of collisions for near items. Given a query q, the items lying in the L buckets identified by $g_i(q)$, with $1 \leq i \leq L$, are retrieved as similar candidates to q. Consequently, in practice, the LSH scheme has to be configured with values for K and L. Often, this is done through the simple configuration of a similarity threshold, with respect to the similarity measure used, since it is possible to analytically determine effective values for K and L given the similarity threshold [RLU14], [SLH12].

Finally, resolving the (R, c)–near neighbour problem using an LSH family of hash functions, given a query q, is efficient by the following theorem:

Theorem 1. *Given an (R, cR, P_1, P_2)–sensitive family of hash functions for a distance D, there exists an algorithm for the (R, c)–near neighbour problem under distance D which uses $\mathcal{O}(dn + n^{1+\rho})$ space, with query time dominated by $\mathcal{O}(n^{\rho})$ distance computations, and $\mathcal{O}(n^{\rho} \log_{1/P_2} n)$ evaluations of hash functions, where $\rho = \frac{\log(1/P_1)}{\log(1/P_2)}$.*

The proof for Theorem 1 can be found in [IM98] or [DIIM04].

3.2.4 LSH hash functions and similarity measures

We now turn our attention to exemplar LSH hash functions and similarity distances that can be used in practical applications of (R, c)–near neighbour search.

MinHash, introduced in [Bro97] and [BCFM98], is an LSH hashing scheme that returns hash codes with high probability of collisions for sets with high Jaccard similarity. Given two sets A and B, their Jaccard similarity is given by:

$$jacc(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

MinHash is defined as follows: given a universe U from which elements of two sets A and B, which we assume are arbitrarily ordered, are drawn, pick a random permutation Π from U, and define $h(A) = \min_{a \in A} \Pi(a)$, where the minimum is applied on the numerical order of elements in the permutation. It is shown in [BGMZ97] that

$$Prob[h(A) = h(B)] = jacc(A, B) \tag{3.2}$$

In practice, the equality in Equation 3.2 becomes an approximation since it is unfeasible to generate a truly random permutation and, therefore, approximate techniques for generating Π are used (e.g., [BGMZ97]).

Random projections–based LSH scheme [AI08], [Cha02] has been proposed to solve the near–neighbour search problem under the angular distance between vectors. For two multi–dimensional vectors $\vec{v}$ and $\vec{u}$, the angular distance θ between them is given by:

$$\theta(\vec{v}, \vec{u}) = \arccos \frac{\vec{v}^T \vec{u}}{||\vec{v}||||\vec{u}||} \tag{3.3}$$

An associated hash function is defined in [Cha02] as follows. First, a vector $\vec{r}$ is chosen at random from a multi–dimensional Gaussian distribution. Then, a hash function $h_{\vec{r}}$ of a vector $\vec{u}$ is defined as follows:

$$h_{\vec{r}}(\vec{u}) = \begin{cases} 1 & if\ \vec{r} \cdot \vec{u} \geq 0 \\ 0 & if\ \vec{r} \cdot \vec{u} < 0 \end{cases}$$

Then, [Cha02] shows that for two vectors $\vec{v}$ and $\vec{u}$,

$$Prob[h_{\vec{r}}(\vec{v}) = h_{\vec{r}}(\vec{u})] = 1 - \frac{\theta(\vec{v}, \vec{u})}{\pi} \tag{3.4}$$

In practice, Equation 3.4 is often used to approximate the cosine similarity between $\vec{v}$ and $\vec{u}$. This is possible because, as shown in [Cha02], the function $1 - \frac{\theta(\vec{v},\vec{u})}{\pi}$ is always within a factor of 0.878 from the function $\cos\theta$ (the cosine similarity).

LSH schemes for other similarity measures have been proposed in the literature, including for the l_p distance [DIIM04] and for the Hamming distance [IM98]. The approach proposed in this chapter builds on Equations 3.2 and 3.4, and, therefore, we do not discuss any other LSH schemes.

3.2.5 Dataset discovery: state–of–the–art

Data lakes are usually seen as vast repositories of company, government or Web data (e.g., [CHW+08, EMH09]). Previous work has considered dataset discovery in the guise of table augmentation and stitching (e.g., [LB17, LHWY13]), unionability discovery, or joinability discovery (e.g., [DSFG+12, NZPM18, FAK+18, FMQ+18]). We add to this work with a focus on a notion of relatedness, defined in the next section, construed as unionability and/or joinability.

LSH-based dataset discovery. LSH has been adopted by the data management research community due to its useful properties regarding similarity

estimation at linear retrieval times w.r.t. the search space size [MNZ+18]. One example is Aurum [FAK+18] (and its extension from [FMQ+18]), a system to build, maintain and query an abstraction of a data lake as a knowledge graph. Similarly, Table Union Search [NZPM18] focuses on the problem of unionability discovery between datasets, treated as an LSH index lookup task. As we do, both proposals use LSH–based indexes to efficiently search for related attributes in data repositories. While the underlying data structures used in both cases are similar to the ones we rely on, there are a number of key differences: (i) we make use of more types of similarity, whose combined import is to inform decisions on relatedness with a diversity of signals; (ii) we adopt an approach based on schema– and instance–level fine–grained features that prove more effective in identifying relatedness, especially in cases when similar entities are inconsistently represented; and (iii) we map these features to a uniform distance space that offers a holistic view on the notion of relatedness between attributes, to which each type of similarity evidence contributes, as instructed by an underlying weighting scheme. We describe in detail how we achieve these advantages in the next sections.

Web table augmentation and search. The idea of table augmentation relies on extending Web tables with additional useful information from external sources that improve the results of schema matching (e.g., [LB17, LHWY13]) or facilitate replies to queries about entities of interest (e.g., [YGCC12]). In the former case, the idea is to combine Web tables, which are often small, into larger ones and to match these enlarged tables with knowledge bases or large base tables, with the intuition that the combined tables emit a stronger similarity signal that can inform the matching process. In the latter case, the goal is to create and/or populate a target schema with information from Web sources, starting from a set of entities of interest. To this end, similarity discovery, based on attribute names and instance values, is employed to discover Web tables that can contribute with relevant information. In our work, our backend is a data lake, and not the Web, and our objective is to enable data wrangling on vast repositories of data by selecting only the relevant datasets given a target. In doing so, we do not assume the existence of external augmentative or descriptive information from Web tables or ontologies.

Web data integration. The discovery of unionable and joinable Web tables has been studied in Octopus [CHK09], which combines search, extraction and cleaning operators to cluster datasets from the Web into unionable groups

by means of keyword-based string similarities and Web metadata. [DSFG+12] identifies entity–complementary (i.e., unionable) and schema–complementary (i.e., joinable) tables by using external knowledge bases to label datasets at instance and schema levels. This leads to a decision on unionability and joinability of those tables. We also search for related tables but because we envisage the need for downstream wrangling we assume a target table and refrain from relying on Web knowledge bases or external metadata about tables in the lake because such data would not always be available.

Data lake management systems: Data lakes have been the focus of recent research on data management systems, e.g., [TSRC15, HKN+16]. Such proposals focus on data lifecycle and rely on extensible metadata models and parsing frameworks for different data types, tailored for the challenges faced by the organisation that builds and uses the data lake, e.g., Goods [HKN+16], is highly oriented towards rapidly changing data sets.

3.3 Overview and contributions

We now present an overview of our approach. Recall that our goal in this chapter is to identify datasets from a data lake that can contribute information for populating a given target. Consider Figure 3.1 as an example. The target T contains information about general practices (i.e., family doctors/primary care centres). We want to find tables (e.g., S_1 and S_2) that are useful for populating T. Moreover, if there is insufficient evidence that S_3 and T are related as well, we want to find join opportunities (e.g., of *Practice Name/Practice* in S_1/S_2 with *GP* in S_3) that allow us to increase target coverage (e.g., by populating *Hours* in T).

Figure 3.2 illustrates the overall approach described in this chapter. The approach can be broadly seen as similarity–based in the sense that, from the attribute names and values in each dataset given at input, we extract features that convey signals of similarity used in quantifying the degree of relatedness between datasets, as defined in Definition 3.1. As shown in Figure 3.2, we extract five types of features which we map to buckets in LSH indexes, thereby guaranteeing that shared bucket membership (i.e., the proximity of the hash keys generated for each attribute) is indicative of similarity, as per the hash function used. Then, given a target dataset, the task of identifying related datasets from a data lake is conducted by repeating the same operations illustrated in Figure 3.2 for the target

S_1: Source: GP practices

Practice Name	Address	City	Postcode	Patients
Dr E Cullen	51 Botanic Av	Belfast	BT7 1JL	1202
Blackfriars	1a Chapel St	Salford	M3 6AF	3572

S_2: Source: GP funding

Practice	City	Postcode	Payment
The London Clinic	London	W1G 6BW	73648
Blackfriars	Salford	M3 6AF	15530

S_3: Source: Local GPs

GP	Location	Opening hours
Blackfriars	Salford	08:00-18:00
Radclife Care	-	07:00-20:00

T: Target: GPs

Practice	Street	City	Postcode	Hours
Radclife	69 Church St	Manchester	M26 2SP	07:00-20:00
Bolton Medical	21 Rupert St	Bolton	BL3 6PY	08:00-16:00

Figure 3.1: Examples of source and target records involved in a dataset discovery process.

attributes. In other words, the LSH buckets where the target attributes end up give the related attributes from which an aggregated, dataset–level relatedness measure is computed for each parent dataset. Lastly, Figure 3.2 suggests the use in this process of two similarity measures (i.e., Jaccard and Cosine similarities), and a different treatment for categorical/textual and numerical values. The specifics and reasoning behind such design choices are detailed throughout this chapter.

More formally, given a data lake $\mathcal{S}$, and a target attribute t with exemplar values, the dataset discovery process finds the set of attributes $\mathcal{A}$ from datasets in $\mathcal{S}$, that are within a distance τ from t (i.e., $\mathcal{A} = \{a | dist(t, a) \leq \tau\}$), with τ a preconfigured threshold parameter. To account for multiple cases, including when (a) exemplar target values are insufficient or unavailable, (b) exemplar target values are not shared by other attributes in the data lake, or (c) the attributes are numerical, we consider five nearest–neighbour sub–problems, each taking a different distance measure. Each such distance measure quantifies a relatedness signal between t and each candidate from $\mathcal{A}$, one for each type of similarity evidence used.

Overall, we make the following contributions:

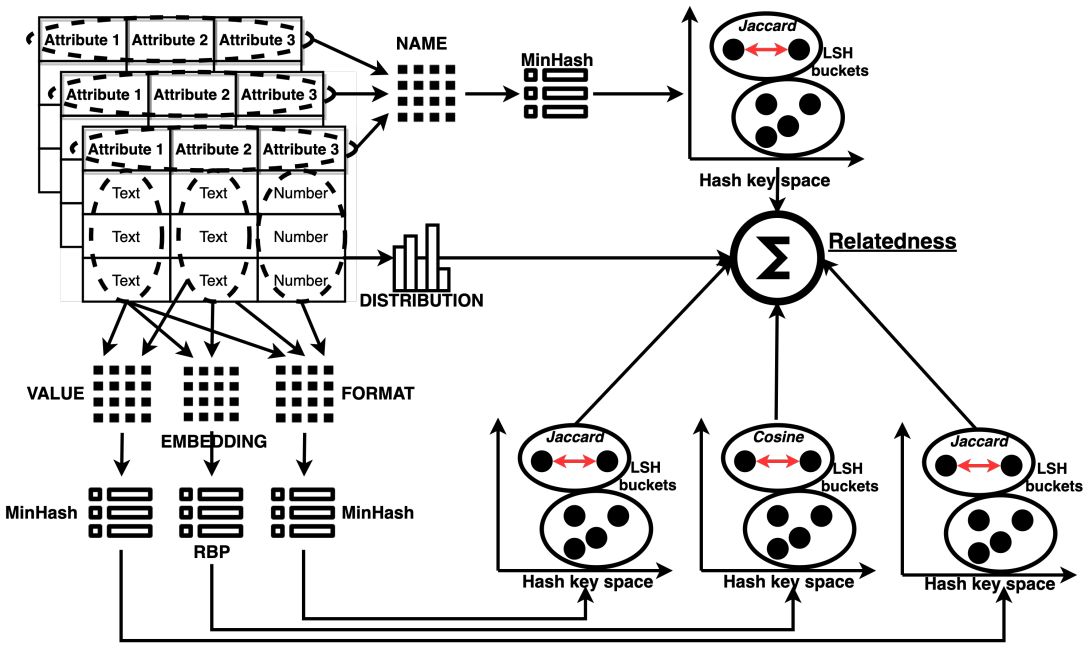


Figure 3.2: Simplified illustrations of the data discovery process applied on a collection of datasets.

- We propose a new distance–based framework that, given a target, can be used to efficiently determine the relatedness between attributes in the target table and datasets in a lake. We do this using five types of evidence: (i) *attribute name similarity*, when schema–level information is available; (ii) *attribute extent overlap*, when attributes share common values; (iii) *word–embedding similarity*, when attributes are semantically similar but have different value domains; (iv) *format representation similarity*, when attribute values follow regular representation patterns; and (v) *domain distribution similarity*, for numerical attributes.

- We show how to map the signal from each of the above evidence types onto a common space, such that the resulting attribute distance vectors combine the separate measurements of relatedness, and propose a weighting scheme that reflects the signal strength from different evidence types.

- We extend the notion of relatedness to tables whose similarity signal with the target is weak but that join with tables that contribute values to additional target attributes.

- We empirically show, using both real–world and synthetic data, that our approach is significantly more effective and more efficient than the closest competitors from the state-of-the-art.

3.4 Attribute relatedness

We first aim to identify related attributes, i.e., attributes whose values can be used to populate some attribute in the target, and to quantify their degree of relatedness. Strictly, this can only be done if they store values for the same property type of the same real world entity. However, data lakes are characterised by a dearth of metadata. There is a need, then, to postulate whether two attributes from two datasets are related, relying only on evidence that the datasets themselves convey.

3.4.1 Relatedness evidence

We use five types of evidence for postulating on attribute relatedness: names ($\mathbb{N}$), values ($\mathbb{V}$), formats ($\mathbb{F}$), word–embeddings ($\mathbb{E}$) [MSC+13], and domain distributions ($\mathbb{D}$). From names we derive q–grams; from values we derive tokens, format–describing regular expressions and word–embeddings; and from extents we derive domain distributions. Note that, in the case of both attribute names and attribute values, we break up string representations with a view to obtaining finer–grained evidence. In other words, we operate at sub–token level, e.g, n–grams, as opposed to operating at full names/values level. The motivation is the expected "dirtiness" of the data lake, e.g., attributes may have names or values that denote the same real–world entity but are represented differently. Using such evidence implies that our approach is lenient in identifying related attributes, reducing the impact of dirty data. This is an important point of contrast with related work, as the experimental results will show.

Let a and a' be attributes with extents $[\![a]\!]$ and $[\![a']\!]$, resp. We now describe how we aim to capture similarity signals for each type of evidence:

$\mathbb{N}$: given an attribute **name**, we transform it into a set of q–grams ($q$set, for short), aiming to construe relatedness between attribute names as the Jaccard distance between their $q$sets. Let $Q(a)$ denote the $q$set of a.

$\mathbb{V}$: given an attribute **value**, we transform it into a set of informative tokens ($t$set,

for short). By informative, we mean a notion akin to term–frequency/inverse–document–frequency (TF/IDF) from information retrieval, as explained later. We aim to construe relatedness between attribute values as the Jaccard distance between their $t$sets. Let $T(a)$ denote the union of the $t$sets of every value in $[\![a]\!]$.

$\mathbb{F}$: given an attribute value, we represent its **format** (i.e., the regular, predictable structure of, e.g., email addresses, URIs, dates, etc.) by a set of regular expressions ($r$sets, for short) grounded on a set of primitives we describe later. We aim to construe relatedness between attribute value formats as the Jaccard distance of their $r$sets. Let $R(a)$ denote the union of the $r$sets of every value v in $[\![a]\!]$.

$\mathbb{E}$: given an attribute value that has textual content, we capture its context-aware semantics as described by a **word–embedding** model (WEM) [GMJB17], as follows: each word in the attribute value is assigned a vector (with a WEM–specific dimension p) that denotes its position in the WEM–defined space. The p-vectors of each such word are then combined into a p–vector for the whole attribute. We aim to construe relatedness between attribute values with textual content as the cosine distance of their vectors. Let $\vec{a}$ denote the set of word–embedding p–vectors of every value in $[\![a]\!]$.

$\mathbb{D}$: given a numeric attribute, only $\mathbb{N}$ and $\mathbb{F}$ are useful in construing similarity, as the others (viz., $\mathbb{V}$ and $\mathbb{E}$) are dependent on the existence of structural components (viz., tokens and words) that can only be reasonably expected in non–numeric data. So, we aim to construe relatedness between numeric attribute values as the Kolmogorov-Smirnov statistic (KS) [Con99b] over their extents understood as samples of their originating **domain**. The smaller KS is, the closer the attributes are w.r.t. to their value distribution.

3.4.2 Distance measures

Each of the five types of evidence above gives rise to a distance measure bounded by the $[0, 1]$ interval. Given two attributes a and a', from different features of their respective names and extents, all of which carry useful but different signals of relatedness, we can compute the following distances:

- From attribute **names** we derive $q$sets and compute the Jaccard distance between $q$sets:

$$D_{\mathbb{N}}(a, a') = 1 - \frac{Q(a) \cap Q(a')}{Q(a) \cup Q(a')}$$

- From attribute **values** we derive *t*sets and compute the Jaccard distance between *t*sets:
$$D_{\mathbb{V}}(a, a') = 1 - \frac{T(a) \cap T(a')}{T(a) \cup T(a')}$$

- From **formats** we derive *r*sets and compute the Jaccard distance between *r*sets:
$$D_{\mathbb{F}}(a, a') = 1 - \frac{R(a) \cap R(a')}{R(a) \cup R(a')}$$

- From **embeddings** we derive vectors and compute the cosine distance between them:
$$D_{\mathbb{E}}(a, a') = 1 - \frac{\vec{a}^T \vec{a}'}{||\vec{a}|| \cdot ||\vec{a}'||}$$

- From domain extents, in the case of attributes with numerical values, we derive domain distributions and compute the KS statistic between them:

$$D_{\mathbb{D}}(a, a') = \sup_x |F_a(x) - F_{a'}(x)|$$

  Intuitively, $D_{\mathbb{D}}(a, a')$ is given by the maximal difference between cumulative probabilities computed at a value x.

In order to avoid carrying pairwise comparisons in computing the above distances, as others have done (e.g., [NZPM18], [FAK+18]), we adopt an approximate solution based on LSH that offers efficient distance computation, at the potential expense of accuracy. To this end, we use Jaccard and cosine distances because of their property of being *locality–sensitive* ([Bro97, Cha02]). Specifically, the probability that MinHash/random–projections returns the same hash value for two sets is approximately equal to their Jaccard/cosine similarity. Since $\mathbb{N}$–, $\mathbb{V}$–, $\mathbb{F}$–relatedness are grounded on Jaccard similarity, and $\mathbb{E}$–relatedness is grounded on cosine similarity, we use MinHash/random projections to efficiently approximate the above distances. We do not use the same strategy for $\mathbb{D}$–relatedness because there is no corresponding LSH hashing scheme that leads to analogous gains.

The combined import of the above measurements is to inform decisions on relatedness with a diversity of signals. The smallest such distances are, the more closely related the attributes, and since the distances are defined over different features of the data, they provide more viewpoints on relatedness.

3.4.3 Index construction

In our approach, given a data lake $\mathcal{S}$ and a target table T, finding the set of related datasets in $\mathcal{S}$ to T is a computational task performed after indexing $\mathcal{S}$. For a given $\mathcal{S}$, we build four LSH indexes, which, resp., are used to compute $\mathbb{N}$–, $\mathbb{V}$–, $\mathbb{F}$–, and $\mathbb{E}$–relatedness between attributes. We call these indexes $I_{\mathbb{N}}$, $I_{\mathbb{V}}$, $I_{\mathbb{F}}$, and $I_{\mathbb{E}}$, resp. Given two attributes a and a', they are $\mathbb{N}$ (resp., $\mathbb{V}$, $\mathbb{F}$, and $\mathbb{E}$)–related if $a' \in I_{\mathbb{N}}.lookup(a)$ (and resp. for the other indexes). Index insertion is shown in Algorithm 3.1. The subroutines in sans–serif are described below, by reference to Example 3.1.

Example 3.1. *Let a be an attribute, with name Address and extent* $[\![a]\!] = \{$ *'18 Portland Street, M1 3BE', '41 Oxford Road, M13 9PL' '9 Mirabel Street, M3 1NN'* $\}$.

Given an attribute a, we call the sets associated with each relatedness type (i.e., its $q$set, $t$set, $r$set and word-embedding vectors $\vec{a}$) the *set representations* of a.

- get_qgrams(a): Obtaining the $q$set $Q(a)$ of an attribute a is the straightforward procedure of computing the q–grams of its name. We have used $q = 4$ as this avoids having too many similar $q$set pair candidates, while benefiting from fine–grained comparisons of attribute names. For Example 3.1, get_qgrams(a) = {addr, ddre, dres, ress}.

- frequent/infrequent tokens: The $t$set $T(a)$ and word–embedding vector $\vec{a}$ of an attribute value are obtained in tandem by construing the extent of a as a set of documents, a value v as a document, each document as a set of parts (split at punctuation characters), and each part as a set of words. With one pass on the extent, we tokenize the values (get_tokens(v)) and construct a histogram of token occurrences (which we assume to have an associated data structure from which we can retrieve its frequent and infrequent token sets). Then, for each part in the value/document, the procedure (a) adds to $T(a)$, the word t in that part that has the fewest occurrences in the extent, and (b) takes the word in that part that has the most occurrences in the extent, retrieves its word-embedding vector from a WEM (e.g., *fastText* [GMJB17]) and adds that vector to $\vec{a}$. For Example 3.1, get_tokens(a) = {portland, 3BE, oxford, 9PL, ...}. Note that since terms like "*street*", "*road*", or the area–level tokens in the UK postcode information

Algorithm 3.1 Index construction

Input: Indexes $I_{\mathbb{N}}$, $I_{\mathbb{V}}$, $I_{\mathbb{E}}$, $I_{\mathbb{F}}$, Attribute a
Output: Updated $I_{\mathbb{N}}$, $I_{\mathbb{V}}$, $I_{\mathbb{E}}$, $I_{\mathbb{F}}$

```
1: function InsertIntoIndexes
2:     Q(a) ← {}; T(a) ← {}; R(a) ← {}; a⃗ ← {};
3:     H ← histogram.new()
4:     Q(a) ← get_qgrams(a)
5:     for all v ∈ ⟦a⟧ do
6:         H.insert(get_tokens(v))
7:         R(a) ← R(a) ∪ get_regex_string(v)
8:     end for
9:     for all t ∈ H.infrequent() do
10:        T(a) ← T(a) ∪ {t}
11:    end for
12:    for all t ∈ H.frequent() do
13:        a⃗ ← a⃗ ∪ get_embedding(t)
14:    end for
15:    I_N.insert(MinHash(Q(a)))
16:    I_V.insert(MinHash(T(a)))
17:    I_E.insert(RandomProjections(a⃗))
18:    I_F.insert(MinHash(R(a)))
19: end function
```

are frequently occurring, they are considered weak signal carriers of value–level similarity, i.e., not part of $T(a)$. However, such terms are indicative of the possible domain–specific types from which the attribute extent is drawn, viz., *Address* in this case. Therefore, they are the terms for which word-embedding vectors are sought, i.e, get_embedding(t).

- get_regex_string(v): The $r$set $R(a)$ of an attribute value builds on the following set of primitive lexical classes defined by regular expressions: `C` $= [A-Z][a-z]+$, `U` $= [A-Z]+$, `L` $= [a-z]+$, `N` $= [0-9]+$, `A` $= [A-Za-z0-9]+$, `P` $= [.,;:/-]+$. `P` also includes any other character not caught by previous primitive classes. Given an attribute value, we tokenize it and, for each token t, once we find its matching lexical class l, we add its denoting symbol to a string that describes the format for the value, and add that string to the set representation $R(a)$. If the same symbol appears consecutively, all occurrences but the first are replaced by '+', e.g., {NC+P+A+}. If an attribute value matches more than

one primitive class, we choose the first match, in the order enumerated above.

The set representations of related attributes are hashed into similar LSH partitions, i.e., rather than indexing full attribute names/values we index set representations, so that signals are both finer–grained and crisper. We can then define $\mathbb{N}$–, $\mathbb{V}$– and $\mathbb{F}$–relatedness in terms of Jaccard similarity of the corresponding set representations, and $\mathbb{E}$–relatedness in terms of cosine similarity of the corresponding set representations, and efficiently approximate these measures: Jaccard/cosine distance between two set representations is approximated by the bit–level similarity of their MinHash/random–projection values, which act as keys for the LSH buckets they are stored in.

3.4.4 Attribute relatedness: the numeric case

Numeric attributes are a special case in our framework. Of the four types of evidence we take into account, only names and formats provide relatedness evidence when dealing with numbers. This is because numbers cannot be analysed in terms of tokens as usefully as text can. Hence, token frequency and word–embedding vectors are not useful signals. Moreover, LSH hashing schemes are not available that can be applied to features that we are able to extract from numeric values. So, we do not index numeric values into the respective indexes. We do index them into the name– and format–related indexes even though, for numbers, formatting is less indicative of conceptual equivalence. For example, an attribute denoting the age of a person might share many values with an attribute denoting the person's weight or height and it is difficult to think of features that might provide the kind of diversity of viewpoints that we adopt for textual values. In such cases, rather than grounding relatedness decisions on their value–level similarity, we use the Kolmogorov–Smirnov (KS) statistic to decide whether the attribute extents, seen as samples, are drawn from the same distribution.

The inability to rely on the $\mathbb{V}$ and $\mathbb{E}$ indexes means that, for numeric attributes, we cannot obtain the same scale of computational savings from blocking and distance computation that we obtain for textual ones. Algorithm 3.2 describes how we characterise $\mathbb{D}$–relatedness instead. We use evidence from the $\mathbb{N}$ and $\mathbb{F}$ indexes in a decision on whether we proceed to consider the $\mathbb{D}$–relatedness or not. In addition, in Algorithm 3.2, we rely on the notion of a *subject attribute* (defined below) to contextualise the numerical value in terms of the entity of which it is a

presumed property. In Algorithm 3.2, by I_* we mean look–ups on all of $I_{\mathbb{N}}$, $I_{\mathbb{V}}$, $I_{\mathbb{E}}$, and $I_{\mathbb{F}}$, with an existential interpretation, i.e., membership in any one of them.

We only compute KS, our measure of $\mathbb{D}$–relatedness when there is sufficient evidence from indexes we already have that a and a' are related, thereby benefiting from the blocking effect they give rise to.

To define (and identify) subject attributes, we follow the approach from [VHM+11] and [DSFG+12]:

Definition 3.3. *Given a table T with two or more attributes $\{a_1, \ldots, a_n\}$, an attribute a_i is called the subject attribute of T if a_i identifies the entities the dataset is about, whereas non-subject attributes describe properties of the identified entity.*

To identify such attributes, we use the supervised learning technique in [VHM+11]. Intuitively, this approach favours leftmost non–numeric attributes with few nulls and many distinct values. As in [DSFG+12], we assume each dataset has only one subject attribute and that this attribute has non–numeric values. For example, in Figure 3.1, the subject attribute of S_1 is *Practice Name*, the subject attribute of S_2 is *Practice*, the subject attribute of S_3 is *GP*, and the subject attribute of T is *Practice*.

To validate the technique proposed in [VHM+11] for the identification of subject attributes on the data we used, we have built a classification model (invoked in get_subject_attribute) and 10–fold cross–validated it on 350 datasets from *data.gov.uk* (the data lake used in experiments) with manually identified subject attributes. The average accuracy is 89%.

3.5 Table relatedness

This section explains how we propagate the relatedness measures from attributes to the datasets they are part of.

For each related attribute, four distances are computed. In addition, if both a_i and the related attribute are numeric, there may be a distribution–based measurement (depending on the guards previously described) computed using the KS statistic, otherwise that measurement is set to 1 (i.e., maximally distant).

Given a target T with attributes $\{a_1, \ldots, a_n\}$, for each a_i we obtain its set representations and use the corresponding hashing schemes to retrieve, from

Algorithm 3.2 Computing $\mathbb{D}$-relatedness

Input: Numeric attributes a in table T and a' in table S
Output: $D_{\mathbb{D}}(a, a')$

```
1: function COMPUTED_D
2:     i ← get_subject_attribute(T)
3:     i' ← get_subject_attribute(S)
4:     if i' ∈ I_*.lookup(i) then
5:         return KS([[a]], [[a']])
6:     else if a' ∈ I_N.lookup(a) then
7:         return KS([[a]], [[a']])
8:     else if a' ∈ I_F.lookup(a) then
9:         return KS([[a]], [[a']])
10:    else
11:        return 1
12:    end if
13: end function
```

each of the four indexes, every attribute that is related to a_i, paired with the corresponding relatedness measure, estimated based on Equations 3.2 and 3.4 (i.e., its distance to a_i). This is so by virtue of the relationship between hashing schemes and similarity types. Specifically, given a target attribute a_i and an LSH index I_M, $M \in \{\mathbb{N}, \mathbb{V}, \mathbb{F}, \mathbb{E}\}$, consider $\mathcal{K}^{a_i} = \{k_1^{a_i}, \ldots, k_L^{a_i}\}$, the set of keys in index I_M that identify the buckets under which a_i has been stored. Recall that L is a preconfigured LSH parameter, together with the length of each key in $\mathcal{K}^{a_i}$. For each candidate attribute a'_j in the data lake, for which $\mathcal{K}^{a_i} \cap \mathcal{K}^{a'_j} \neq \varnothing$ (i.e., there is at least one bucket in I_M that contains both a_i and a'_j), the distance $dist(a_i, a'_j)$ is given by:

$$dist(a_i, a'_j) \approx 1 - \frac{|\mathcal{K}^{a_i} \cap \mathcal{K}^{a'_j}|}{L} \tag{3.5}$$

Intuitively, and as per Equations 3.2 and 3.4, the distance between a_i and a'_j is estimated based on the probability that at least one key of a_i collides with a key of a'_j. The higher this probability, the more similar the two attributes are, and the smaller the distance between them.

Each such distance value is a number between 0 and 1, quantifying the degree

of relatedness between the two attributes, as defined by the type of evidence used. Note that there may be many datasets in a data lake that can be considered related to a target. This is because table relatedness is possible if at least one pair of attributes (a_i, a'_j) are considered related by at least one LSH index, even though $dist(a_i, a'_j)$ may be large. To limit the number of related candidates to a target, we consider only the top–k candidate datasets returned as potentially related to that target, with k being set at search time. Consequently, our goal in this chapter becomes:

> *Given a target schema, find the top–k most related datasets from a data lake that are relevant for populating as many target attributes as possible.*

The statement above implies that the list of all potentially related datasets with the target have to be ranked and the top–k picked as the solution. This, in turn, implies that the relatedness between the target and a candidate datasets has to be aggregated into a single value. Note that, so far, we have described how relatedness at attribute–level is quantified, for each type of evidence used. We now describe how the relatedness between attributes is propagated to dataset–level. To this end, we propose two aggregation schemes to quantify the relatedness between a target T and a source S, as a single measure, irrespective of how many pairs of related attributes exist between them.

Consider again the example in Figure 3.1. As mentioned, for each target attribute we retrieve similar in-lake attributes using the indexes. We group the results by the dataset the attributes originate from. As an example of the structures that are created through this grouping (one for each dataset that has at least one attribute that is related to some attribute of the target), consider Table 3.1. Here, for simplicity, we use hypothetical distance values (the exact ones can be obtained by applying Equation 3.5) to exemplify the degree of similarity between attribute pairs. The table contains three rows because, of the five attributes in the target T in Figure 3.1, only three attributes in the S_2 dataset are in any degree related to it. We observe that *(T.Practice, S_2.Practice)* and *(T.City, S_2.City)* have identical attribute names so $D_{\mathbb{N}}$ is 0. For all three pairs in the table, we have $D_{\mathbb{V}}$ and $D_{\mathbb{E}}$ smaller than 1, which means that there is evidence of their $\mathbb{V}$- and $\mathbb{E}$-relatedness, and the distribution distance $D_{\mathbb{D}}$ equal to 1, since all three pairs contain attributes with textual values.

Table 3.1: Example distances for Figure 3.1

Pair	$D_{\mathbb{N}}$	$D_{\mathbb{V}}$	$D_{\mathbb{F}}$	$D_{\mathbb{E}}$	$D_{\mathbb{D}}$
$(T.Practice, S_2.Practice)$	0.0	0.9	0.6	0.2	1.0
$(T.City, S_2.City)$	0.0	0.2	0.2	0.3	1.0
$(T.Postcode, S_2.Postcode)$	0.0	0.6	0.1	0.8	1.0

3.5.1 Type–specific aggregation scheme

We now describe how each relatedness distance D_t, $t \in \{\mathbb{N}, \mathbb{V}, \mathbb{F}, \mathbb{E}, \mathbb{D}\}$, of two data sets S and T is computed from the distances between their related attributes. Intuitively, we want to aggregate, column–wise, the distances that appear in the cells of a table like Table 3.1 to obtain a 5–dimensional vector that captures the relatedness between the two corresponding datasets (i.e., T and $S2$ from Figure 3.1, in this case). We aggregate using a weighted average of the relatedness distances $\mathcal{D} = \{D_{\mathbb{N}}, D_{\mathbb{V}}, D_{\mathbb{F}}, D_{\mathbb{E}}, D_{\mathbb{D}}\}$ to obtain the desired 5–dimensional vector.

More formally, let T and S be the target and source tables from which Table 3.1 is constructed. We use Equation 3.6 on each column of Table 3.1 to aggregate its values:

$$D_t(T, S) = \frac{\sum_{i=1}^{m} w_t^i D_t^i}{\sum_{i=1}^{m} w_t^i} \tag{3.6}$$

with m, the number of rows in Table 3.1, i.e., the number of attributes in T that are related to some attribute in S, and $t \in \{\mathbb{N}, \mathbb{V}, \mathbb{F}, \mathbb{E}, \mathbb{D}\}$.

We must define the weights to use in Equation 3.6. Recall that, for each distance type t, by performing a look–up on the corresponding index for a target attribute a, we retrieve every attribute of datasets in the lake that is related to a, paired with the corresponding relatedness measure, i.e., its distance to a computed with Equation 3.5. In other words, for each target attribute a, we can compute a distribution of relatedness measurements of type t, i.e., the set of all distances of type t between a and every attribute in the lake that is related to a. We denote such a set as $\mathcal{R}_t$. Given a distance value D_t^i between two attributes, e.g., a cell value from Table 3.1, its associated weight w_t^i is given by the complementary cumulative distribution function evaluated at D_t^i:

$$w_t^i = 1 - Prob[d \leq D_t^i], \forall d \in \mathcal{R}_t \tag{3.7}$$

Intuitively, each weight w_t^i represents the probability that an observed distance D_t^i is the smallest in $\mathcal{R}_t$. This allows the weights to compensate for the presence of a potentially high number of weakly related attributes to a target attribute: attributes far away from the target attribute will have a small weight.

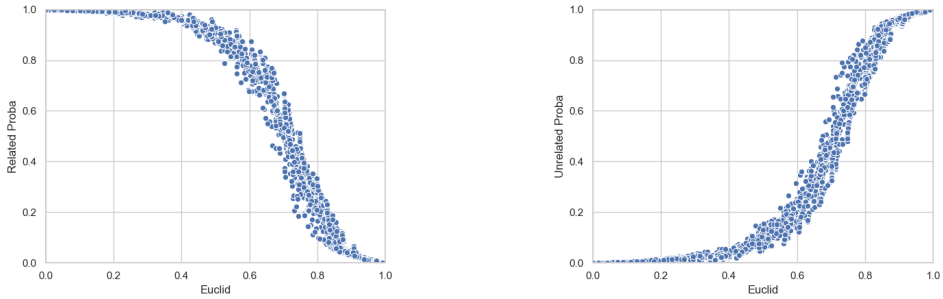
As an example, consider again the pair of datasets (T, S_2) from Figure 3.1, with their aligned attributes shown in Table 3.1. For each $t \in \{\mathbb{N}, \mathbb{V}, \mathbb{F}, \mathbb{E}, \mathbb{D}\}$, we use the distribution, $\mathcal{R}_t$, of all computed distances of type t between the target attribute and all other t–related attributes in the lake, to decide how important a given distance D_t^i is in Equation 3.6. For instance, if $S_2.Postcode$ is among the most $\mathbb{V}$–related attributes to $T.Postcode$ in the entire data lake, $D_{\mathbb{V}}^3$ (i.e., the third value on $D_{\mathbb{V}}$ column of Table 3.1) will have a high weight in Equation 3.6, denoting a strong relatedness signal, relative to all other attributes $\mathbb{V}$–related to $T.Postcode$. Conversely, if $S_2.Postcode$ is among the least $\mathbb{E}$–related attributes to $T.Postcode$ in the entire data lake, $D_{\mathbb{E}}^3$ (i.e., the third value on $D_{\mathbb{E}}$ column of Table 3.1) will have a low weight in Equation 3.6, denoting a weak relatedness signal, relative to all other $\mathbb{E}$–related attributes to $T.Postcode$.

3.5.2 Dataset–level aggregation scheme

Equation 3.6 is applied on each column of a table like Table 3.1, and this results in a 5–dimensional vector, $\vec{dv}_{(T,S)} = [D_{\mathbb{N}}(T,S), D_{\mathbb{V}}(T,S), D_{\mathbb{F}}(T,S), D_{\mathbb{E}}(T,S), D_{\mathbb{D}}(T,S)]$. In order to derive a scalar value from $\vec{dv}_{(T,S)}$ that can stand as a measurement of the relatedness between T and S, we consider S to be a point in a 5–dimensional Euclidean space, where each distance measure represents a different dimension. In this space, the coordinates of T are $[0,0,0,0,0]$. This allows us to compute a combined distance of S from T using the weighted l^2–norm of $\vec{dv}_{(T,S)}$ (i.e., the weighted Euclidean distance):

$$D(T,S) = \sqrt{\frac{\sum_{t=1}^{5}(w_t \times \vec{dv}_{(T,S)}[t])^2}{\sum_{t=1}^{5} w_t}} \tag{3.8}$$

Again, we must define the weights to use in Equation 3.8. Note that here the weights represent a proposal as to the relative importance of each evidence type t, i.e., each type of relatedness measure.



(a) Related correlation (b) Unrelated correlation

Figure 3.3: Illustration of the existing correlation between Equation 3.8 (i.e., Euclidean distance) and the probability of two tables being related.

We have set the weights using an empirical procedure: we started by construing relatedness discovery as a binary classification problem defined on two data repositories, one with real–world data from `data.gov.uk`, and one with synthetic data used in [NZPM18]. Both these repositories are analysed in Section 3.7 of this chapter. The procedure we have followed to determine the weights to be used in Equation 3.8 was:

1. We used the benchmark provided in [NZPM18], which comes with the ground truth about relatedness, to create a training set by choosing related and unrelated pairs of the form (T, S) (i.e., positive and negative examples, resp.) from the benchmark ground truth. In the training set, if S is related to T, then we label the pair as *related* (i.e., 1), otherwise we label it as *unrelated* (i.e., 0). Each such pair has a feature vector of five associated elements, i.e., the five distance measures obtained through Equation 3.6.

2. We built a logistic regression classifier using the training set, relying on *coordinate descent* [HCL$^+$08] to optimise the coefficient of each feature. We tested the resulting model against a test set, created similarly to the training set, using data from a ground truth of a manually created real–world benchmark, and obtained an accuracy of approx. 89%. Details about the test benchmark are given in Section 3.7.

3. We used the coefficients of the resulting model as the respective weights in Equation 3.8.

The same procedure was repeated for a data repository of real–word data, for which the ground truth has been created manually, by a human. In both cases,

the most important distance type proved to be the one based on instance values, followed by the name–based distance, the embeddings–based distance, and the format–based distance. In practice, we have observed that using either set of weights with Equation 3.8 returns similar results.

The intuition behind the use of this approach to determine suitable weight values is that the coefficients obtained through optimisation will minimise the distance between highly–related datasets and maximise it between unrelated datasets. Figure 3.3 confirms our hypothesis and shows that, for $10,000$ pairs of tables (T, S) from the ground truth of the data repository used in [NZPM18]: (a) the weighted Euclidean distance between T and S, computed using Equation 3.8, is negatively correlated with the probability of S *being related* to T, as returned by the trained model, and (b) the weighted Euclidean distance between T and S, computed using Equation 3.8, is positively correlated with the probability of S *not being related* to T, computed by subtracting the probability of the two datasets being related from 1. In other words, when the probability of two datasets being related is high, the distance between the two datasets returned by Equation 3.8 is small, and when the same probability is low, i.e., the probability of the two datasets not being related is high, the distance between the two datasets is large.

Given a target table T to be populated, and a data repository $\mathcal{S} = \{S_1, S_2, ..., S_n\}$, the *dataset discovery* problem is the problem of finding the k-most related datasets to T in $\mathcal{S}$, where dataset relatedness is measured using Equation 3.8.

3.6 Extending relatedness through join paths

The techniques described so far construe relatedness discovery as finding datasets in the lake with attributes that are aligned (by which we mean *related by any of the evidence types*) to as many attributes in the target as possible. In this section, we show how some of the indexes we build for characterising similarity can be used to discover join opportunities between the k–most related tables to a target and non–top–k tables. Thus, tables with weaker relatedness signal are included in the solution if, through joins, they contribute to covering more attributes in the target. More formally, given a target T, let $\mathcal{S} = \{S_1, \dots S_n\}$ be the set of all datasets from a data lake, and $\mathcal{S}^k, k \leq n$, the k–most related datasets to T. In this section, we describe how to identify datasets in $\mathcal{S} - \mathcal{S}^k$ that, through joins with datasets in $\mathcal{S}^k$, contribute to populating T.

We focus on joins based on postulated (possibly partial) inclusion dependencies. Although these can be computed using data profiling techniques [PBF+15], this is not practical given the size of the data lakes we are focusing on, i.e., they adopt and all-against-all comparison strategy. Thus, we consider two datasets S and S' to be *joinable* if they are *SA–joinable* (*SA* for subject–attribute, as per Definition 3.3).

Definition 3.4. *Two datasets S and S' are SA–joinable if $\exists$ a, an attribute from S, and $\exists$ a', an attribute from S', such that:*

- *at least one of a or a' is a subject attribute, and*
- *there is $I_{\mathbb{V}}$-based evidence that the tsets $T(a)$ and $T(a')$ overlap.*

Thus, we rely on $I_{\mathbb{V}}$ to identify inclusion dependencies and, instead of using candidate keys, we rely on subject attributes.

To determine whether two *t*sets overlap, we consider the *overlap coefficient* between two *t*sets as follows:

$$ov(T(a), T(a')) = \frac{|T(a) \cap T(a')|}{\min(|T(a)|, |T(a')|)} \tag{3.9}$$

Let τ be a similarity threshold computed based on the parameters K and L of MinHash–based LSH, as defined in [RLU14]: $\tau = (\frac{1}{L})^{\frac{1}{K}}$. If a and a' are $\mathbb{V}$–related, given the properties of LSH under MinHash, then they are Jaccard–similar with a similarity between their respective *t*sets of at least τ. Then, by the set–theoretic inclusion–exclusion principle, it follows that

$$\frac{\tau \times (|T(a)| + |T(a')|)}{(1+\tau) \times \min(|T(a)|, |T(a')|)} \leq ov(T(a), T(a')) \leq 1 \tag{3.10}$$

We construe the discovery of join paths as a *graph traversal* problem, and, in order to identify *SA–join* paths among the elements of $\mathcal{S}$, we define an *SA-join graph*, $G_{\mathcal{S}} = (\mathcal{S}, \mathcal{I})$ over the entire data lake, where $\mathcal{S}$ is the node set and $\mathcal{I}$ the edge set defined using the two *SA–joinability* conditions from Definition 3.4: each edge from $\mathcal{I}$ connects two *SA–joinable* nodes S_i and S_j.

Given an *SA–join* graph $G_{\mathcal{S}}$ and a set $\mathcal{S}^k$ of k–most related datasets to a given target T, we find the set of *SA–join* paths from each $S \in \mathcal{S}^k$ to all other vertices in $G_{\mathcal{S}}$ (or in a connected component of $G_{\mathcal{S}}$ that contains S) that are not in $\mathcal{S}^k$, using Algorithm 3.3. Specifically, given datasets $S \in \mathcal{S}^k$, the function traverses $G_{\mathcal{S}}$ depth–first, starting from S. A join path is added to a globally accessible set of join paths $\mathcal{J}$ when (i) all its constituent nodes, apart from the starting node S, are not in $\mathcal{S}^k$, (ii) the path is not cyclic, and (iii) there is evidence from at least one index that every node in the path is related to the target.

Algorithm 3.3 is called for each $S \in \mathcal{S}^k$ and returns a set of *SA–join* paths, each of which starts from S. Each dataset in such a join path has the potential to improve target population, either through the addition of new instance values to an already covered target attribute, or by populating previously uncovered attributes. In the next section we show that, by taking join opportunities into account, both the achievable ratio of covered target attributes and the precision of attributes that are considered for populating the target are improved.

3.7 Dataset discovery evaluation

In this section, we firstly describe the data repositories used to evaluate our proposed technique (referred to as D^3L for Dataset Discovery in Data lakes) and the measures we report in evaluating it. Then, we evaluate the effectiveness of each of the relatedness evidence types and compare them against their aggregation. We then compare the effectiveness and the efficiency of D^3L with that of the techniques proposed in [NZPM18] (referred to as TUS for Table Union Search) and in [FAK+18] (referred to as *Aurum*). Finally, we evaluate the impact on target coverage and attribute precision when, in addition to the top–k datasets, we also consider datasets that are joinable with the ones in the top–k.

3.7.1 Data repositories used in evaluation

We use the following repositories in the experiments:

- *Synthetic* (∼1.1GB): ∼5,000 tables (used in TUS [NZPM18]) synthetically derived from 32 base tables containing Canadian open government data using random projections and selections on the base tables. We use this repository to measure comparative effectiveness in terms of precision and

Algorithm 3.3 Join paths discovery

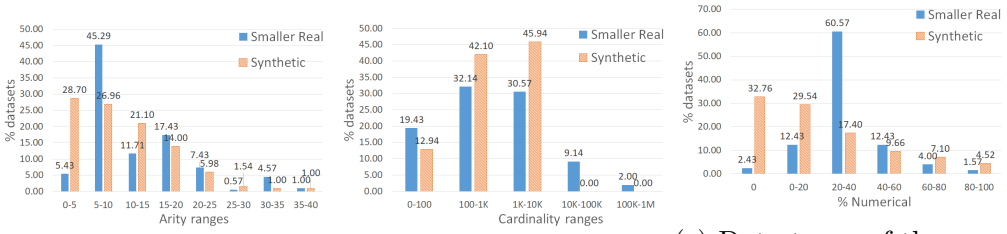
```
1: function FindJoinPath(start, path)
2:     path.append(start)
3:     for all N_i ∈ G_S.neighbours(start) do
4:         if N_i ∉ S^k and N_i ∉ path and N_i ∈ I_*.lookup(T) then
5:             new_path ← FindJoinPath(N_i, path)
6:             J ← J ∪ {new_path}
7:         end if
8:     end for
9:     return path
10: end function
```

recall. The average answer size is 260 (i.e., the average number of related tables over 100 randomly picked targets). This dataset is available at `github.com/RJMillerLab/table-union-search-benchmark.git`.

- *Smaller Real* (∼600MB): ∼700 tables from real world UK open government data, with information on domains such as business, health, transportation, public service, etc. Again, we use this repository to measure comparative effectiveness. The average answer size is 110.
- *Larger Real* (∼12GB): ∼43,000 tables with real world information from different UK National Health Service organisations (`webarchive.nationalarchives.gov.uk/search/`). We only use this repository to measure comparative efficiency. The scripts used to download the two real world data sets are available at `github.com/alex-bogatu/DataSpiders.git`.

For *Synthetic* the ground truth has been obtained by recording through the derivation procedure, for every table, which other tables are related to it. For *Smaller Real* a human has manually recorded, for every table in the lake, which other tables are related to it, as defined in Definition 3.1. In both ground truth instances each table T in the repository is listed with all its attributes along with every table T', along with its own attributes that are related to some attribute in T. As per Definition 3.1, two attributes are considered related in the ground truth if both contain values drawn from the same domain.

Figure 3.4 describes the arity, the cardinality and the percentage of numerical attributes of the two repositories used in measuring effectiveness. Arity can have a significant impact on the top–k ranking because sources with many similar attributes tend to be ranked higher by our weighted scheme, and on target



(a) Arity of the constituent tables.

(b) Cardinality of the constituent tables.

(c) Data types of the attributes in the constituent tables

Figure 3.4: Different statistics measured on the constituent tables of the *Synthetic* and *Smaller Real* repositories.

coverage because it is influenced by the number of attributes related to some target attribute. Cardinality influences the accuracy of similarity estimation and of join path discovery because a high overlap between instance values can determine a high probability of collisions between MinHash hashes. Lastly, numerical attributes are an important special case, as discussed in Section 3.4.4.

3.7.2 Baselines and reported measures

TUS [NZPM18] proposes a unionability measuring framework that builds on top of three types of evidence extracted exclusively from instance–values, aiming to inform decisions on unionability between datasets from different viewpoints. *TUS* uses similar indexing and querying models to D^3L and, therefore, it is a good candidate for a comparative analysis against D^3L w.r.t. both effectiveness and efficiency.

Aurum [FAK+18] uses both schema– and instance–level information to identify different relationships between attributes of a data lake. A two–step process profiles and indexes the data, creating a graph structure that can be used for key–word search, unionability, or joinability discovery. This makes *Aurum* a good candidate for a comparative analysis against D^3L w.r.t. indexing time, effectiveness, and the added value of join paths. Conversely, *Aurum* employs a different querying model, treating queries as graph traversal problems, rather than LSH index lookups. This means that the discovery process in *Aurum* is not influenced by the same parameters, e.g., k, as in D^3L. This makes an efficiency comparison between *Aurum* and D^3L w.r.t. search time infeasible.

In our experiments we measure the following characteristics:

1. **Individual effectiveness**. We measure how effective each type of index is in identifying related datasets. To this end, we report the *Precision* and the *Recall* at k (for a range of k values) in discovering the top–k most related datasets to a randomly picked target.

2. **Comparative effectiveness**. We measure how effective D^3L is, using all five types of relatedness evidence, against TUS and $Aurum$. To this end, we report the *Precision* and the *Recall* at k (for a range of k values) in discovering the top–k most related datasets using D^3L, TUS, and $Aurum$.

3. **Comparative efficiency**. We measure (a) the time taken to create the indexes for D^3L, TUS, and $Aurum$; (b) the time necessary to query the indexes and compute the top–k solution for D^3L and TUS (for reasons mentioned above), and (c) the space overhead incurred for all three solutions.

4. **Join impact**. We measure the impact of using join opportunities in performing dataset discovery using each of the three solutions, relative to (a) a measure of target coverage defined as the ratio of target attributes covered by at least one dataset from the result, and (b) to a measure of target attribute precision defined as the percentage of candidate attributes correctly covering some target attribute.

For the purposes of computing precision and recall for (**1**) and (**2**) above, we define:

- **TP**. A table from repository R that is in the top-k tables returned and is related to the target in the ground truth for R is a true positive;

- **FP**. A table from repository R that is in the top-k tables returned and is not related to the target in the ground truth for R is a false positive;

- **FN**. A table from repository R that is related to the target in the ground truth for R but is not a member of the top–k tables returned is a false negative.

As usual, precision p is defined by:

$$p = \frac{TP}{(TP + FP)}$$

Recall r is defined by:

$$r = \frac{TP}{(TP + FN)}$$

In assessing the result (i.e., the top–k tables returned), we count the occurrence of a table in the answer as a true positive if, as per the corresponding ground truth, at least one, but not necessarily all, attributes of a table in the solution is related to the target. We apply this interpretation to both D^3L and TUS.

In our interpretation of true positives, we consider that failing to identify one related attribute should not be considered a sufficient condition for concluding that the table it belongs to is unrelated to the target: every attribute that can contribute to populating the target does indeed so contribute. We present more insight on the coverage of the target in the experiments pertaining to relatedness as joinability.

For the purposes of computing the target coverage for (**4**) above, firstly, let $\mathcal{S}^k = \{S_1, \ldots S_k\}$ be the k-most related datasets to a given target T. Given a dataset $S_i \in \mathcal{S}^k$, let $\mathcal{J}_{S_i}$ be the set of all join paths J_l that start from S_i. We denote the arity of T by $arity(T)$ and the projection from S_i of the attributes that are related to some attribute in T by $\pi_{related(T)}(S_i)$. Similarly, we denote the projection from the result of the join path $\mathcal{J}_{S_i}$ of the attributes that are related to some attribute in T by $\pi_{related(T)}(\mathcal{J}_{S_i})$.

We define the coverage of S_i on T as the ratio of attributes in T that are related to some attribute in S_i:

$$cov_{S_i}(T) = \frac{arity(\pi_{related(T)}(S_i))}{arity(T)} \tag{3.11}$$

Note that the coverage of a join path $J_l \in \mathcal{J}_{S_i}$ can be defined in a similar way by replacing S_i with the result of the join. For the purpose of our comparison though, we are interested in the combined coverage of all the join path results in $\mathcal{J}_{S_i}$, since each join path can contribute new attributes to the target. As such, we define the coverage of $\mathcal{J}_{S_i}$ on T as:

$$cov_{\mathcal{J}_{S_i}}(T) = \frac{arity(\bigcup_{J_l \in \mathcal{J}_{S_i}} \pi_{related(T)}(J_l))}{arity(T)} \tag{3.12}$$

Given Equations 3.11 and 3.12, we average the coverage measures of all $S_i \in \mathcal{S}^k$, and use the resulting measures for different k, to show how the target coverage increases when we consider datasets in join paths.

For the purpose of computing attribute precision for (**4**) above, we count an alignment between an attribute of a dataset S_i and a target attribute as a true positive if, as per the ground truth, the two attributes are related (by Definition 3.1), and as a false positive otherwise. Correspondingly, we extend this definition for a set of join paths $\mathcal{J}_{S_i}$. Firstly, we find the set of all attributes of datasets in $\mathcal{J}_{S_i}$ that are aligned with the same target attribute, and count this set as a true positive if it contains at least one element that is related with that target attribute in the ground truth, and as a false positive otherwise. As before, for both cases, we report the average attribute precision of all S_i in $\mathcal{S}^k$, for different k.

All experiments have been run on Ubuntu 16.04.1 LTS, on a 3.40 GHz Intel Core i7-6700 CPU and 16 GB RAM machine.

In the results below, each point is the average computed by running D^3L (and TUS, where pertinent) over 100 randomly selected targets in the respective repository.

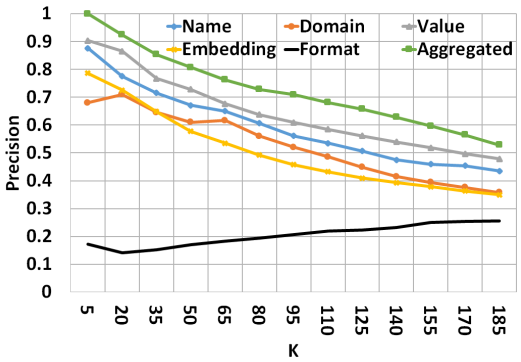
Each solution, *viz.* D^3L, TUS, and *Aurum*, is implemented using LSH Forest[BCG05], configured with a threshold of 0.7 and a MinHash size of 256.

3.7.3 Individual effectiveness

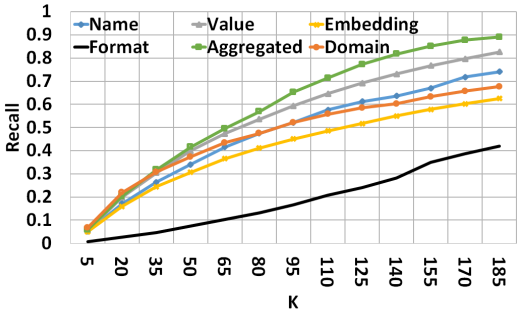
We first evaluate the effectiveness of the discovery process conducted using each evidence type individually. We only discuss here the results obtained for the *Smaller Real* repository; running the experiment on the *Synthetic* repository returned similar behaviour in terms of precision and recall.

Experiment 3.1. ***Smaller Real: Individual precision and recall***

The purpose of this experiment is to evaluate the effectiveness of individual evidence types against what is achievable when using the combined approach. In Figure 3.5, the low precision (e.g., $[0.10, 0.30]$) and recall (e.g., $[0.03, 0.43]$) achieved using *format* suggests that the format representation in itself is not sufficiently discriminating, e.g., there may be many single–word or number attributes



(a) Precision



(b) Recall

Figure 3.5: Average individual precision and recall measured on the results of 100 dataset discovery processes with targets randomly selected from *Smaller Real*.

that represent different entities. The remaining evidence types yield higher precision: at the average answer size, $k = 110$, all four evidence types achieve between 0.43 (*embeddings*) and 0.60 (*values*) precision, and between 0.49 (*embeddings*) and 0.70 (*values*) recall.

Aggregating all five measures using the aggregation framework described in Section 3.5 results in a nearly constant increase in both precision and recall, compared with the best individual evidence type: *values*. For instance, at $k = 110$, the 60% precision achieved when using *value*-based similarity is increased to almost 70% when considering all evidence types. Similarly, recall increases from 65% when using values to more than 70% when combining all measurements. Overall, there is a 29% increase in the percentage of correct values returned at $k = 110$, which explains the increase in both precision and recall when all relatedness evidence types are considered.

Performing the same experiment for non–numerical attributes only, i.e., $D_{\mathbb{D}} = 1$, resulted in an average decrease in the aggregated precision and recall of less than 3.5% each. This suggests that, for this benchmark, most of the discoverable related numerical attributes are already identified by other types of evidence, e.g., $\mathbb{N}$, $\mathbb{F}$.

3.7.4 Comparative effectiveness

In the experiments below, we report the precision and recall of D^3L, *TUS*, and *Aurum* at k (i.e., computed over the top-k tables returned) on the *Synthetic* and *Smaller Real* repositories w.r.t. to their respective ground truth.

Experiment 3.2. ***Synthetic: Precision and recall as answer size grows***

The purpose of this experiment is to compare the effectiveness of D^3L and the two baselines when the repository contains "*well–behaved*" data , i.e., since the data has been generated through random projections and selections starting from the same base tables, there is no variation in the representation of attribute names or instance values.

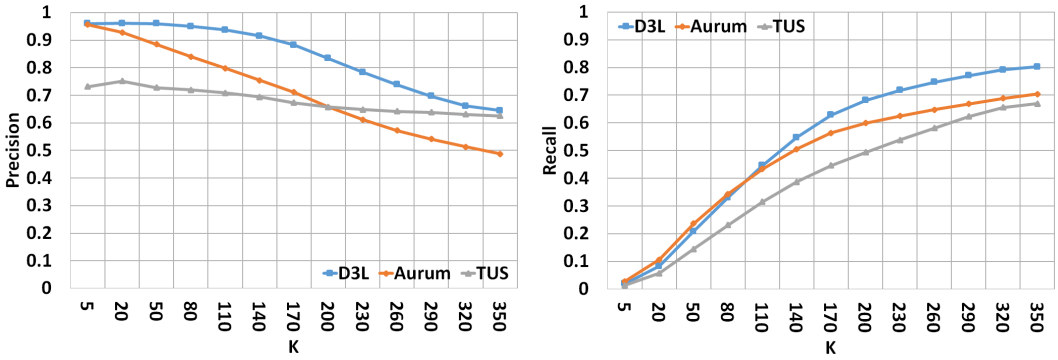
The results are shown in Figure 3.6. D^3L proves to be highly precise for $k \in [5, 140]$ and to linearly decrease in the second part of the interval (down to 0.65 when $k = 350$). This suggests that most of the closely related datasets are at the top of the ranking. Similarly, *Aurum* is comparatively precise for $k \in [5, 50]$ but degrades linearly for the rest of the interval (down to 0.49 when $k = 350$). TUS precision suggests that between 20% and 30% of the retrieved results are false positives consistently ranked higher than truly related tables.

Overall, D^3L performs better than the baselines because the finer–grained features are more diagnostic of similarity and the aggregation framework allows each evidence to contribute to the ranking, therefore reducing the impact of highly–scored false positives, i.e., a strong score in one dimension is balanced with a, potentially, lower score in another. By contrast, both baselines employ a *max–score* aggregation that only considers the highest similarity score. In case of TUS, the transformation of similarity scores into probabilities determines a further dispersion of true positives across the entire set of results.

Recall rises fast for $k \in [5, 140]$ for all approaches and levels out beyond the average answer size. As k increases, D^3L is able to identify up to 20% more relevant tables compared to TUS, and up to 10% more relevant tables compared to *Aurum*. This is because D^3L employs a multi–evidence relatedness discovery that guards against too many misses. We found that both TUS and *Aurum* tend to miss relevant attributes that do not share values with some target attribute. This is because TUS relies exclusively on instance values evidence, and *Aurum*'s name and TF/IDF–based evidence proves less dependable than content–based evidence.

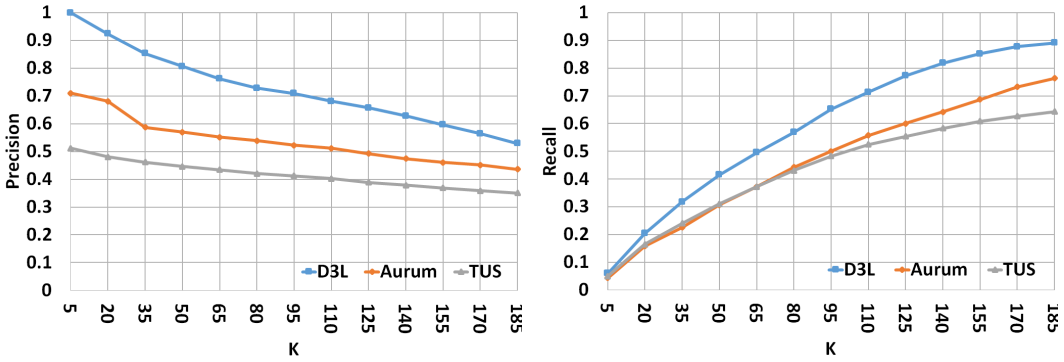
Experiment 3.3. ***Smaller Real: Precision and recall as answer size grows***

The purpose of this experiment is to compare the effectiveness of D^3L, TUS, and *Aurum* when the data repository contains less "*well–behaved*" data , i.e., since



(a) Precision (b) Recall

Figure 3.6: Average precision and recall measured on the results of 100 dataset discovery processes with targets randomly selected from *Synthetic*.



(a) Precision (b) Recall

Figure 3.7: Average precision and recall measured on the results of 100 dataset discovery processes with targets randomly selected from *Smaller Real*.

the the repository contains real world data from `data.gov.uk`, there are variations in the representation of attribute names and instance values that represent the same entity.

The results are shown in Figure 3.7. D^3L correctly identifies highly related datasets, e.g., $k \in [5, 110]$, resulting in precision between 0.2 and 0.4 higher compared to *TUS*, and between 0.05 and 0.3 higher compared to *Aurum*. This is because the value–based similarity evidence used by *TUS* and *Aurum* expect equality between the instance values of similar attributes, which is not a characteristic of *Smaller Real*. As with the *Synthetic* benchmark, the aggregation framework of D^3L also contributes to the improved precision.

Regarding recall, at the average answer size of $k = 110$, D^3L identifies more than 70% of the related datasets, while both *TUS* and *Aurum* identify around

55%. In fact, the performance gap is wider between D^3L and the two baselines for *Smaller Real* than for *Synthetic* across the entire range of k values. This could be explained by the fact that D^3L employs a more lenient approach w.r.t. format representation of values when indexing and comparing attributes. In contrast, TUS and *Aurum* are more dependent on consistent, clean values than D^3L. High levels of consistency and cleanliness are features of the synthetically generated tables but are less prevalent in real–world data.

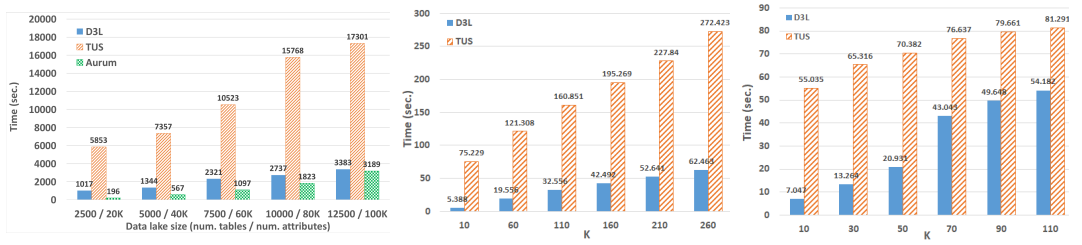
3.7.5 Comparative efficiency

In these experiments, We report performance data for D^3L, TUS, and *Aurm*, where pertinent. We report the time it takes to create the indexes and the time it takes to compute the top-k solution. Note that the implementation of TUS in [NZPM18] is not publicly available so we have implemented it ourselves using information from the paper. For *Aurum*, we have used the implementation from `github.com/mitdbg/aurum-datadiscovery`

Experiment 3.4. ***Time to create the indexes as the data lake size grows***

The results are shown in Figure 3.8a. For each system, the reported values include the times required for preprocessing the data and for creating all data structures later used in performing dataset discovery.

Compared to TUS, D^3L performed up to 4x better on small and medium sized lakes, e.g., $7.5K$ tables, and up to 6x better on larger ones, e.g., $12.5K$. *Aurum* performs up to 5x better that D^3L for small data lakes, e.g., $2.5K$ tables, and comparable with D^3L for larger lakes, e.g., $12.5K$. The dominant task in both D^3L and TUS is data pre–processing, e.g., generating summary representations for each attribute, while in *Aurum* the dominant task is the creation of the graph structure used to perform discovery. The common tasks of generating *MinHash*/random–projection signatures and creating LSH indexes have been found to take comparable amounts of time in all three systems. The main reason for the poorer performance of TUS seems to lie in its approach to semantic evidence, for which, in [NZPM18], YAGO [SKW07] is used. Having to map each token of each instance value into a YAGO knowledge base significantly slows down index construction and, as the effectiveness results have shown, for perhaps insufficient return on investment.



(a) Indexing time (b) *Synthetic* search (c) *Smaller real* search

Figure 3.8: Average LSH indexing and searching performance measured on the results of 100 dataset discovery processes with targets randomly selected from *Synthetic* or *Smaller Real*.

Experiment 3.5. ***Synthetic: Effect on search time as answer size grows***

In the next two experiments, we report the effect of the answer size on the time needed to compute the answer. The requested size of the answer is the parameter that most significantly affects the search time for D^3L and TUS. Conversely, the *Aurum* query model is not impacted by the size of the result: even when using LSH Forest, the indexes are queried only once, when the graph structure is created. In the case of D^3L and TUS, every query is an index–lookup task configured with a value for k.

Figure 3.8b shows the results for *Synthetic*. D^3L performs much better than TUS because the reliance on YAGO of the latter to provide semantic information proves to be a performance leakage point: recall that, at search time, the same process of mapping each instance value to YAGO is applied on the target. Moreover, in TUS, the index is only a blocking mechanism, i.e., there remains a significant amount of computation to be done before the unionability measurements are obtained. In contrast, D^3L does not use knowledge–base mapping and its distance–based approach means that search returns plug directly into relatedness measurements.

Although not directly comparable, we also report the average search time of *Aurum* obtained for 100 queries on *Synthetic*, with a graph structure that accommodates a result size of at least 260 datasets: 22.42 seconds.

Experiment 3.6. ***Smaller Real: Effect on search time as answer size grows***

In this experiment, we use the *Smaller Real* for which we vary the answer size

from $k = 10$ to $k = 110$ (the average answer size) growing by 20 at each step.

The results shown in Figure 3.8c tell a different story from Figure 3.8b. While D^3L still outperforms TUS, the performance gap shrinks considerably, particularly for $k > 50$. This is because *Smaller Real* contains a greater ratio of numeric values (shown in Figure 3.4c) and fewer tables overall than *Synthetic* (700 v. 5000). While D^3L spends computation time in considering numeric attributes, they are completely ignored by TUS. Thus, the performance leaks that were significant before do not occur in this case. The flip side, for TUS, is a loss of about 0.2 in both precision and recall at $k = 110$.

We also report the average search time of *Aurum* obtained for 100 queries on *Smaller real*, with a graph structure that accommodates a result size of at least 110: 18.37 seconds.

Experiment 3.7. ***Space overhead of the indexes***

In Table 3.2, we report the total space occupied by indexes, relative to the data lake size, for three repositories: *Synthetic* (1.1 GB), *Smaller Real* (600 MB), and a sample of *Larger Real* (3 GB). We used a sample of the *Larger Real* because building the TUS indexes for the full 12 GB repository requires more than 20 hours. We also report the combined space overhead of *Aurum*'s graph data structure, profile store, and LSH indexes.

For the *Synthetic* repository, TUS and *Aurum* occupy 13% less space compared to D^3L. This is because D^3L indexes four types of relatedness evidence, as opposed to only three in TUS and *Aurum*. The differences in occupied space increase for *Smaller* and *Larger Real* repositories. This is because, in addition to creating more indexes, D^3L uses finer–grained features for relatedness discovery, which results in more related attributes being discovered, which, in turn, results in more entries (buckets) per index.

3.7.6 Impact of join opportunities

In this section, we report on the impact of identifying join paths that start from some table from the top-k. Our stated motivation for searching join paths is to populate as many target attributes as possible. As such, we adopt the notions of coverage and attribute precision defined in Section 3.7.2 to compare what is achievable when we take into account join opportunities and when we do not, i.e., what is the impact of considering tables resulting from join paths on the coverage

Table 3.2: Space overhead for different repositories.

	Synthetic	*Smaller Real*	*Larger Real (sample)*
D^3L	69%	33%	58%
TUS	56%	19%	32%
$Aurum$	55%	20%	29%

of the target and on the attribute precision. Our hypothesis is that by considering join paths we can identify relevant datasets that are not part of the initial ranked solution, but can improve the target coverage.

In these experiments, we use the *Synthetic* and *Smaller Real* repositories since measuring precision requires a ground truth.

We report the target coverage and attribute precision with ($D^3L+J/Aurum+J$) and without ($D^3L/Aurum/TUS$) augmenting the top–k result with joinable datasets. Note that the graph structure built by $Aurum$ includes PK/FK candidate relationships, but TUS does not address joinability discovery.

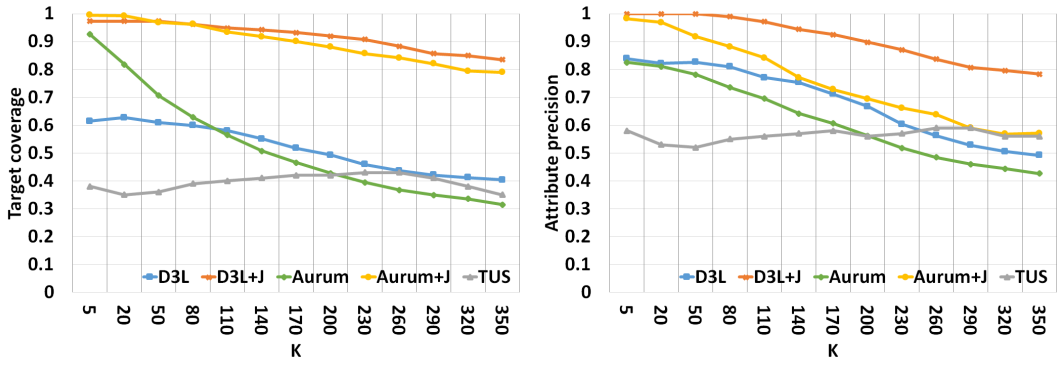
Experiment 3.8. ***Synthetic: Target coverage as answer size grows***

The purpose of the next two experiments is to compare the target coverage and the attribute precision achieved on *Synthetic*, and, therefore, on data "well–behaved".

The $D^3L + J$ and $Aurum + J$ curves from Figure 3.9a suggests that the two systems are able to cover most target attributes by following join paths. The sharp decrease in coverage when join paths are not considered confirms our hypothesis that join paths allow us to identify sources potentially far away from the target but relevant for maximizing its coverage. The superior coverage manifested by $Aurum$ for $k \in [5, 80]$ can be explained by the fact that the ranking strategy employed in $Aurum$ favours the quantity of covered target attributes, over the strength of the relatedness. In D^3L's case, the aggregation framework splits the ranking criteria between the number of covered attributes and the strength of the similarity. TUS seems to return many unrelated datasets with the given target at the top of the ranking and, therefore, is less effective in covering it.

Experiment 3.9. ***Synthetic: Attribute precision as answer size grows***

Figure 3.9b shows how many of the attributes used to populate the target are correct in each case. Attribute precision is between 85% and 100% when populating the target with the attributes returned by $D^3L + J$ and $k < 260$, in



(a) Target coverage (b) Attribute precision

Figure 3.9: Average coverage and attribute precision measured on the results of 100 dataset discovery processes with targets randomly selected from *Synthetic*.

contrast with $Aurum + j$, which decreases more sharply, i.e., a lower bound of 65% at $k = 260$. The results are consistent with the ones reported in Section 3.7.4 and are the consequences of the same characteristics: finer–grained features that are more diagnostic and multi–evidence similarity signals considered by $D^3L(+J)$. Furthermore, the join paths in $D^3L + J$ are built on more than just uniqueness of values (as is the case for $Aurum + J$), i.e., they use subject– attributes, and, therefore, they introduce fewer false positives. As before, TUS returns more unrelated tables at the top of the ranking than D^3L and $Aurm$ and, therefore, is less precise.

Experiment 3.10. ***Smaller Real: Target coverage as answer size grows***

The purpose of this experiment is to compare the target coverage and the attribute precision achieved on *Smaller Real*, and, therefore, on data less "well–behaved".

Figure 3.10a shows that both $D^3L + J$ and $Aurum + J$ achieve considerable improvements in coverage over their join–unaware variants. The increase is, as expected, smaller at low k values, e.g. $k \in [5, 20]$, because the top of the ranking already covers the target well. As k increases, the improvement in coverage becomes more significant, especially in $Aurum's$ case. This suggests, once again, that tables that are related to the target but are not included in the top-k (due to an index miss, or weak relatedness signals) can be identified by traversing join paths from some top-k datasets.

The low TUS coverage shown in Figure 3.10a suggests that the top-k solution covers only a small fraction of the target attributes. This is because the datasets

at the top of the ranked solution contain attributes aligned with approx. 25% of target attributes, while the rest (even as k increases) do not contribute many additional attributes.

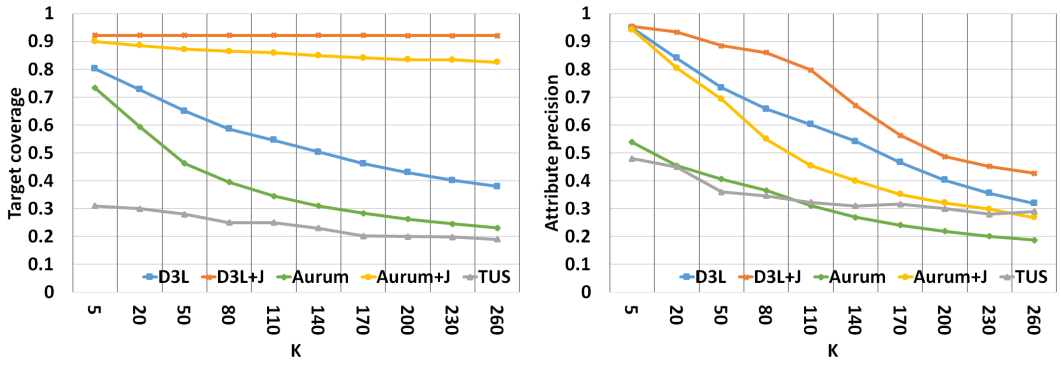
D^3L proves significantly better at covering target attributes than TUS and $Aurum$ for the entire interval of k values. This is because, as previous experiments showed, D^3L retrieves higher quality datasets (i.e., more related to the target) from the lake. The decrease of the curve as k increases can be explained by the fact that the measure is an average of individual coverage values. As k increases, there are more datasets with small coverage and the overall average decreases.

Experiment 3.11. ***Smaller Real: Attribute precision as answer size grows***

Figure 3.10b suggests that only 35% to 45%, and only 20% to 50% of the target attributes populated by TUS and $Aurum$, resp., are correct. This is not surprising since the dataset–level precision reported in Section 3.7.4 showed that at most 50%, in the case of TUS, and at most 70%, in the case of $Aurum$, of the retrieved datasets are indeed relevant for populating the target.

The increased precision of D^3L is explained by its ability to identify attribute relatedness even when the format representation of values differs. The difference is preserved when joinable tables are considered. By including tables from the join paths in the solution, at $k \in [50, 170]$, the attribute precision increases by up to 0.2. Note that, for $D^3L + J$, there is not much increase at the head and tail of the k values interval, since datasets at the top already cover the target precisely, while datasets that are joinable with tables far away from the target provide low quality attributes. Furthermore, the precision of $D^3L + J$ does not descend below the precision of D^3L, suggesting that most of the attributes contributed by the former are true positives.

Finally, as in the *Synthetic* case, the use of more restrictive conditions (i.e., the use of subject attributes, when searching for join), allows $D^3L + J$ to cover the target more precisely.



(a) Target coverage (b) Attribute precision

Figure 3.10: Average coverage and attribute precision measured on the results of 100 dataset discovery processes with targets randomly selected from *Smaller Real*.

3.8 Summary and conclusions

This chapter has focused on describing an effective and efficient solution to the problem of dataset discovery, seen as a nearest–neighbour search task. The approach employs five types of relatedness evidence with the goal of maximising effectiveness by covering a wide range of cases, and LSH techniques, for blocking and quantifying relatedness, with the goal of maximizing efficiency by identifying potential relatedness relationships in linear time. The main characteristics of D^3L are:

- The use of relatedness evidence conveyed only by the data representations themselves, without the use of external metadata or ontologies. The effectiveness experiments in Section 3.7 show that by relying only on this type of relatedness evidence, we can still effectively identify datasets related to a given target. Therefore, D^3L meets the desideratum stemming from the potential lack of metadata associated with a data lake, as stated in Section 3.1.

- The use of multiple types of relatedness features, each grounded on a different type of evidence, and the mapping of those features into a uniform distance space. The effectiveness experiments in Section 3.7 also show that, with the exception of format, each type of feature used can be effective individually. Moreover, all of them complement each other when used in tandem. Therefore, D^3L meets the desideratum for multiple types of relatedness evidence that can be reliably aggregated, as stated in Section 3.1.

- The use of fine–grained features of the data values that allow a reliable identification of relatedness in the presence of format representation heterogeneity. The effectiveness experiments in Section 3.7, especially those performed on the *Smaller Real* repository, show that D^3L is lenient w.r.t. the potential discrepancies in format heterogeneity existing in real–world data lakes. Therefore, D^3L meets the desideratum for reliable relatedness evidence, even when the values from which features are extracted are inconsistent in their textual representation, as stated in Section 3.1.

- The use of LSH indexes for identifying potentially related attributes and for estimating values for the various relatedness measures used. The efficiency experiments in Section 3.7 show that the time required to index a data lake grows linearly w.r.t. the data lake size, and that the time required to compute the top–k solution grows linearly w.r.t. the size of the expected solution. Therefore, D^3L avoids all–against–all comparisons when discovering related datasets and meets the scalability desideratum, as stated in Section 3.1.

- The use of join paths, which are discoverable using the same indexing structures, to extend the notion of relatedness. The join paths experiments in Section 3.7 show the benefit in terms of target coverage and attribute precision of including joinable tables in the discovery solution.

In summary, the results constitute compelling empirical evidence that D^3L leads to better precision and better recall in faster time than the closest state–of–the–art competitors, and that the first objective of our research, as stated in Section 1.4 has been met: (i) D^3L is able to effectively identify the most relevant input for a given target and, therefore, minimise the cost of applying data wrangling by avoiding running the wrangling process on irrelevant datasets; (ii) given the target schema, the discovery process is fully automatic so that the cost associated with the need for users with expert–level skills is reduced.

Chapter 4

Automatic format transformation

The techniques presented in Chapter 3 provide a means to contain the size of the input of a data wrangling process by directly addressing the scalability challenge introduced in Section 2.3.1. In addition, dataset discovery aims to be lenient in postulating the relatedness of datasets with heterogeneous representations of instance values by employing a granular analysis of data. However, being lenient with respect to different format representations does not solve the problem introduced by the heterogeneity challenge from Section 2.3.2. Indeed, data format transformation, the subject of this chapter, is required to normalise the representation of values, and thereby reduce inconsistencies. Thus, this chapter treats the task of format normalisation as an approach to overcome the heterogeneity challenge. We do so by construing the identification of format manipulation operations as the problem of synthesising string processing operations from input–output examples. Later, in Chapter 5, we will see that performing format normalisation at larger scales (e.g., in data lakes) introduces the new challenge of identifying suitable training data for our synthesis problem.

The aim of this chapter can be summarised as follows:

> *Given exemplar pairs of input and expected output strings, automatically synthesise string processing operations (which we call transformations) that, consistently with the examples, transform new strings whose format representation conform with the input examples into output strings whose format representation conform with the corresponding output examples.*

In this chapter, we describe a novel method for synthesising format representation transformations, independently from most characteristics of a data lake. In the next chapter, we discuss the implications of applying this chapter's techniques on large scale repositories such, as data lakes.

In Section 4.1, we motivate our choice of viewing format transformation as a program synthesis problem, and present our desiderata for the proposed method. Then, in Section 4.2, we introduce the concepts underpinning our approach, together with a brief discussion of the state–of–the–art on format transformation. The contributions of this chapter are briefly outlined in Section 4.3, and then described in detail in Sections 4.4 and 4.5. Finally, we empirically evaluate our proposal in Section 4.6, as part of a comparative analysis with the closest state–of–the–art alternative, before concluding in Section 4.7.

4.1 Motivation and desiderata

The importance of format transformation as a task in preparing data for analysis has been acknowledged since the rise of data warehouses and ETL. For example, systems such as AJAX [GFSS00] or Potter's Wheel [RH01] defined regular expression based languages for expressing transformations, with the goal of normalising the representation of string values. Later on, [KHP+11] put format transformation at the core of data wrangling and proposed Wrangler, an interactive system for authoring transformation operations. The approach to format transformation followed in Wrangler has also been adopted by industry leaders in data wrangling, e.g., Trifacta (`https://www.trifacta.com`) and Talend (`https://www.talend.com`).

Much of this focus on format transformation has been motivated by the recurrent conclusion that "janitorial" tasks, such as cleaning and reformatting data values, are the most tedious in preparing data for analysis [Data, Datb]. Such tediousness comes also with a need for expert–level knowledge due to format transformation being performed mostly through manually authored scripts consisting of string manipulation operations. Consequently, the task can be characterised as time consuming and involving advanced technical/programming knowledge. This comes in contradiction with the principles of cost–effective data wrangling defined in Section 2.3. As such, in this chapter we propose a format transformation solution with the following properties:

Simple specifications. At heart, format transformation is a domain–specific programming problem where the user has to be familiar with the data so that she can derive the logical specifications from which a transformation program is constructed. If we wish to avoid an appeal to programming experts, then we need a format transformation technique that is amenable to synthesis, e.g., based only on very simple requirement specifications, such as input–output example pairs.

Automated transformations. Transformations have to be expressed using an underlying, often domain–specific, language. An iterative or user–guided transformation process would require, in addition to knowledge about the data to be transformed, familiarity with the language as well. Therefore, we aim at a fully automated solution that does not require end–users to provide any other annotations/hints than exemplar inputs and expected outputs.

Fast convergence. In addition to reduced technical knowledge, time–related efficiency is necessary for cost–effective data wrangling. As such, we aim at an efficient solution for synthesising a transformation from few examples, as required for on–demand data wrangling scenarios.

Effective results. Effectiveness is paramount for there to be trust in an automated solution. Therefore, we aim at delivering a format transformation proposal that, whilst aiming at being efficient, achieves a level of effectiveness similar to that of the current state–of–the–art.

4.2 Background and related work

The above desiderata suggest that we aim to provide a format transformation solution that requires reduced familiarity with the input data and reduced familiarity with the language used to express the transformations. To this end, we first define the central notions and discuss the technical background upon which our approach builds. We also analyse how representative state–of–the–art alternatives fall short of fully delivering on the above requirements.

4.2.1 Format transformation: definition

In this section we formally define the notion of format transformation used in this chapter. To this end, consider a simple example string denoting a calendar date, *24/04/1989*, which we will refer to throughout this section.

We start by defining the notion of format of a string, construed as a sequence of tokens, each of which has an associated domain–independent type denoted by an element in a collection of lexical primitives such as *number*, *word*, *punctuation*, etc.. A full description of the primitives used is provided in Section 4.4. For the purpose of defining the concept of format transformation, we consider the format of the example string s =*24/04/1989* to be $\mathtt{f}_s = \mathtt{N}_d/\mathtt{N}_m/\mathtt{N}_y$, where each token represents the number denoting the day, the number denoting the month, and the number denoting the year, respectively. The '/' characters act as separators. It follows that $\mathtt{f}_s$ identifies a class of strings with the same format, e.g., dates represented consistently with $\mathtt{f}_s$. We say that a string s =*24/04/1989* is an *instantiation* of the class represented by $\mathtt{f}_s = \mathtt{N}_d/\mathtt{N}_m/\mathtt{N}_y$, and we denote such relationship by $s : \mathtt{f}_s$. We can then define a format transformation:

Definition 4.1. *Given two formats* $\mathtt{f}_{in}$ *and* $\mathtt{f}_{out}$*, and* $\mathcal{M} = \{(in, out)|in : \mathtt{f}_{in}, out : \mathtt{f}_{out}\}$*, i.e., the mapping from instantiations of* $\mathtt{f}_{in}$ *to their corresponding instantiations of* $\mathtt{f}_{out}$*, a format transformation is a function* $\mathcal{T}$ *such that* $\forall(in_i, out_i) \in \mathcal{M}$*,* $\mathcal{T}(in_i) = out_i$*.*

Intuitively, for $\mathtt{f}_{in} = \mathtt{N}_d/\mathtt{N}_m/\mathtt{N}_y$ and $\mathtt{f}_{out} = \mathtt{N}_m/\mathtt{N}_d/\mathtt{N}_y$, a transformation between the two formats would transform any calendar date that follows $\mathtt{f}_{in}$ to an equivalent date represented following $\mathtt{f}_{out}$, e.g., such a transformation can simply swap the $\mathtt{N}_d$ and $\mathtt{N}_m$ tokens of each input instantiation of $\mathtt{f}_{in}$ to obtain the corresponding representation that follows $\mathtt{f}_{out}$. The challenge, however, is to identify the sub–strings, i.e., numbers in this case, that represent the same token type in each (in, out) string pair. This is a central task in our format transformation proposal, as seen later in this chapter.

4.2.2 Format transformation through program synthesis

One of the goals of cost–effective data wrangling in both, industry and research communities, has been the ability to delegate data preparation tasks to non–specialist users, e.g., users who do not have programming skills, or who are not

particularly familiar with the characteristics of the data [FGL+16], [HHK19]. For a format transformation technique to meet this goal, a methodology is required that relieves the user from the burden of manually writing (parts of the) transformations. One approach is to use *Program Synthesis* (PS), by means of which we can automatically find a program, expressed in an underlying language, that satisfies the user intent expressed in the form of requirement specifications and/or constraints [GPS17]. While PS has a long research history, in this thesis, we only focus on characteristics that are relevant to format transformation. In this thesis, we consider a program to be represented by a collections of transformations, each transformation conforming to Definition 4.1.

At the core of PS there are three main components: (i) an underlying language that determines what is achievable in terms of transformations; (ii) a class of input specifications that define the requirements for transformations; and (iii) a synthesis algorithm, often called a program synthesizer (or, simply, a synthesizer) that searches over the space of possible programs defined by (i) to find the program that is consistent with most, if not all, of (ii). As such, the complexity of (iii) is governed by (i) and (ii). In the rest of this section, we briefly explore each of these dimensions and introduce the principles upon which we ground our proposal in later sections.

The underlying language determines the means to express the operations that the synthesizer must learn to combine in order to meet the input requirements. At the same time, the language defines the search space to be covered by the synthesizer at synthesis time, as we describe next. Languages can be imperative or functional, have restricted control structures or not, and come with diverse operator sets (e.g., [SGF13]). Others are restricted to a subset of an existing, general purpose or domain–specific programming language, or to a specifically designed domain–specific language (e.g., [Nix85], [Ang87], [Gul11], [HG11]).

Formally, a language $\mathcal{L}$ associated with a synthesis algorithm is defined by a Context Free Grammar (CFG) $G = (V, \Sigma, S, P)$, where Σ is a finite alphabet of terminal symbols, V is a finite set of non–terminal symbols, S is the starting symbol, and P is a finite set of production rules for rewriting non–terminals into a combination of other non–terminal and/or terminal symbols. Each production rule has the form $V \rightarrow (V \cup \Sigma)^*$, where the asterisk represents the Kleene star operation, i.e., zero or more.

The search space to be explored by a synthesis algorithm is determined by

the CFG that defines the underlying language, and includes all possible combinations of language–specific operators/constructs that are consistent with the input specifications (defined next). Broadly speaking, the search space can range from the space of Turing-complete programs, for very complex languages, to more restricted computational models defined by a much simpler CFG. Since covering a large search space can quickly become inefficient, the design of languages that are to be synthesised often involves a trade–off between expressiveness and the complexity of finding simple consistent hypotheses within that space. For example, to quote from [Gul10], "the underlying language of a synthesis task has to be large/expressive enough to cover the possible operations required by the input specifications, while being restrictive enough so that it is amenable to efficient search". This is an important property of synthesis programming which will inform our design of the transformation language proposed in this chapter.

The input specifications determine how the user intention, or the expected program behaviour, is conveyed to the synthesizer. Often, such specifications are expressed using formulas in some logic (e.g., first–order predicate logic) that map inputs of the program to their corresponding outputs by telling the synthesizer what the resulting program should do. For example, a logical specification that instructs a synthesizer that any input array A with n elements should be mapped to a sorted output would look like: $\forall k.(0 \leq k < n-1) \Rightarrow (A[k] \leq A[k+1])$.

Writing first–order formulas to describe complex program behaviours requires skills beyond the reach of non–specialised users. Therefore, at the expense of clarity, simpler types of specifications have been proposed that relieve the user from having to formalise the expected behaviour. For instance, it has been shown that it is possible to map natural language specifications into logical representations [ZC09]. The challenge here would be to overcome the potential ambiguity introduced by natural language. An even simpler form of specification is represented by pairs of input–output examples, which we use in our approach, where the user specifies instances of suitable inputs to the program and expected outputs. The challenge in such a case is to prevent the synthesizer from overfitting the input, thereby limiting the effectiveness of the resulting program to the examples provided at synthesis time.

For a more comprehensive discussion of program synthesis, we refer the reader to [Gul10] and [GPS17].

The search technique is the final dimension in PS and defines the manner

in which the synthesizer tries to cover all programs in the space defined by the underlying language, that are consistent with the input specifications. Often, this step also involves a ranking task that chooses the best (or the top-k best) program(s). Examples of search techniques range from brute force, where every possible program is checked against each input constraint (e.g., [BT03]), through probabilistic and logical reasoning (e.g., [KP08]), to more specialised ones based on Version Space Algebra (VSA) [Mit82] (e.g., [RGM14]) that aims to use DAG–based techniques to efficiently cover the space of possible programs, given the input constraints. VSA has been successfully used for learning shell scripts and text editing programs[LDW00], and is the search technique of choice for many of the state–of–the–art algorithms (e.g., [Gul11], [Sin16]).

4.2.3 Format transformation: state–of–the–art

We now review the main state–of–the–art solutions for performing format transformation, as part of a data wrangling process or as a standalone task. Format transformation is seen as a data cleaning task in the research literature [ACD+16], normalising and transforming data values through *pattern enforcement and transformation tools*. When it comes to automating format transformation, two approaches are prominent: Programming–by–Example (PBE) solutions and Programming–by–Demonstration (PBD) solutions.

PBE format transformation. In PBE, the user is only asked for input specifications represented by pairs of input–output strings, the first element of which is in some occurring format, and the second element conveys the expected output, i.e., is in the target format.

FlashFill [Gul11], as well as its variation BlinkFill [Sin16], are PBE–based tools for automating string processing operations, focusing on spreadsheet data. Given one or more input specifications in the form of input–output pairs, these synthesis algorithms perform a search of the space of possible programs with a view to finding those that are consistent with the given input–output pairs. These programs are then ranked based on Occam's razor principle that a simpler explanation (a program, in this case) is preferable over a more complex alternative. The size of the search space is defined by a language consisting of string processing operators (e.g., *substring*, *concatenate*). The search space is explored using Version Space Algebra techniques to decrease the cost of covering the potentially huge number of operator combinations consistent with the given examples, but even

so, these algorithms are still exponential in the number of examples and highly polynomial in the length of the examples [RGM14].

FlashFill and BlinkFill have been highly influential due to their excellent results in automating format transformation for spreadsheets. Spreadsheet processing often involves datasets of manageable sizes, a small number of examples and active user involvement in providing additional example data when needed. In contrast, in data wrangling and data analysis, format transformation is applied repetitively on large datasets from many sources, where examples are not readily available and user supervision is often impractical. In cases where diversity causes a great many examples to be needed, user interaction again becomes impractical. This provides an opportunity for taking the human factor out of the synthesis process, but a large number of examples presents a challenge for synthesis algorithms, such as FlashFill and BlinkFill, due to their high complexity. This is a motivating factor for the contributions reported in this chapter. We also use input–output examples as input specifications for performing string processing operations, but we perform a more efficient search of the space of possible programs, which allows to contend with larger collections of training examples, when available.

Other tools that build on the same theoretical principles as FlashFill and BlinkFill include FlashExtract [LG14] for extracting structured (tabular or hierarchical) data from semi–structured text/log files and Webpages, FlashRelate [BGHZ15] for extracting tabular/relational data out of semi-structured spreadsheets, and [SG16] for cleaning spreadsheet data types (e.g., date, time, name, and units). A comprehensive survey can be found in [Gul12].

Other proposals that rely on PBE include TDE [HCG+18] where the aim is to collect many predefined transformations and to build a search engine for end–users to easily reuse them. When examples are provided, the search engine is used to find the most suitable transformation consistent with the given specifications. Other proposals, such as DataXFormer [AMI+16], rely on the same principle for creating a transformation search engine that, in addition to input–output examples, uses attribute names to find relevant Web data, from which desired output values can be extracted.

PBD format transformation. In PBD [Bro93], as opposed to PBE, the user is required to provide an initial state, intermediate states, and the final state of the process. In other words, the synthesising procedure is much more interactive and assumes the user is much more knowledgeable with respect to the

underlying language and its capabilities.

Wrangler [KPHH11] is a PBD–based tool in which the input specifications are provided by the user through a user interface by highlighting values (or parts of data values) that are subject to transformations. These interactions are translated into input specifications through a technique known as predictive interaction [HHK15], and a program is synthesised by performing a user–guided exploration of the space of possible transformations, searching for the ones that are consistent with the input specifications. Intermediate input from the user is further required for refining the possible transformations, until a suitable transformation is found (chosen by the user as well). The underlying language consists of string operators, such as *move*, *replace*, operators for manipulating table columns/rows, and basic join operations.

In this chapter we describe a solution for automating the production of format transformation based on edit operations and automata theory, which we call *SynthEdit*. It uses PBE and is, therefore, similar to the other PBE–based techniques in the sense that we use the same type of input specifications, i.e., input–output examples, but differs from them in the following respects:

- *SynthEdit* uses a simple language based on edit operations that proves to be efficient to synthesise, while being expressive enough to accommodate commonly used string manipulation operations.

- *SynthEdit* uses a search method based on finite–state automata operations, which proves to be order of magnitudes more efficient than the methods used in solutions such as FlashFill.

4.3 Overview and contributions

We now present an overview of our approach to automatic format transformation, and the main contributions in this chapter.

We start by describing a representative example, Example 4.1, where data has been extracted from the Web and the task is to derive a transformation that can generate the *Target* values using the *Source* values. Moreover, such a transformation should be applicable on any string that follows a format similar to the one followed by the *Source* values.

Example 4.1. *20th century NY state governor names and years in office:*

Source	Target
Hugh Leo Carey (74-82)	Hugh L. Carey (1974-1982)
Jay Henry Lehman (33-42)	Jay H. Lehman (1933-1942)

We start by presenting the language we use to express the transformations. In defining this language, we bear in mind the trade–off characteristic to program synthesis to the effect that the language has to be simple enough to admit efficient synthesis, but still expressive enough to describe frequently required string manipulation tasks. We define our language in terms of three simple edit operations, i.e., *Insert*, *Delete*, *Substitute*, applied on substrings of the input values, which we call *tokens* (e.g., *Hugh*, *Leo*, *74*, etc.).

The input specifications are represented by pairs of the form illustrated in Example 4.1, i.e., *Source* values denote the occurring input and *Target* values denote the desired corresponding output. Each such pair is called an *example instance* and we often refer to the input and output values of one example instance as the *source* and *target* strings, respectively.

Finally, the synthesis algorithm performs a search over the space of possible edit operation combinations and returns the simplest such combination, i.e., the transformation with the fewest steps. It then moves to match each edit operation against correspondences between input (source) and output (target) tokens.

SynthEdit is the result of the following specific research contributions:

- A format transformation language over strings that is expressive enough to represent a variety of string manipulation operations, such as token substitution, deletion, insertion, as well as substring extraction and concatenation. At the same time, the language is restrictive enough to allow efficient synthesis from input–output examples.

- A synthesis algorithm that, given a collection of input–output string pairs, efficiently covers the space of possible transformations expressed in the previously mentioned language, that are consistent with the given input–output pairs, and returns the simplest such transformation. The algorithm also allows for the synthesis of conditional transformations, based on the formats followed by the input example pairs.

- An empirical evaluation showing that *SynthEdit* is significantly more efficient than the closest state–of–the–art competitor, while achieving comparable effectiveness.

4.4 Transformation language

In defining a language $\mathcal{L}$ for our transformation synthesis solution, we operate within the triangle of interrelated constraints from Figure 4.1. Focusing on one corner detracts from what is achievable in the other two. Specifically, $\mathcal{L}$ is *expressive* if it allows for the representation of various manipulation operations on strings, such as substring extraction, concatenation, etc.. $\mathcal{L}$ is *searchable* if there is a synthesis algorithm (discussed in Section 4.5) that, given a set of input specifications, can identify in reasonable time programs expressed using $\mathcal{L}$ that are consistent with the input specifications. The trade–off between these two properties has already been discussed in Section 4.2.2. Finally, $\mathcal{L}$ is *descriptive* if the operations represented using it are intuitive, i.e., close to natural language. We impose this last constraint motivated by the desire to create a transformation synthesis solution that can be improved through user feedback, if needed, and, hence, whose results can be understood by users without a strong programming background. In defining such a language, we started from, arguably, the simplest/most intuitive set of operations on strings: the set of edit operations [JM09] *Insert*, *Delete* and *Substitute*. These operations represent the core of the transformation language, which is defined next.

Note that, in this chapter, we only discuss automatic format transformation, i.e., the user is not involved beyond the provision of input–output examples. Features such as manual editing/improvement of the resulting programs (performed by non–programmers) are left for future work, but we argue that in order for this to be possible, the language has to retain its descriptive character.

4.4.1 Syntax and language elements

We use INS, DEL, SUB to denote the edit operations *Insert*, *Delete* and *Substitute*, respectively; we use ϵ to denote an empty string, $len(s)$ for the length of a string s, $s[i:j]$ for the substring of s that starts at the i–th character and ends at the j–th in s, and c to denote the c–th occurrence of a token in a string, with

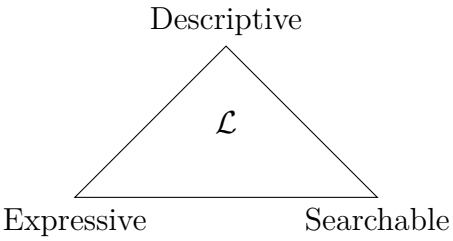


Figure 4.1: Triple of constraints that characterises the proposed transformation language.

all string positions starting from 0. Lastly, we denote the beginning and end of a string s by ^ and \$, respectively.

In Figure 4.2 s is a string (a collection of characters), i, j are string indexes/character positions, and c is a token position. The figure shows the production rules of a grammar that defines our transformation language $\mathcal{L}$. Before detailing the semantics of $\mathcal{L}$, we review its core elements.

Tokens: We view a string s as a collection of *tokens*, where each token is represented as a pair of the type r of the token and the position of the token in the string. Similarly to existing algorithms for format transformation, *SynthEdit* supports three classes of tokens: (i) regular expression tokens that match a predefined regular expression pattern; (ii) constant string tokens with a value equal to the corresponding constant string, and (iii) special tokens that mark the beginning/end of a string. The token type r is further detailed below, while the position expression is treated in the next section.

Regular expression primitives: Obtaining the set of tokens for a string s builds on a set of primitive lexical classes defined by regular expressions. Each such class denotes a token type. The regular expression primitives we use are as follows:

`N` denotes a sequence of one or more digits. The corresponding regular expression is [0-9]+.

`U` denotes a sequence of one or more uppercase letters. The corresponding regular expression is [*A*-*Z*]+.

`L` denotes a sequence of one or more lowercase letters. The corresponding regular expression is [*a*-*z*]+.

$$
\begin{aligned}
\textit{Transformation program } \mathcal{Q} &:= \mathit{Switch}((\mathtt{f}_1, \mathcal{T}_1), \ldots, (\mathtt{f}_n, \mathcal{T}_n)) \\
\textit{Format descriptor } \mathtt{f} &:= \mathtt{r}_1, \ldots, \mathtt{r}_k \\
\textit{Regex primitive } \mathtt{r} &:= \hat{} \mid \mathtt{N} \mid \mathtt{U} \mid \mathtt{L} \mid \mathtt{A} \mid \mathtt{Q} \mid \mathtt{P} \mid \mathtt{W} \mid \$ \\
\textit{Transformation } \mathcal{T} &:= \mathcal{O}_1;\ \mathcal{O}_2;\ \ldots;\ \mathcal{O}_m; \\
\textit{Edit operation } \mathcal{O} &:= \mathit{INS}(\mathcal{E}) \mid \mathit{DEL}(\mathtt{t}) \mid \mathit{SUB}(\mathtt{t}, \mathcal{E}) \\
\textit{String expression } \mathcal{E} &:= \mathit{Copy}(\mathtt{t}) \mid \mathit{Const}(s) \\
&\quad \mid \mathit{Substr}(\mathtt{t}, i, j) \mid \mathit{Concat}(\mathcal{E}_1, \ldots, \mathcal{E}_n) \\
\textit{Token } \mathtt{t} &:= (\mathtt{r}, \mathcal{P}) \\
\textit{Position expression } \mathcal{P} &:= \mathit{Pos}(\mathtt{r}_1, \mathtt{r}_2, c)
\end{aligned}
$$

Figure 4.2: Formal representation of the proposed transformation language.

`A` denotes a sequence of one or more letters. The corresponding regular expression is [*A-Za-z*]+.

`Q` denotes a sequence of one or more letters or numbers. The corresponding regular expression is [*A-Za-z*0-9]+.

`W` denotes a sequence of one or more spaces. The corresponding regular expression is *s*+.

`P` denotes a sequence of one or more punctuation signs or other special symbols. The corresponding regular expression is [.,;: /-_?!&$]+.

Format descriptors. We call a sequence of regular expression primitive identifiers, denoting the *format* of a given string, a *format descriptor*. The order of the primitives corresponds to the order of the respective tokens in the given string. For instance, going back to Example 4.1, the format of the values in the *Source* column is `A W A W A W P N P N P`, i.e., an alphabet token followed by white space, then an alphabet token, and so on. Note that the format descriptor is a type expression, so it does not include instance values.

These are the singleton elements (i.e., lexical elements in grammatical terms), of our language $\mathcal{L}$. The rest of the elements of $\mathcal{L}$ are represented by expressions (i.e., non–terminals in grammatical terms) whose semantics are given next.

4.4.2 Language semantics

In this section we review each type of expression, describing its semantics and how expressions are assembled to form a transformation program.

Position expressions: In $\mathcal{L}$, a token `t` is represented by a pair consisting of a token type `r` and a position expression $\mathcal{P}$. `r` is a regular expression primitive that matches the token value, i.e., a substring of a given string s. The semantics of a position expression $Pos(\mathtt{r_1}, \mathtt{r_2}, c)$ is to evaluate to a substring $s[j : k]$ of a given string s, such that $\exists\ i, j, k, l\ (0 \leq i < j < k < l \leq len(s) - 1)$, where $s[i : j]$ matches $\mathtt{r_1}$ and $s[k : l]$ matches $\mathtt{r_2}$. Furthermore, $s[j : k]$ is the c–th such substring in s. If such a substring does not exist then the expression evaluates to ϵ. Note that the value of c is relative to the token's neighbours and that such an expression allows us to uniquely identify each token in a given string using its neighbour tokens and, at the same time, allows for the expression to evaluate to ϵ when applied on strings with different format representations. Note that `t` only exists if the string value returned by $\mathcal{P}$ matches `r`.

String expressions: A string expression $\mathcal{E}$ has one of *Copy*, *Const*, *Substr*, or *Concat* as constructor, and takes as argument a string s or the result of another string expression. A string expression returns a new string, as follows:

> $Copy(\mathtt{t})$ evaluates to the string value of token `t`.
>
> $Const(\mathtt{s})$ evaluates to a constant string s.
>
> $Substr(\mathtt{t}, i, j)$ returns the substring that starts at position i and ends at position $j - 1$ of the string value of `t`.
>
> $Concat(\mathcal{E}_1, \ldots, \mathcal{E}_n)$ performs string concatenation on the results of applying $\mathcal{E}_1, \ldots, \mathcal{E}_n$.

Edit operations: An edit operation $\mathcal{O}$ has one of INS, DEL, or SUB as constructor, and takes as argument a token or a string expression. Edit operations perform insertion (INS), removal (DEL), or replacement (SUB) of the string resulting from applying the expressions given as parameters, on a given string s.

Transformations: A transformation $\mathcal{T}$ is a sequence of edit operations.

Transformation program: The top–level expression of a transformation program $\mathcal{Q}$ has a *Switch* constructor that, when applied, the expression effects

transformations on a input string s, conditioned by the equality between the format descriptor of s and a given format descriptor.

As a simple illustration of how all the elements of language $\mathcal{L}$ come together, consider Example 4.2. The transformation program that, as we explain in the next section, is synthesised from the two pairs of strings, *source* and *target*, is given below:

Example 4.2. *Extract the month name from the given date:*

Source	Target
28 March 2010	March
1 June 99	June

$$
\begin{aligned}
&Switch((\mathtt{f}_{source}, \mathcal{T})),\ where\\
\mathtt{f}_{source} &= \mathtt{N\ W\ A\ W\ N}\\
\mathcal{T} &= DEL((\mathtt{N}, Pos(\hat{}, \mathtt{W}, 0)));\ DEL((\mathtt{W}, Pos(\mathtt{N}, \mathtt{A}, 0)));\\
&\quad SUB((\mathtt{A}, Pos(\mathtt{W}, \mathtt{W}, 0)), Copy((\mathtt{A}, Pos(\mathtt{W}, \mathtt{W}, 0))));\\
&\quad DEL((\mathtt{W}, Pos(\mathtt{A}, \mathtt{N}, 0)));DEL((\mathtt{N}, Pos(\mathtt{W}, \$, 0)));
\end{aligned}
$$

The intuition in Example 4.2 is to remove the first occurring number between the beginning of the source string and the white space, remove the first occurring white space flanked by a number and an alphabet token, replace the first occurring alphabet token with itself, and delete the rest of the tokens. The output of such a transformation program will be the target value corresponding to the source value showed in Example 4.2. This transformation program is associated with the format descriptor of the source values (note that in this case both source values have the same format descriptor). Consequently, each new string that follows the associated format descriptor will be a suitable input for the transformation program.

There are a number of questions left unanswered with respect to Example 4.2. For instance, why is the illustrated transformation program preferred over other, potentially equivalent programs, e.g., insert *"March"/"June"* at the beginning of the source string and delete all other tokens to obtain the target string? Or how is the *Copy* string expression synthesised? We answer such questions in the next section by describing the synthesis algorithm used to learn transformation programs from examples.

4.5 Synthesis algorithm

In this section we describe the algorithm that produces the transformation programs that edit the source values into their corresponding target values, and illustrate it using Examples 4.1 and 4.2. In particular, we focus on Example 4.1 because it requires a more complex program that includes multiple types of string expressions. Specifically, for the two source and target pairs mentioned in Example 4.1, the transformation program shown in Expression 4.1 is synthesised by the algorithm we are about to describe. Note that the transformation program from Expression 4.1 omits the position expressions for clarity and replaces them with the index of each token, counted from 0. The overall intuition is to obtain each target token using some processing of a source token. This would allow for the application of the same transformation program on new source strings, for which the corresponding target is unknown. Note that the transformation program in Expression 4.1 is consistent with both rows in Example 4.1, and, hence, the top–level *Switch* constructor has only one branch. In cases where the source strings instantiate more than one format descriptor, the top–level *Switch* constructor contains multiple branches, one for each format descriptor. In other words, the synthesis algorithm learns a conditional transformation program, as described in Section 4.5.1.1.

Given one example instance, the algorithm for synthesising the transformation program from Expression 4.1 consists of two main tasks:

1. Find the combination of edit operations that transform a string with the format of the exemplar source string into a string with the format of the exemplar target string. Clearly, there can be many such valid combinations, so the first task aims at exploring the, potentially huge, search space, to find *the simplest* edit operation combination, i.e., the combination with the fewest edit operations. We call this the *transformation search* task.

2. Map each generic token type from the generic transformation obtained at the previous step, against the correct string expression applied on some source token. Consequently, each target token in the transformation resulting from (1) above is expressed as a processing of some source token, making the resulting transformation applicable on new input strings, for which the corresponding output is yet unseen, i.e., not part of the example instances. We call this the *transformation instantiation* task.

$$
\begin{aligned}
&Switch((\mathtt{f}_s, \mathcal{T})),\ where\\
&\mathtt{f}_s = \mathtt{A\ W\ A\ W\ A\ W\ P\ N\ P\ N\ P}\\
&\mathcal{T} = SUB(\mathtt{A}^{\mathtt{s}}_0, Copy(\mathtt{A}^{\mathtt{s}}_0));\ SUB(\mathtt{W}^{\mathtt{s}}_0, Copy(\mathtt{W}^{\mathtt{s}}_0));\ SUB(\mathtt{A}^{\mathtt{s}}_1, Substr(\mathtt{A}^{\mathtt{s}}_1, 0, 1));\\
&\quad INS(Const(\text{"."})); SUB(\mathtt{W}^{\mathtt{s}}_1, Copy(\mathtt{W}^{\mathtt{s}}_0));\ SUB(\mathtt{A}^{\mathtt{s}}_2, Copy(\mathtt{A}^{\mathtt{s}}_2));\\
&\quad SUB(\mathtt{W}^{\mathtt{s}}_2, Copy(\mathtt{W}^{\mathtt{s}}_0));\ SUB(\mathtt{P}^{\mathtt{s}}_0, Copy(\mathtt{P}^{\mathtt{s}}_0));\\
&\quad SUB(\mathtt{N}^{\mathtt{s}}_0, Concat(Const(\text{"19"}), Copy(\mathtt{N}^{\mathtt{s}}_0)));\ SUB(\mathtt{P}^{\mathtt{s}}_1, Copy(\mathtt{P}^{\mathtt{s}}_1));\\
&\quad SUB(\mathtt{N}^{\mathtt{s}}_1, Concat(Const(\text{"19"}), Copy(\mathtt{N}^{\mathtt{s}}_1)));\ SUB(\mathtt{P}^{\mathtt{s}}_2, Copy(\mathtt{P}^{\mathtt{s}}_2));
\end{aligned}
\tag{4.1}
$$

$\mathtt{A}^{\mathtt{s}}_{\mathtt{i}}/\mathtt{A}^{\mathtt{t}}_{\mathtt{j}}$: the i^{th}/j^{th} token of type A from source/target, $i, j \geq 0$.

Before giving more details on the two steps above, consider the following. Given a collection of input specifications, input–output examples in our case, most traditional program synthesis algorithms include a search task that aims at identifying every transformation expressible in the underlying language that is consistent with the input specifications. Next, these transformations are usually ranked using some function to determine the most suitable transformation to be returned by the algorithm (or the top–k such transformations).

There are two important challenges faced by the solutions that employ such an approach. Firstly, subject to the number of operators supported by the underlying language, the space of all possible combinations of such operators that would be consistent with the input specifications can be huge. [Gul11] discusses the simple example in Figure 4.3 of transforming phone numbers from one format into another. The figure shows some of the different ways of generating parts of the output string from the input string, using FlashFill–specific *substring* and *constant string* operations. Even with only two string operations used, the various combinations shown suggests a problem of scale when searching for transformation candidates that are consistent with the input specifications.

The second challenge faced by most of synthesis algorithms for format transformation is the need to account for possible ambiguity in the input specifications. For instance, consider the following hypothetical pair of input–output strings, similar to the ones in Example 4.1: (*Gavin Paul Parker (90-96)*, *Gavin P. Parker (1990-1996)*). Assuming that we consider each target token as the result of transforming some source token, there is a question as to whether the middle name initial, P, in the target comes from the middle name or from the last name in the source. Of course, a human would use intuition to conclude that P is the

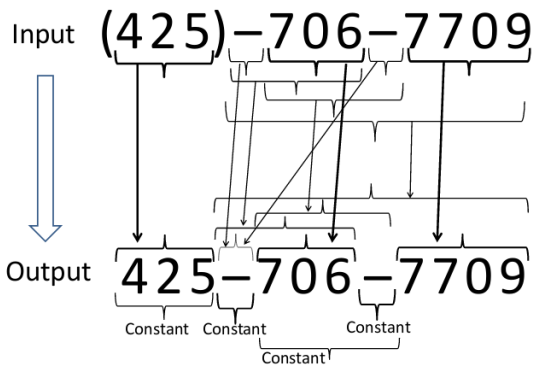


Figure 4.3: A sampling of different ways of generating parts of an output string from the input string.

initial of the middle name, but a synthesis algorithm has no access to intuition and has to consider both options, unless there is enough evidence in the example instances to filter out one of them. This has the effect of increasing the number of transformations that have to be considered.

Due to challenges such as discussed above, algorithms such as FlashFill are exponential in the number of examples and highly polynomial in the length of the examples [RGM14]. This is an important motivating factor for our work in this chapter, since exponential complexity is impractical for format transformation on many datasets.

4.5.1 Transformation search

We show how *SynthEdit* addresses the scalability challenge in this section. In searching for the transformations that are consistent with the input specifications, *SynthEdit* only considers a subset of the operators of the language presented in Section 4.4, viz., it only considers the three edit operations *insert*, *delete*, *substitute*, and delays the synthesis of string expressions until the transformation instantiation phase. One immediate consequence of this decision is that the resulting transformations will consist of operations with generic parameters, i.e., for each operand of an edit operation we will know its type, e.g., `A`, `N`, etc., but not its position in the source/target string. Identifying the correct positions for each operand of a generic edit operations, and, more importantly, the string expressions that have to be applied on some source token to obtain a target token, are addressed in the next section.

The generic version of the transformation in Expression 4.1, is Expression

4.2. Note that the token positions shown in Expression 4.2 simply denote the order of appearance of tokens in the format descriptors of the source string, $\mathtt{f}_{source} = \mathtt{A\ W\ A\ W\ A\ W\ P\ N\ P\ N\ P}$, and the target string, $\mathtt{f}_{target} = \mathtt{A\ W\ U\ P\ W\ A\ W\ P\ N\ P\ N\ P}$. Furthermore, each operation is a function of some source and some target token. Such transformations do not offer the possibility of applying them on some source string for which the corresponding target is unknown, i.e., target values are only known at synthesis time, nor do they provide a means to determine how to obtain the target tokens from some source token. However, using only three operations to express the transformations makes the search space more manageable. Furthermore, because these operations are the edit operations, there is an opportunity for using known edit distance computation techniques to perform the search task efficiently, as described next. To this end, we first need to introduce the notion of a *finite state transducer* (FST). We only introduce here the notions needed to explain the search technique we used in *SynthEdit*. A more comprehensive description of these notions is found in [JM09].

$$\begin{aligned} &SUB(\mathtt{A}^{\mathtt{s}}_{0}, \mathtt{A}^{\mathtt{t}}_{0});\ SUB(\mathtt{W}^{\mathtt{s}}_{0}, \mathtt{W}^{\mathtt{t}}_{0});\ SUB(\mathtt{A}^{\mathtt{s}}_{1}, \mathtt{U}^{\mathtt{t}}_{0});\ INS(\mathtt{P}^{\mathtt{t}}_{0});\\ &SUB(\mathtt{W}^{\mathtt{s}}_{1}, \mathtt{W}^{\mathtt{t}}_{1});\ SUB(\mathtt{A}^{\mathtt{s}}_{2}, \mathtt{A}^{\mathtt{t}}_{1});\ SUB(\mathtt{W}^{\mathtt{s}}_{2}, \mathtt{W}^{\mathtt{t}}_{2});\ SUB(\mathtt{P}^{\mathtt{s}}_{0}, \mathtt{P}^{\mathtt{t}}_{1});\\ &SUB(\mathtt{N}^{\mathtt{s}}_{0}, \mathtt{N}^{\mathtt{t}}_{0});\ SUB(\mathtt{P}^{\mathtt{s}}_{1}, \mathtt{P}^{\mathtt{t}}_{2});\ SUB(\mathtt{N}^{\mathtt{s}}_{1}, \mathtt{N}^{\mathtt{t}}_{1});\ SUB(\mathtt{P}^{\mathtt{s}}_{2}, \mathtt{P}^{\mathtt{t}}_{3}); \end{aligned} \tag{4.2}$$

Definition 4.2. *A finite state transducer is a 7–tuple* $(Q, \Sigma, \Gamma, \delta, \omega, q_0, F)$*, where:*

- Q *is a finite set of states;*
- Σ *is the finite input alphabet of the transducer;*
- Γ *is the finite output alphabet of the transducer;*
- $\delta : Q \times \Sigma \to Q$ *is the transition function;*
- $\omega : Q \times \Sigma \to \Gamma$ *is the output function;*
- $q_0 \in Q$ *is the start state;*
- $F \subseteq Q$ *is the set of accept states.*

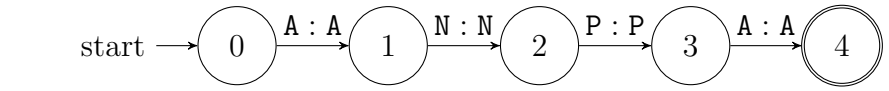
Such constructions are often used in areas such as speech recognition and machine translation (e.g., [MPR02]) to represent valid word sequences in a language model. One other application of particular interest for our case is in computing the edit distance between two strings, as described in [Moh02] and summarised next.

Broadly speaking, the action of an FST can be viewed as computing an operation on two strings. For example, consider an input alphabet Σ consisting of a subset of the regular expression–based token type symbols introduced in Section 4.4.1, i.e., $\Sigma = \{\mathtt{A}, \mathtt{N}, \mathtt{P}\}$. Consider the following two format descriptors consisting of characters from Σ, $\mathtt{f}_s = \mathtt{ANPA}$, and $\mathtt{f}_t = \mathtt{NANAP}$. Each of $\mathtt{f}_s$ and $\mathtt{f}_t$ can be represented in terms of a FST ($FST_{\mathtt{f}_s}$ and $FST_{\mathtt{f}_t}$), as shown in Figure 4.4, where each FST denotes an operation that leaves each character unchanged.

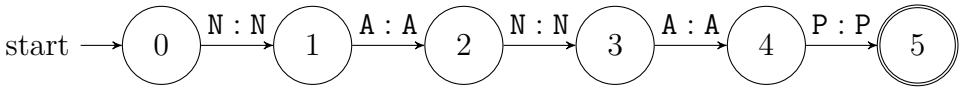
Similarly, and inspired by the algorithm for computing the edit distance between two FST–represented strings from [Moh02], for the entire alphabet $\Sigma = \{\mathtt{A}, \mathtt{N}, \mathtt{P}\}$, we can define an FST, called the *edit transducer* (ET_Σ), that represents the possible edit operations over elements of Σ. This transducer is shown in Figure 4.5.

Intuitively, the edit transducer ET_Σ describes a restricted set of edit operation as transitions between the elements of Σ, where $\mathtt{A} : \mathtt{A}$, $\mathtt{N} : \mathtt{N}$, $\mathtt{P} : \mathtt{P}$ denote *substitutions*, i.e., a transition from some element of Σ to itself; $\epsilon : \mathtt{A}$, $\epsilon : \mathtt{N}$, $\epsilon : \mathtt{P}$ denote *insertions*, i.e., a transition from the empty string ϵ to some element of Σ; and $\mathtt{A} : \epsilon$, $\mathtt{N} : \epsilon$, $\mathtt{P} : \epsilon$ denote *deletions*, i.e., a transition from some element of Σ to the empty string ϵ. Note that we do not consider substitutions where the left–hand side and the right–hand side are different, e.g., $\mathtt{A} : \mathtt{N}$. The intuition is that, when construing A, N and P as token types, defined in Section 4.4.1, a substitution between, say, an alphabet token and a numeric token, is not useful because there is no string expression in the language defined in Section 4.4.1 that would successfully translate a word to a number. The usefulness of this restriction becomes clear when the instantiation stage of the synthesis algorithm is presented, in the next section.

We now address the problem of computing the edit distance between two strings, defined as the minimal cost of a series of symbol insertions, deletions, or substitutions transforming one string into the other. The intuition is that the series of edit operations, whose cumulative cost equates to the edit distance, can be construed as a transformation that edits the source string into the target



(a) $FST_{\mathtt{f}_s}$



(b) $FST_{\mathtt{f}_t}$

Figure 4.4: FST representations of $\mathtt{f}_s$ and $\mathtt{f}_t$
where States 0 are the initial states and States 4 and 5 are the final states.

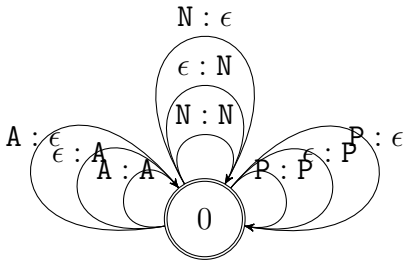


Figure 4.5: Edit transducer ET that succinctly describes all transitions allowed between elements of an alphabet $\Sigma = \{\mathtt{A}, \mathtt{N}, \mathtt{P}\}$.

string. The flip side is that the space of possible combinations of edit operations that have to be considered can be huge. For example, Figure 4.6 illustrates an FST representation of the search space of edit operations (with the restriction mentioned above that substitutions are only allowed between the same elements of alphabet Σ) that would transform $\mathtt{f}_s$ into $\mathtt{f}_t$.

The problem of efficiently computing the edit distance between two strings s and t given by their FST representations and an alphabet Σ, has been studied in [Moh02] where Theorem 2 is shown to hold:

Theorem 2. *Let U be an FST defined by $U = FST_s \circ ET_\Sigma \circ FST_t$. Let π be a shortest path of U from the initial state to the final states. Then, π is labeled with one of the best alignments between the string accepted by FST_s and the string accepted by FST_t and the edit distance between the two strings $d(s,t) = w[\pi]$, where $w[\pi]$ is the cumulative weight of the edges constituting the shortest path.*

In Theorem 2, the $\circ$ operator denotes the composition operation on automata [JM09]. [Moh02] also shown that the complexity of computing the edit distance between two string s and t is $\mathcal{O}(len(s) \times len(t))$. Theorem 2 guarantees, therefore,

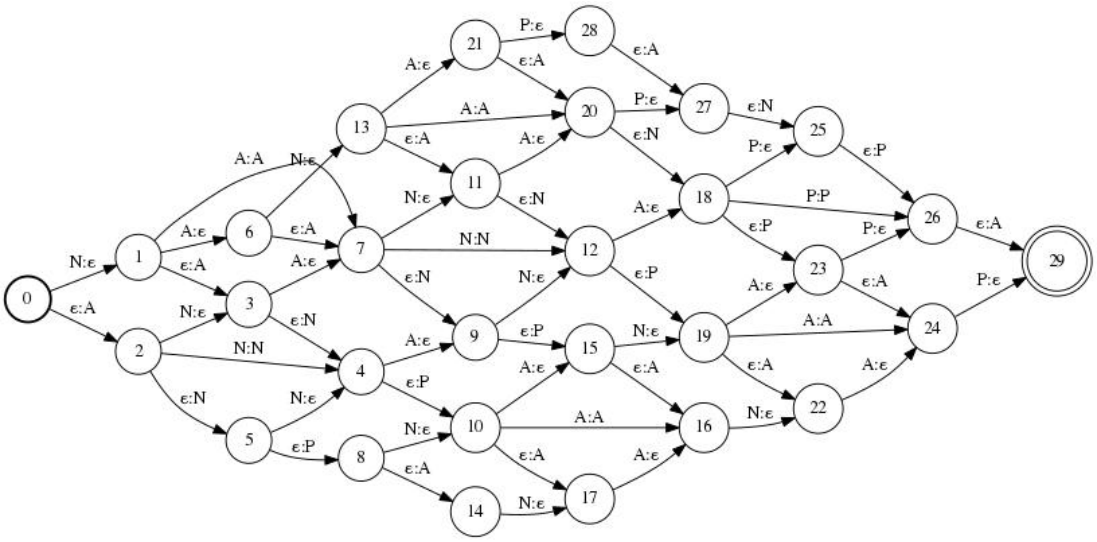


Figure 4.6: Edit operation search space: all possible combinations of edit operations that transform the $\mathtt{f}_s$ into $\mathtt{f}_t$.

the low complexity (at most quadratic on the length of the input–output strings) of our transformation search task.

Given $\Sigma = \{\mathtt{A}, \mathtt{N}, \mathtt{P}\}$, $\mathtt{f}_s = \mathtt{ANPA}$, $\mathtt{f}_t = \mathtt{NANAP}$, $FST_{\mathtt{f}_s}$ as depicted in Figure 4.4a, $FST_{\mathtt{f}_t}$ as depicted in Figure 4.4b, and ET_Σ as depicted in Figure 4.5, then U, i.e., the result of the composition between $FST_{\mathtt{f}_s}$, ET_Σ, and $FST_{\mathtt{f}_t}$, is shown in Figure 4.6. This is a representation of all possible edit operation combinations (i.e., the search space) that can transform $\mathtt{f}_s$ into $\mathtt{f}_t$. The shortest path from the initial state 0 to the final state 29 is illustrated in Figure 4.7. Note that, for the purpose of this chapter, we are not interested in the actual distance between the two strings, but in the sequence of edit operations that constitute the shortest path. As such, our transducers are not weighted. This means that the shortest path is simply the path with the fewest operations, and we construe such shortest path between the initial and final states as the simplest generic transformation, consistent with the input–output pair of strings. In Figure 4.7, this transformation is: $\mathcal{T} = DEL(\mathtt{N}); SUB(\mathtt{A}, \mathtt{A}); SUB(\mathtt{N}, \mathtt{N}); INS(\mathtt{P}); SUB(\mathtt{A}, \mathtt{A}); DEL(\mathtt{P})$. Note that, when there are multiple equivalent shortest paths from the initial state to the final state, we can simply choose one of them, since any such path would translate the source into the target.

Theorem 2 gives rise to Algorithm 4.1, which we employ in *SynthEdit* to perform the first task for the synthesis algorithm: searching for the best transformation consistent with the input–output pair of strings.

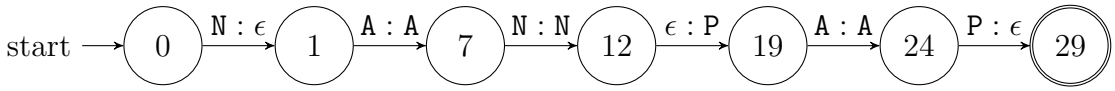


Figure 4.7: FST representation of the simplest transformation from $\mathtt{f}_s = \mathtt{ANPA}$ to $\mathtt{f}_t = \mathtt{NANAP}$

Algorithm 4.1 Transformation search

Input: Example instance (s, t), edit transducer ET_Σ of token types alphabet $\Sigma = \{\mathtt{N}, \mathtt{U}, \mathtt{L}, \mathtt{A}, \mathtt{Q}, \mathtt{W}, \mathtt{P}\}$
Output: A generic transformation $\mathcal{T}$

1: **function** TRANSFORMATIONSEARCH
2: $\mathtt{f}_s \leftarrow$ get_format_descriptor(s)
3: $\mathtt{f}_t \leftarrow$ get_format_descriptor(t)
4: $FST_{\mathtt{f}_s} \leftarrow$ get_fst$(\mathtt{f}_s)$
5: $FST_{\mathtt{f}_t} \leftarrow$ get_fst$(\mathtt{f}_t)$
6: $FST_{search\_space} \leftarrow FST_{\mathtt{f}_s} \circ ET_\Sigma \circ FST_{\mathtt{f}_t}$
7: $\mathcal{T} \leftarrow$ shortest_path(FST_{search_space})
8: **return** $\mathcal{T}$
9: **end function**

In Algorithm 4.1, the get_format_descriptor(s) function gets a string s as input and returns a sequence of tokens, from which we only consider the token types in this algorithm, by searching for the substrings of s that match one of the regular expressions presented in Section 4.4.1, a process we call *tokenization*. Specifically, through *tokenization* for each of the source values from Example 4.1, we obtain the following sequence of tokens:
$(\mathtt{A}, Pos(\hat{}, \mathtt{W}, \mathtt{0}))$, $(\mathtt{W}, Pos(\mathtt{A}, \mathtt{A}, 0))$, $(\mathtt{A}, Pos(\mathtt{W}, \mathtt{W}, 0))$, $(\mathtt{W}, Pos(\mathtt{A}, \mathtt{A}, 1))$, $(\mathtt{A}, Pos(\mathtt{W}, \mathtt{W}, 1))$, $(\mathtt{W}, Pos(\mathtt{A}, \mathtt{P}, 0))$, $(\mathtt{P}, Pos(\mathtt{W}, \mathtt{N}, 0))$, $(\mathtt{N}, Pos(\mathtt{P}, \mathtt{P}, 0))$, $(\mathtt{P}, Pos(\mathtt{N}, \mathtt{N}, 0))$, $(\mathtt{N}, Pos(\mathtt{P}, \mathtt{P}, 1))$, $(\mathtt{P}, Pos(\mathtt{N}, \$, 0))$.

The collection of left–hand side terms of each pair denotes the format descriptor of the tokenized string.

Function get_fst$(\mathtt{f})$ gets a format descriptor $\mathtt{f}$ as input, construed as a string, and returns an FST representation of $\mathtt{f}$, similar to the examples in Figure 4.4.

Lines 6 and 7 in Algorithm 4.1 represent the application of Theorem 2 from which we obtain the resulting sequence of edit operations (as exemplified in Figure 4.7), as a generic transformation (similar to the one exemplified in Expression 4.2) that we instantiate as described later. Therefore, under the theoretical guarantees of Theorem 2, the operations at lines 6 and 7, viz. *transducer composition* and

shortest path, are tractable, with a complexity given by $\mathcal{O}(len(s) \times len(t))$ [Moh02].

4.5.1.1 Transformation instantiation

Given an example instance such as the ones shown in Example 4.1, the transformation search task described in the previous section identifies a generic sequence of edit operations, such as the one shown in Expression 4.2. We call such a transformation *generic* because it allows for the editing of the token–type format representation of the source into the token–type format representation of the target, but not for the editing with respect to token values. Furthermore, each edit operation is a function of a source and a target token. The latter is know at synthesis time, but is not relevant when the transformation is applied on a new input string for which the corresponding expected output is unknown. In order to obtain a transformation that acts at token–value level and that can be applied on new input strings, we need to consider each target token in Expression (4.2) in the context of the value(s) of some source token(s), in order to obtain Expression 4.1. This is the goal of the methods described in this section.

The transformation instantiation task comprises the following two steps:

Step 1. Given a pair of input–output strings, the first step in instantiating a generic transformation is to identify, for each target token, the source tokens that can be used to derive it. In other words, for a target token t_i, we identify the source tokens whose values are syntactically closest to the value of t_i. To this end, we create an inverted index I in which each target token value identifies a list of pairs of the form (s_j, lcs_j), where lcs_j is the *longest common substring* between a source token s_j and t_i. For instance, the first two columns in Table 4.1 depict three index entries obtained for the first row in Example 4.1. The third column indicates the type of string expressions that can be applied on the source token to obtain the value of t_i, as described next.

The longest common substring between two strings s and t is the longest string shared by both s and t. This is a well studied problem, and we use a dynamic programming algorithm [Gus97] to solve it.

Step 2. With the index I indicating the source tokens for deriving each target token value, we can synthesise the string expressions to be applied on the former to obtain the latter. Algorithms 4.2 and 4.3 depict the process of synthesising a string expression given a target token and its corresponding list of pairs of the form (s_j, lcs_j), contained in index I created at the previous step.

Table 4.1: Index entries

	t_i	$[(s_j, lcs_{(s_j,t_i)})]$	**Expression**
1	Hugh	[(Hugh, Hugh)]	*Copy*
2	L	[(Leo, L)]	*Substr*
3	1974	[(74, 74)]	*Concat*

Algorithm 4.2 String expression synthesis

Input: Index entry: $t_i \rightarrow pairs_{t_i}$
Output: A string expression $\mathcal{E}$

```
1: function ExpressionSynthesis
2:     if pairs_{t_i} is empty then
3:         return Const(t_i)
4:     end if
5:     (s_j, lcs_j) ← best_pair(pairs_{t_i})
6:     if s_j == lcs_j == t_i then
7:         E ← Copy(s_j)
8:     else if lcs_j ⊂ s_j && lcs_j == t_i then
9:         E ← Substr(s_j, indexOf(lcs_j, s_j))
10:    else
11:        E ← Concat(ConcatSynthesis(t_i, pairs_{t_i}))
12:    end if
13:    return E
14: end function
```

For each entry of I, we can apply a function `ExpressionSynthesis` (defined in Algorithm 4.2) to synthesise a string expression that uses some source token values to obtain the target token value. The function in line 5 in Algorithm 4.2 returns a pair (s_j, lcs_j) where s_j is the source token whose value is the most similar to the target token value. Note that we only process the best such pair because its source token has the most useful value to derive the target token value. When the best pair is not the desired one, i.e. when the target token has to be obtained from a different source token, or when it is not clear which source token has the closest value to the target token value, we rely on multiple example instances to disambiguate and to synthesise a string expression. The `indexOf`(lcs_j, s_j) function at line 9 returns the start and end indexes of lcs_j in s_j.

When there is no source token that can be used to obtain t_i, `ExpressionSynthesis` returns a *Const* expression. Alternatively, if the longest common substring is identical to both the source and the target token values, the result is a *Copy*

expression (e.g., row 1 in Table 4.1). If the longest common substring is only equal to the target token value, it means that t_i can be obtained from the source token using a *Substr* expression (e.g., row 2 in Table 4.1).

Finally, when the source token value is a substring of the target token value, a *Concat* expression is synthesised using Algorithm 4.3. The `ConcatSynthesis` function processes all pairs for the current index entry, as opposed to only the best pair, and consumes the target token value as soon as it is able to derive a part of it. For example, for row 3 in Table 4.1, 74 is a source token and, therefore, the condition at line 4 in Algorithm 4.3 is met. At the next iteration $t_i = 19$, because we have consumed the previous value, there is no pair in $pairs_{t_i}$ that covers the new value, which means that the condition in line 11 evaluates to true and the next expression learned is *Const*. In the end, the target token in row 3 in Table 4.1 is obtained using the following string expression: $Concat(Const(\text{"19"}), Copy(\mathtt{N}_0^{\mathtt{s}}))$, where $\mathtt{N}_0^{\mathtt{s}}$ is the first occurring number in the source string in Example 4.1, i.e., 74.

Algorithms 4.2 and 4.3 describe the procedure that maps each target token to a string expression to be applied on one or more source token values. Thus, we can now replace each target token in Expression 4.2 with its corresponding string expression to obtain Expression 4.1, which expresses the desired transformation, is consistent with the given examples, and is suitable to be applied on new input strings that are similar in format representation to the source string.

In terms of complexity, the dominant task in instantiating a transformation is the construction of the inverted index in Step 1 above, where the longest common substring for each pair of tokens is generated using dynamic programming. Therefore, the complexity of index construction is given by $\mathcal{O}(k \times l \times u \times v)$, where k is the number of source tokens, l is the number of target tokens, u is the source token value length, and v is the target token value length.

4.5.1.2 Learning from multiple examples

The synthesis algorithm described in the last two sections receives as input one example instance, i.e., one pair of source and target strings, and returns a transformation that is associated with the format descriptor of the source string to create one branch in the top–level *Switch* constructor, as described in Section 4.4. In practice though, there are often multiple example instances available. There are two important aspects to consider here:

Algorithm 4.3 *Concat* expression synthesis

Input: Index entry: $t_i \rightarrow pairs_{t_i}$
Output: A list of string expression exp

```
1: function ConcatSynthesis
2:     exp ← [ ]
3:     for all (s_j, lcs_j) ∈ pairs_{t_i} do
4:         if s_j == lcs_j then
5:             exp ← exp + [Copy(s_j)]
6:         else
7:             exp ← exp + [Substr(s_j, indexOf(lcs_j, s_j))]
8:         end if
9:         t_i ← replace(t_i, lcs_j, "")
10:    end for
11:    if len(t_i) > 0 then
12:        exp ← exp + [Const(t_i)]
13:    end if
14:    return exp
15: end function
```

- If there are multiple examples with the same source format descriptor and the synthesis results in more than one transformation, the transformation that is consistent with most example instances is the one associated with the source format descriptor in the *Switch* constructor branch. This makes *SynthEdit* lenient with respect to potentially erroneous or ambiguous example instances, as long as there are more instances with the same source format descriptor available, thereby addressing the ambiguity challenge discussed before.

- If there are multiple example instances with different source format descriptors the examples are partitioned according to their source format descriptors, resulting in a conditional transformation program with as many branches as there are unique source format descriptors.

Finally, applying *SynthEdit* on new strings requires the format descriptor of the new input to match the format descriptor of one transformation program branch. If this does not hold, then the transformation is not performed and the input string is left unchanged.

The transformation language and synthesis algorithm described in this chapter

enable the synthesis of conditional transformation programs that can perform string processing operations, starting from input–output examples. The types of transformations that are covered by *SynthEdit* include:

- Permutations of existing string tokens, or of sub–tokens of existing string tokens achieved by applying SUB operations on tokens or string expressions.
- Addition of new (constant) tokens, based on other existing tokens or unrelated to them, achieved by applying INS operations on tokens or string expressions.
- Removal of existing tokens achieved by applying DEL operations.

One transformation type that is not supported by *SynthEdit* involves looping programs, i.e., performing the same operation on successive tokens. Synthesising and performing such transformations with *SynthEdit* requires an extension to the notion of format descriptor with the "one or more" (+) construct specific to regular expressions, so that strings with an arbitrary number of words are matched to the same program branch, i.e., the one that provides the loop transformation. We leave such extensions for future work.

4.6 Format transformation evaluation

In this section, we firstly present the data we use to evaluate the effectiveness and the efficiency of *SyntheEdit* and describe the measures we report. We then show the outcomes of a comparative evaluation of *SynthEdit* and *FlashFill* [Gul11]. The implementation of *FlashFill* used is from the PROSE SDK project (`https://microsoft.github.io/prose/`).

4.6.1 Data repositories used in evaluation

In the experiments, we have used the data benchmark available at `microsoft.com/en-us/research/wp-content/uploads/2016/12/WebTableBenchmark.zip`. It consists of 33 real–world data sets, with a total of approximately 80 pairs of input–output columns. Each pair of input–output columns consists of up to 200 example instances from several domains such as person names, phone numbers, websites, songs, etc. There are up to six basic types of transformations in the benchmark,

Table 4.2: Format transformation types

Transformation	**Source**	**Target**
Punct. removal	"Ask Me Why"	Ask Me Why
(Sub)Token extraction	Gov. Earl Warren - Republican	Earl Warren
(Sub)Token permutation	Peter Hardeman Burnett	Burnett Petter H.
(Sub)Token insertion	Maggie Paoletti	Prof. Maggie Paoletti
Token abbreviation	Statistical and Computer Science	SCS
Case alteration	CANNED DRIED ASPARAGUS	canned dried asparagus
Case alteration, token permutation, token insertion, token extraction	Prof. Benson Theo	tbenson@duke.edu

with many examples describing different combinations of the basic types. Table 4.2 gives source–target examples for each transformation type and one combination example (viz., the last row of Table 4.2). All transformation types are grounded on string operations that are supported by both *SynthEdit* and *FlashFill*.

4.6.2 Reported measures

We measure the following characteristics of both *SynthEdit* and *FlashFill*:

1. **Effectiveness** in performing string transformations on input strings. To this end, we report the *precision* and *recall* of both *SynthEdit* and *FlashFill* measured over all pairs of input–output columns mentioned above.

2. **Efficiency** in synthesising string transformations, given various numbers of example instances. To this end, we report the *synthesis time* required for each of *SynthEdit* and *FlashFill* to synthesise a transformation program.

For the purposes of computing precision and recall for (**1**) above, we define:

- **TP**. Any input string, not included in the examples from which a transformation program has been synthesised, that is correctly transformed by that transformation program is a true positive, in which case the output of the transformation applied on the input is equal to the expected output provided by the benchmark (recall that each pair of input–output strings contained up to 200 records of inputs and expected outputs).

- **FP**. Any input string, not included in the examples from which a transformation program has been synthesised, that is incorrectly transformed by that transformation program is a false positive, in which case the output of the transformation applied on the input is different from the expected output provided by the benchmark.

- **FN**. Any input string, not included in the examples from which a transformation program has been synthesised, that is left unchanged by that transformation program is a false negative, in which case the format descriptor of the input string did not match any format descriptor from the conditional transformation program.

As usual, precision p is defined by:

$$p = \frac{TP}{(TP + FP)}$$

Recall r is defined by:

$$r = \frac{TP}{(TP + FN)}$$

We report the average precision, recall and synthesis time over all pairs of input–output columns using a k–fold cross–validation technique and controlled numbers of examples. At each iteration (fold), we synthesise a transformation program from n randomly picked example instances and test on the remaining instances.

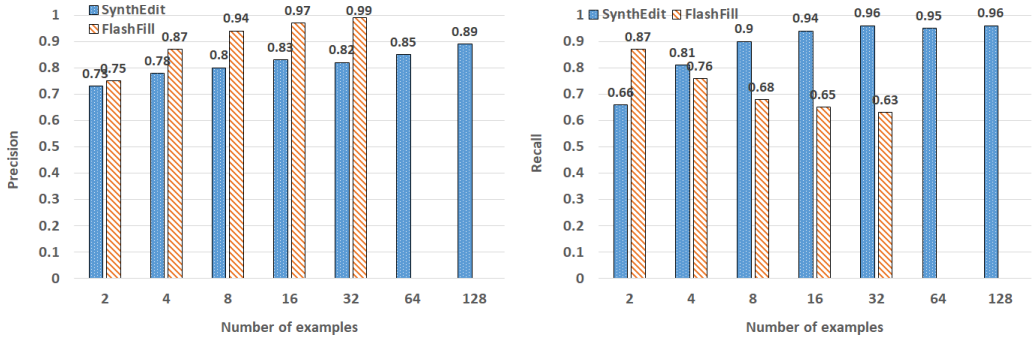
All experiments have been run on a 2.60 GHz Intel Core i7-4720HQ CPU with 12 GB RAM.

In the results below, each point is the average computed by running $SynthEdit$ and $FlashFill$ over ten folds.

4.6.3 Comparative effectiveness

The purpose of this experiment is to compare the effectiveness of $SynthEdit$ and $FlashFill$ as the number of examples varies.

The precision results are shown in Figure 4.8a. $SynthEdit$ achieves lower precision compared with $FlashFill$ for the different numbers of examples. The difference can be explained by the ability of $FlashFill$ to better generalise transformations as more examples are added, by using a classifier trained on example



(a) Format transformation precision (b) Format transformation recall

Figure 4.8: Average format transformation precision and recall resulted from a 10–fold cross–validation process.

instances. This allows it to correctly transform strings with format representations not covered, but close to the ones in the examples, and to partition the example instances based not only on format descriptor–like patterns, but on features of the values as well. As a result, strings that, according to *SynthEdit* have similar format descriptors and, therefore, are subject to the same transformation, would be treated differently in *FlashFill* by being subject to different transformations. For instance, consider Example 4.3 where the intuition is to parse the name from *Source* as last name followed by first name initial, unless the person is a Professor, in which case the title should be included in the target. For such a scenario, *SynthEdit* synthesises a conditional transformation program with three branches, out of which only one covers examples that contain the title in the source string, even though row 1 in Example 4.3 should generate a different transformation from the one generated by rows 2 and 5. In contrast, *FlashFill* is able to create a conditional transformation program based on the value of the title, which is the correct way of treating this scenario.

For the last two cases in Figure 4.8a, i.e., 64 and 128 examples, there are no results to report for *FlashFill* because it required more than the 12 GB of RAM memory that was available.

Example 4.3. *Name parsing:*

Source	Target
Dr. Eran Yahav	Yahav, E.
Prof. Kathleen Fisher	Prof. Fisher, K.
Bill Gates	Gates, B.
George Ciprian Necula	Necula, G.
Prof. Ken McMillan	Prof. McMillan, K.

The down side of using classifiers in picking the right transformation given a new input string, as is the case for *FlashFill*, is that, in some cases, there can be multiple transformation candidates with close probabilities. In other words, the classifier can become confused when the features of the new input format are too close to the features of more than one class used during learning, or even fail to classify a valid input string. This means that, for some input strings, *FlashFill* fails to identify the appropriate transformation, or the transformation picked is not consistent with the input, e.g., the transformation expects a type of token that is not present in the input. The consequence is a drop in recall that is reflected in Figure 4.8b as the number of examples increases. In contrast, *SynthEdit* achieves better recall because the strict one–to–one mapping between format descriptors and input strings enables it to better differentiate between transformation cases.

The lower precision of *SynthEdit* can be countered by performing additional evaluation before deciding on the transformation to apply. At present, the conditional evaluation and the partitioning of examples at synthesis time are done based on format descriptor equality. A potential improvement would be to perform a subsequent value–based conditional evaluation when the format descriptors are equivalent, but the transformations synthesised are different. This is a potential path for future work.

4.6.4 Comparative efficiency

The purpose of this experiment is to compare the efficiency of the synthesis algorithms employed by *SynthEdit* and *FlashFill*, as the number of examples increases.

Figure 4.9 confirms the high cost of *FlashFill* when the number of examples is large. In contrast, *SynthEdit* proves more than two orders of magnitude faster in synthesising transformations (note the logarithmic scale of the plot).

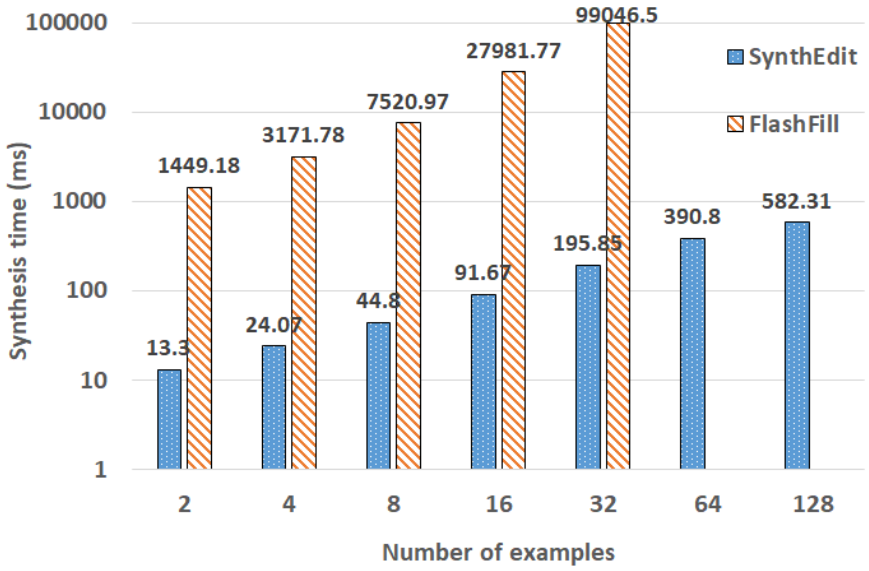


Figure 4.9: Average format transformation efficiency resulted from a 10–fold cross–validation process.

As opposed to *FlashFill*, *SynthEdit* does not aim to exhaust the search space of transformations for each example instance. Our algorithm uses edit distance computations to find the shortest path between the token–type representations of the source and target strings by employing FST operations which perform transformation search in quadratic time for each input–output example instance (as described in Section 4.5.1). Consequently, the transformation language is simpler but more efficient to learn.

As before, for 64 and 128 examples, *FlashFill* requires more than 12 GB of RAM memory.

4.7 Summary and conclusions

This chapter has contributed an effective and efficient solution to the problem of automating format transformation given input–output examples. We have used an edit–distance based approach that identifies the shortest path from a source string to a target string and uses matching of source and target tokens to generalise transformations applicable on new input strings, similar in format representation to the examples. Results from a comparative evaluation provide evidence that *SynthEdit* performs substantially more efficiently than the state-of-the-art, while achieving comparable precision and better recall in comparison

with the state–of–the–art.

Overall, the main characteristics of the solution proposed in this chapter are:

- *SynthEdit* requires a simple input consisting of input strings, paired with expected output strings. Input–output examples are one of the simplest forms of input specification in program synthesis, requiring limited knowledge about the data to be transformed and no knowledge about the underlying language used to express the transformations. Therefore, *SynthEdit* meets the desire for simple specifications stated in Section 4.1.

- Assuming the set of input–output examples describes a transformation program within the boundaries of the language defined in Section 4.4, *SynthEdit* can reach a transformation that is consistent with the given examples automatically, without additional user intervention. Therefore, the automation desideratum in Section 4.1 is satisfied.

- Owning to being grounded on automata–specific techniques when computing and searching the space of possible transformations consistent with the given examples, the synthesis algorithm proposed in Section 4.5 is characterised by quadratic complexity with respect to the length of the examples (recall Theorem 2). This algorithmic complexity, together with empirical evidence shown in Section 4.6, means that the fast convergence desideratum in Section 4.1 is satisfied.

- When the synthesised transformation program is checked against new inputs, *SynthEdit* proves effective, as long as the required sequence of operations is (a) consistent with the examples, (b) expressible using the language defined in Section 4.4, and (c) the examples used to synthesise the transformations do not exhibit ambiguity with respect to string processing operations. When compared with the state–of–the–art, *SynthEdit* achieves better recall and comparable precision. In other words, the desideratum for effectiveness in Section 4.1 is satisfied if (a) there are enough examples to synthesise a transformation, (b) the input data has a regular structure, and (c) there is a clear correspondence between the tokens of the input and corresponding output strings. For all other cases (e.g., repetitive (or loop) transformations), high effectiveness can still be achieved in the presence of many examples covering the same transformation case.

Finally, the evaluation results constitute compelling evidence that *SynthEdit* has the potential of fulfilling the second research objective stated in Section 1.4 of Chapter 1: (i) *SynthEdit* relies on a transformation language that can express common string processing operations involved in modifying the format representation of strings, and (ii) *SynthEdit* includes a synthesis algorithm that, starting from simple input–output example strings, can efficiently learn a transformation program expressed in the language from (i) that is consistent with the examples. When applied at larger scale, the simplicity of the input–specifications required demands reduced familiarity with the data values from the user, while the automated solution for synthesising a transformation program demands no advanced technical knowledge, e.g., programming expertise, from the user.

Chapter 5

Data format transformation in data lakes

The discussion in Chapter 4 has focused on algorithms that enable the synthesis of format transformation programs from input–output examples. In describing such algorithms we took a detour from the data lake–centric viewpoint that had governed the discussion until then. We have shown that the proposed synthesis algorithm, i.e., *SynthEdit*, outperforms the state–of–the–art alternatives with respect to the time required to learn a transformation program. However, independently of how many input–output examples are used, the assumption in Chapter 4 is that the user provides them and that they cover most of the transformation scenarios that have to be performed. Returning to the issue of performing data format transformation at data lake scale, we observe that these assumptions do not hold when there are different transformation needs among the instance values of the same column, many such columns and many datasets to transform. Thus, this chapter aims to provide automatic techniques that can assist (or even replace) the user in the task of providing input–output examples to transformation synthesis algorithms such as *SynthEdit* or *FlashFill*, and, therefore, fully automate the process of format transformation.

In short, the aim of this chapter can be summarised as follows:

> *Given a collection of datasets from which pairs of source and target columns with overlapping, albeit potentially differently formatted, values can be identified, automatically pair source and target instance*

values that represent similar concepts and can be construed as input–output example instances for a format transformation synthesizer.

If Chapter 4 aimed at automating the production of format transformation programs starting from given input–output examples, this chapter aims at automating the production of examples, i.e., it focuses on the discovery of value pairs that can be used as examples for synthesis algorithms. As already mentioned before, automatic solutions are unlikely to be able to match the reach or quality of transformations produced by data scientists, but any level of automation provides the possibility of added value for minimal cost by relieving the user from the burden of finding suitable example instances for each transformation case. Thus, we adhere to the ethos of cost–effective data wrangling introduced in Chapter 2 that aims for minimal user involvement in the data preparation stage.

In Section 5.1 we motivate the need for automatic discovery of examples for format transformation synthesizers and discuss our desiderata for the proposed methods. Then, in Section 5.2 we introduce the underlying notions upon which the proposed techniques are built, and discuss the relevant related work. The contributions claimed in this chapter are briefly outlined in Section 5.3, and detailed in Sections 5.4 and 5.5. We close the chapter with an empirical evaluation of the chapter's proposals in Section 5.6, before drawing some conclusions in Section 5.7.

5.1 Motivation and desiderata

Many of the state–of–the–art proposals for format transformation (e.g., FlashFill [Gul11], Wrangler [KPHH11]) assume format transformation scenarios that are characterised by (i) inputs of manageable sizes, i.e., few transformation cases that require a reduced number of input specifications, so that the search space of potential transformations is manageable, and (ii) a small number of datasets to be transformed, so that the user can analyse them individually, provide covering examples, and supervise the format transformation task applied on each one. While we have shown in Chapter 4 how *SynthEdit* dispenses with (i), the impact of assumption (ii) was not addressed there. Therefore, in this chapter, we are motivated by the need to automate the identification of suitable example instances for format transformations synthesizers, so that they can be applied

when there are too many transformation cases involved, or when the user is not so familiar with the transformation needs of each dataset, as to be able to manually provide covering examples. Specifically, in this chapter we propose an automated solution for the production of input–output example instances with the following properties:

Validity. The candidate pairs of input–output strings to act as example instances have to denote a valid transformation. For instance, the two strings of an example instance have to represent the same real world entity when the transformation denotes a variation in the format representation (e.g., calendar dates expressed in different formats), or the entity represented by the target value has to be subsumed by the source entities when the transformation denotes an extraction operation (e.g., the street name in an address string). When this is the case, we say that the example instances are valid.

Representativity. Synthesis algorithms such as *SynthEdit* rely on example instances to learn transformations that can be applied on unseen input strings. Therefore, the example instances have to be representative for the unseen strings to be transformed, i.e., to cover the formats and transformation cases exhibited by the strings to be transformed.

Non–redundancy. Although *SynthEdit* proved more efficient in synthesising transformations than the closest alternative from the state–of–the–art, it can still be impacted by a large number of examples. Consequently, when the number of automatically generated example instances is large, a pruning strategy must eliminate instances that exemplify transformations already covered by previous examples.

Automation. The generation of examples that meet the conditions mentioned above has to be performed automatically, without user intervention. Consequently, we have to identify suitable sources of evidence and devise techniques that ensure the validity, representativity, and non–redundancy of the example instances produced.

5.2 Background and related work

In achieving our goal, in this chapter we rely on technical background that allows us to meet the desiderata defined in Section 5.1. We now describe this technical

background. We also analyse how the closest alternatives for identifying pairs of similar strings relate to our proposal, although many such alternatives have not been designed for the generation of examples for format synthesis algorithms and, consequently, fall short of delivering on the above desiderata.

5.2.1 Matching relationships

Given pairs of attributes that contain instances from similar real world domains, we need to pair (i.e., align) instance values that represent similar entities. Consequently, only the pairs of attributes with values drawn from the same domain (i.e., that are semantically related, and that share some values) are candidates to have their instances aligned. We use schema matching [RB01] for this purpose, i.e., we consider such attributes to be in a *matching relationship*. In particular, we are interested in *instance–level matching* relationships: a matching relationship defined by the overlap in the value extents of two attributes.

Definition 5.1. *Given two attributes s and t from two different datasets, with their value extents seen as sets of values $V(s)$ and $V(t)$, resp., we say that s matches t (or that t matches s) iff $|V(s) \cap V(t)| \geq \theta$, where θ is a matching threshold.*

The role of θ in Definition 5.1 is to avoid performing all–against–all alignments between the attributes of two given datasets, i.e., we use schema matching to obtain evidence of instance–level similarity between attributes before aligning their values.

Note that Definition 5.1 does not take into account potential discrepancies between the format representation of s values and t values. In fact, in this chapter, such discrepancies are assumed to be present, since uniform format representations would not require any format transformation. Also note that matching discovery relies on instance values set–similarity, as discussed in Section 3.2.2. Therefore, and as per Definition 3.1, s and t can be considered related. It follows that, by replacing the sets of values $V(s)$ and $V(t)$ from Definition 5.1 with *t*sets $T(s)$ and $T(t)$, defined in Section 3.4.1, we can rely on techniques described in Section 3.4 to identify instance–level matching candidates, despite the potential presence of inconsistencies in format representation of values. Alternatively, we can use schema matching tools, such as COMA++ [ADMR05], to discover matching

relationships. This is possible because matching tools such as COMA++ include string similarity discovery techniques that act at token level, therefore not being impacted by format representation heterogeneity.

Schema and ontology matching is a vast research area that engages interest from multiple research communities such as relational databases and data integration (e.g., [Rah11], [HRO06], [RB01]), and ontologies and the semantic web (e.g., [SE13], [Noy04]). The main goal is to find correspondences between table columns/ontology concepts that can be postulated as equivalences and thereby underpin the integration of data from multiple such sources. In this chapter, we use off–the–shelf schema matching techniques as a precursor for examples generation as our needs expand to identifying column–level correspondences, i.e., with such correspondences in hand, we perform instance value alignment.

Finally, we note that, although the discovery of matching column pairs, i.e., candidates for performing examples generation and transformation synthesis, can be automated through the use of techniques such as the ones mentioned above, the transformations themselves are not symmetric, i.e., given a transformation τ, $\tau(s) = t$ does not entail that $\tau(t) = s$. Therefore, in this chapter, we rely on the user to specify the dataset whose (source) columns contain values that have to be formatted differently, and the dataset whose (target) columns contain values that denote the expected/desired formatting. Although it is possible to set these roles automatically, in this chapter we do not address this need. We do focus on how to automatically generate example instances for a pair of matching column, once the source and the target are known, which is the basic, default scenario in data wrangling.

5.2.2 Functional dependencies

In addition to matching relationships, we use hypothesised functional dependencies (FDs) [Cod71], when available, to generate (and to check the validity of) instance value alignments. In this section, we briefly recall the well known notion of functional dependencies in tabular data, such as relational databases. Later, in Section 5.4, we describe how such relationships are used to generate value alignments.

Definition 5.2. *Given two attributes s_1 and s_2 from the same dataset D, a functional dependency (FD), written as $s_1 \rightarrow s_2$, expresses that all pairs of tuples*

from D that have the same value on s_1 must also have same values on s_2.

Consequently, the left–hand side of a FD might be a key in a relational dataset, because keys uniquely determine all other attributes. Functional dependencies arise in real–world datasets as well, e.g., a national insurance number determines a person's name, or a person's name may determine the person's gender, etc.

In Definition 5.2, s_1 is called *determinant* and s_2 is called *dependant*. Although, both the determinant and the dependant can be sets of attributes, in this chapter, we only consider FDs between single attributes, as these suffice for us to reliably identify instance value alignments suitable as examples for synthesizers.

Lastly, in practice, FD discovery is often performed using specialised data profiling tools, e.g., Metanome [PBF+15], and algorithms, e.g., HyFD [PN16].

5.2.3 State–of–the–art

Research on data cleaning and transformation includes a number of proposals for automating the alignment of instance values between matching columns, many of which have been designed with other goals in mind than the generation of example instances for synthesis algorithms.

An important body of work that is close to our proposal and was developed with the goal of improving transformation synthesis is [WK16]. In this approach, an example recommending algorithm is proposed to assist the user in providing enough examples for an entire column to be transformed. To this end, the approach samples a set of records for automatic inspection. It then uses machine learning techniques to identify potentially incorrect records and presents these records for the users to examine. This proposal can be considered orthogonal to our approach as the techniques described in this chapter are construed as an automatic initialisation phase of a pay-as-you-go process [PBE+16], of which [WK16] can be construed as the refinement phase, i.e., one where previous results are improved with user support.

In recent years, there has been an increasing number of proposals that use declarative, constraint-based quality rules to detect and repair data values (e.g. [YGCC12, CIP13, FLM+12], and see [Fan08, Fan15, FG12] for surveys). For many of these heuristic techniques, rule–based corrections are possible, as long as the repairing values are present either in an external reference data set or in the original one. For example, in [FLM+12] the correct values are searched for

in a master dataset using editing rules that specify how to resolve violations, at the cost of requiring intense levels of user involvement. While the semantics of editing rules can be seen as related to the approach we describe in Section 5.4, there are two essential differences. Firstly, editing rules deal with instance–level repairs, i.e. every tuple is checked against every rule and the values of the columns covered by the rule are replaced with correct ones from the reference data set (if they exist). Our approach determines pairs of values from which we can learn transformations that hold for entire columns, so we do not search for the correct version of every value that needs transformation, but for a small number of values that describe the formatting of the correct version. Secondly, we determine the alignments automatically, without user guidance.

Recent work on transformation–driven join operations [ZHC17] resulted in a technique for automatically joining two tables using fuzzy value pairing and synthesis of transformation programs. Their approach leverages substring indexes to efficiently identify candidate row pairs that can potentially join, and then it synthesises a transformation program whose execution can lead to equijoins. Although their principle is similar to the approach described in Section 5.5, their focus on joining operations requires the columns from which examples are being searched to be candidate keys in their respective datasets (or one key and one foreign–key). Our approach aims to enable normalisation for any column that has a match with a target column.

5.3 Overview and contributions

In this chapter, we propose two techniques for automatically generating input–output example instances for synthesis algorithms that demand such input specifications, starting from a target dataset (that contains some instances of values with the desired formatting), and one or more source datasets (that contain values to be transformed). The overall goal is to enable program synthesis from examples to work with numerous sources or with many transformation cases that would be laborious to cover by user–provided examples.

The first technique makes use of matching candidates and hypothesised functional dependencies to identify value alignments. Specifically, given two datasets S and T in which we identify two functional dependency candidates, $S.a \rightarrow S.b$ and $T.c \rightarrow T.d$, and two matching candidates $(S.a, T.c)$ and $(S.b, T.d)$, we align values

from the right–hand sides of the functional dependencies, $S.b$ and $T.d$, where their corresponding left–hand sides, $S.a$ and $T.c$, have equal values. For example, in Figure 5.1 (a) and (b), in order to pair together the values from $S.Date$ and $T.Date$, we need the FDs $S.Permit\_Nr. \rightarrow S.Date$ and $T.Permit\_Nr. \rightarrow T.Date$, and the matching instances $(S.Permit\_Nr., T.Permit\_Nr.)$ and $(S.Date, T.Date)$. In addition, we need the values of the $Permit\_Nr.$ columns to have overlapping values. We argue that the resulting pairs can be used as examples for synthesizers to transform the format of the values from $S.Date$ to the format represented in $T.Date$, as described in the next section.

The second technique addresses Web–extraction scenarios. In the case of Web–extracted data, functional dependency candidates are often hard to discover due to data inconsistencies or simply because such relationships do not exist. To address this scenario, we propose a second, less restrictive technique, based on string similarities and candidate matching relationships, which proves to have comparable effectiveness at the expense of efficiency. Specifically, for each matching pair candidate $(S.a, T.b)$, we do a string similarity–based pairing of values from $S.a$ and $T.b$. Then, we assign a confidence measure to each pair of values and use it to select a subset of pairs as examples for synthesis algorithms. For instance, in Figure 5.1, we pair the values of $S.Date$ and $T.Date$ that represent the same date using their string similarity, without requiring a common tuple identifier such as $Permit\_Nr.$

Overall, the main contributions of this chapter are:

- We describe a FD–based examples generation technique that, given a target and one or more source datasets, relies on the existence of hypothesised FDs and matching relationships to automatically align the instance values of some source column and the matching target column. The resulting alignments can be construed as examples for transforming the rest of the source values.

- In case FDs cannot be discovered, we describe a string similarity–based examples generation technique that, given a source and a target datasets, relies on schema matching and string similarity (i.e., approximate) alignment of source and corresponding target values. As before, the resulting alignments can be construed as examples for transforming the rest of the source values that are not part of the generated examples.

- Given the approximate nature of the second proposal, the number of potentially

S.Permit_Nr.	S.Date	S.Contractor	S.Cost	S.Address	S.Permit_Type
100484472	2013-05-03	BILLY LAWLESS	83319	730 Grand Ave	Renovation

(a) Source

T.Permit_Nr.	T.Date	T.Cost	T.Contractor	T.Address	T.Type
100484472	05/03/2013	$83319.00	BILLY LAWLESS	Grand Ave, Nr. 730	Renovation

(b) Target

$Matches(S,T)$	$\{\langle S.Permit\_Nr., T.Permit\_Nr.\rangle, \langle S.Date, T.Date\rangle, \langle S.Cost, T.Cost\rangle, \langle S.Address, T.Address\rangle, ...\}$
$S.FD = T.FD$	$\{Permit\_Nr. \rightarrow Date, Permit\_Nr. \rightarrow Cost, Permit\_Nr. \rightarrow Address, Permit\_Nr. \rightarrow Contractor, ...\}$
Generated Examples	$\{\langle S.Date, T.Date, \langle$"2013 − 05 − 03", "05/03/2013"$\rangle\rangle, \langle S.Cost, T.Cost, \langle$"83319", "$83319.00"$\rangle\rangle, \langle S.Address, T.Address, \langle$"730 *Grand Ave*", "*Grand Ave, Nr.* 730"...$\}$

(c) Partial intermediate and final results from the algorithm

Figure 5.1: Examples of source input, target input, intermediate results, and final results of the examples generation process.

valid value alignments generated can be very large. We describe a pruning and selection strategy that identifies the most relevant value alignments to be used as examples for a synthesis algorithm.

- We perform an empirical evaluation of the above approaches with real–world data.

5.4 Discovering examples: FD–based scheme

In this section, we describe an approach to the automatic identification of examples, drawing on the relationships between attributes. Our aim is to use transformation program synthesis in more complex scenarios than spreadsheets. For example, consider a scenario, similar to the one in Figure 5.1, in which we would like to integrate information about issued building permits from several different sources, and for the result to be standardised to a single format per column. Specifically, we want to represent the columns of the resulting dataset using the formatting conventions used in one of the original datasets, which acts as the target. Providing manual examples to synthesise the transformations needed can be a non–trivial and costly task that requires knowledge of the formats of the values existing in the entire dataset. In the approach described next, the core idea

is to identify/align values from source and target datasets, where the source and target values for a column can be expected to represent the same information.

5.4.1 Examples generation

Assume we have two descriptions of an issued building permit, as depicted in Figure 5.1. To generate a transformation that applies to the *Address* columns, we need to know which values in the source and target *Address* columns are likely to be equivalent, despite being differently formatted. There are different types of evidence that could be used to reach such a conclusion. In the approach described here, we would draw the conclusion that *730 Grand Ave* and *Grand Ave, Nr. 730* are equivalent from the following observations:

- the names of the first column in the two tables match (because of the identical substring $Permit\_Nr.$ they share);
- a functional dependency holds for the instances given, $Permit\_Nr. \rightarrow Address$ in each of the tables;
- the names of the fifth column ($Address$) of the two tables match;
- the values for the first column in the two tuples are the same.

In practice, the name–based matching evidence has to be complemented by value overlap because the latter ensures the existence of valid alignments between values.

Note that the previous types of evidence do not guarantee a correct outcome since it is possible for the given conditions to hold, and for the values not to be equivalent. As an example of such a case, the *Address* attributes of the source and target tables might have had different semantics.

More formally, assume we have two data sets, source S and target T. S has the attributes $(sa_1, ...sa_n)$, and T has the attributes $(ta_1, ...ta_m)$. We want values from S to be formatted as in T. Further, assume that we have instances of S and T available. Then we can run an FD discovery algorithm (e.g. [PN16]) to postulate that FDs exist between attributes of S and T. This gives rise to collections of candidates FDs for S and T, $S.FD = \{sa_i \rightarrow sa_j, ...\}$ and $T.FD = \{ta_u \rightarrow ta_v, ...\}$ (as exemplified in Figure 5.1 (c)).

In addition, assume we have a function $Match$ that, given S and T, returns a set of pairwise matches between the attribute in S and T, $Match(S,T) = \{\langle sa_i, ta_j\rangle, ...\}$. $Match$ can be implemented using a schema matching algorithm, most likely in our context making use of instance level matchers [RB01], given the need for shared values.

Then, Algorithm 5.1 can be used to compute a set of examples for synthesising transformations between values from columns of S and of T.

Algorithm 5.1 relies on the definition of functional dependencies (Definition 5.2) and the existence of matching relationships (Definition 5.1) between attributes of S and T to align instance values that represent similar real world concepts and that can be used as examples for synthesis algorithms. When such relationships exist, the query in lines 8–10 generates value alignments such as the ones exemplified in Figure 5.1 (c).

5.4.2 Examples validation

Although Algorithm 5.1 can return sets of examples that can be used for synthesising transformations, there is no guarantee that the transformation generation algorithm will produce effective transformations. The synthesised transformations may be unsuitable for various reasons, e.g., (a) the required transformation cannot be expressed using the available transformation language, (b) the data is not amenable to homogenisation (e.g. because there is no regular structure in the data), or (c) there are errors in the data. As a result, there is a need for an additional validation step that seeks to determine, again automatically, whether or not a suitable transformation can be synthesised.

In our approach, the set of examples returned by Algorithm 5.1 is discarded unless a 10–fold cross validation process is successful. In this process, the set of generated examples is randomly partitioned into ten equally sized subsets. Then, transformations are synthesised, in ten rounds, using examples from nine partitions, and tested on the remaining partition. Thus, the validation step ensures that the generated examples are likely to return valid transformations (recall the validity desideratum from Section 5.1).

The cross validation step can be used with different thresholds on the fraction of correct results produced. In Section 5.6, our approach is to retain a set of examples only if the synthesised transformations behave correctly throughout all the iterations of the validation process.

Algorithm 5.1 Example discovery using functional dependencies.

```
1: function FdEgsGen(S,T)
2:     Egs ← {}
3:     for all sa ∈ S and ta ∈ T do
4:         if ⟨sa, ta⟩ ∈ Matches(S, T) then
5:             for all (sa → sad) ∈ S.FD and
6:                 (sa → sad) ∈ T.FD do
7:                 if ⟨sad, tad⟩ ∈ Matches(S, T) then
8:                     EgVals ← select distinct
9:                 S.sad, T.tad from S, T where
10:                S.sa = T.ta
11:                    EgPairs ← (sad, tad, ⟨EgVals⟩)
12:                    Egs ← Egs ∪ {EgPairs}
13:                end if
14:            end for
15:        end if
16:    end for
17:    return Egs
18: end function
```

5.5 Discovering examples: weighted scheme

The technique described in the previous section uses a set of hypothesised functional dependencies to pair values from a source dataset $S : (sa_1, ...sa_n)$ and a target dataset $T : (ta_1, ...ta_m)$, that represent the same real world entity. While such an approach can be effective in some scenarios (as shown in Section 5.6), often the assumptions underlying Algorithm 5.1 are too strong. For instance, candidate FDs may be missing, or state–of–the–art FD discovery is unable to find them due to inconsistencies in values. Furthermore, the number of example pairs generated by Algorithm 5.1 can be very large, while the number of cases covered by the examples is small, i.e., Algorithm 5.1 offers no control over the number of examples generated for one transformation case, e.g., changing the format representation of a date string. For such simple cases, techniques such as *FlashFill* or *SynthEdit* often require few example instances. Not only is it the case that many examples that cover the same transformation scenario are redundant, but they can increase the runtime of the synthesis process. As mentioned in Chapter 4, *FlashFill* is known to be exponential in the number of examples and high degree polynomial in the size of each example [RGM14].

To address these scenarios, in this section we describe an approach to automatic

identification of examples that uses string similarities between values in pairs of matching column, as returned by the $Match$ function introduced in the previous section. We also propose an incremental example selection algorithm that, out of the example pairs that are generated, selects a relevant subset for which there is evidence that effective transformations can be synthesised.

The objective of the example discovery technique described next is: given two columns from different tables, to heuristically identify pairs of values that are similar. To this end, we start by *tokenizing* each column value, where each token is a substring of the original value delimited by punctuation or spaces. The tokens enable the *grouping* of values into blocks, where each block contains values from both columns with common tokens. An intra–block, pairwise *comparison* of values is conducted to determine potentially equivalent instances. Given the fact that the syntactic comparison does not guarantee the optimal pairing of equivalent values, each candidate pair obtained will have an associated score. This quantifies the degree of similarity of the paired values, and is used later to determine a minimal sub–set of candidate value pairs that can be used as examples to synthesise an effective transformation program.

5.5.1 Examples generation

Consider a pair of columns $(S.sa, T.ta)$, exemplified in Figure 5.2, between which there is a matching correspondence. The objective is to identify which values in the source column sa and target column ta are likely to be equivalent. Starting from the existence of a matching relationship between the two columns, we can pair values based on their string representations and postulate that, for example, *730 Grand Ave* and *Grand Ave, Nr. 730* are sufficiently similar as to represent the same address.

More formally, assume we have two datasets, source S and target T. S has the attributes $(sa_1, ...sa_n)$, and T has the attributes $(ta_1, ...ta_m)$. We want values from S to be formatted as in T. Further, assume we have a function, $Match$, that returns a set of pairwise matches between the attribute names in S and T. Then, we can use Algorithm 5.2 to pair values from matching column sa_i and ta_j that are likely to be equivalent. The obtained pairs can then be used as examples for synthesis algorithms to generate transformation programs that will format values from sa_i as in ta_j.

Figure 5.2 depicts a pair of matching columns, each with 4 values, that we

#	sa
1	730 Grand Ave
2	93 Roland St
3	21 Duke Ave
4	44 Park Rd

#	ta
1	Robinson St, Nr. 14
2	Park Rd, Nr. 44
3	Grand Central Ave, Nr. 331
4	Grand Ave, Nr. 730

Figure 5.2: Examples of source and target records from a matching column pair.

Algorithm 5.2 Weighted examples discovery using string similarities.

```
1: function WEGSGEN(S,T)
2:     Egs ← {}
3:     for all ⟨sa, ta⟩ ∈ Matches(S, T) do
4:         Tok_s ← Tokenise(sa)
5:         Tok_t ← Tokenise(ta)
6:         Idx ← Index(Tok_s, Tok_t)
7:         Egs ← Egs ∪ WeightedPairing(Idx)
8:     end for
9:     return Egs
10: end function
```

will use throughout this section to describe the details of our approach. Notice that in a real–world case, the number of tuples of *sa* and *ta* would most likely be different and so would the ratio of shared values, i.e., the candidates for the resulting examples.

When applied to the column pair from Figure 5.2, Algorithm 5.2 has the following steps:

Tokenise, in lines 4 and 5: For each value in columns *sa* and *ta*, the $Tokenise$ method transforms the string representation of the value into an array representation, where each element of the array is a token as defined by the list of regular–expression–based primitives in Table 5.1. Note that Table 5.1 contains primitives similar to the ones used by *SynthEdit* to identify format descriptors. Moreover, the same primitives are at the core of *FlashFill*. Consequently, the technique described in this section is compatible with both synthesis algorithms. The last primitive types, *punctuation* and *space*, are used for separators only. The intuition is that a punctuation sign has small significance in determining the similarity of two values, e.g., the separators have a small weight in determining the equivalence of *24/04/1989* and *04.24.1989*. To illustrate this, Figure 5.3, describes the tokenised representations of values from Figure 5.2.

Index, in line 6: The tokens are used to create an inverted index $I(sa, ta)$

Primitive	Regex	Description
Alph	[*a–zA–Z*]+	One or more letters
LowAlph	[*a–z*]+	One or more lowercase letters
UppAlph	[*A–Z*]+	One or more uppercase letters
Num	[0–9]+	One or more digits
AlphNum	[*a–zA–Z*0–9]+	One ore more letters or digits
Punct	[.,;: /-_?!&$]+	One or more punctuation signs
Space	[\s]+	One or more spaces

Table 5.1: Regex primitives

#	sa - tokenised
1	[730,Grand,Ave]
2	[93,Roland,St]
3	[21,Duke,Ave]
4	[44,Park,Rd]

#	ta - tokenised
1	[Robinson,St,Nr,14]
2	[Park,Rd,Nr,44]
3	[Grand,Central,Ave,Nr,331]
4	[Grand,Ave,Nr,730]

Figure 5.3: Examples of source and target tokenised values from a matching column pair

for the pair of matching columns. For each token t, the inverted list $I[t]$ is a list of all values from *sa* and *ta* which contain token t. For example, if t =*Ave*, then $I[t]$ = [sa_i.*'730 Grand Ave'*,ta_j.*'Grand Ave, Nr. 730'*, ...], i.e., all the values from *sa* and *ta* containing token *Ave*. Figure 5.4 shows three index entries for tokenised values in Figure 5.3 with each row denoting an index entry where the *Token* column is the key and *Inverted list* is the list of values containing the token.

WeightedPairing, in line 7: Each source value in each index entry is paired, by Algorithm 5.3, with the most similar target value according to a confidence measure described below. For instance, if t = *Ave* and $I[t]$ = [*sa.'730 Grand Ave'*, *ta.'Grand Ave, Nr. 730'*, *ta.'Grand Central Ave, Nr. 331'*], then the returned pair would be (*sa.'730 Grand Ave'*, *ta.'Grand Ave, Nr. 730'*). Similar examples can be seen in Figure 5.5 for indexed values in Figure 5.4.

Weight: Each pair of values $(sa.x, ta.y)$, returned in the previous step, has a weight σ assigned to it that is computed by the *SimPairing* method (line 3 in Algorithm 5.3), according to Equation 5.1. We define θ, in Equation 5.2, as the overlap coefficient between two strings of characters which divides their intersection by the size of the smaller of the two sets; M as the number of tokens under which the pair is indexed; IDF_{rk}, in Equation 5.3, as the inverse document frequency (IDF) of a token r_k, under which the pair $(sa.x, ta.y)$ has

Token	Inverted list
Grand	[*sa.*'730 Grand Ave',*ta.*'Grand Central Ave, Nr. 331', *ta.*'Grand Ave, Nr. 730']
Ave	[*sa.*'730 Grand Ave',*sa.*'21 Duke Ave', *ta.*'Grand Central Ave, Nr. 331',*ta.*'Grand Ave, Nr. 730']
Park	[*sa.*'44 Park Rd',*ta.*'Park Rd, Nr. 44']

Figure 5.4: Examples of token–sharing buckets containing source and target values.

Algorithm 5.3 The WeightedPairing function of Alg. 5.2

```
1: function WeightedPairing(Idx)
2:     for all e ∈ Idx.entries do
3:         pairs ← SimPairing(e)
4:         maxPair ← null
5:         for all p ∈ pairs do
6:             if maxPair.weight < p.weight then
7:                 maxPair ← p
8:             end if
9:         end for
10:        Egs ← Egs ∪ {maxPair}
11:    end for
12:    return Egs
13: end function
```

been indexed, computed as a logarithmically scaled fraction obtained by dividing the total number of values from both columns by the number of values containing token r_k; and ϕ as a similarity measure, e.g., the Euclidean distance metric, of the two strings. In Equation 5.1, θ is used to penalise pairs with very dissimilar values, and IDF_{rk} is used to weight down pairs indexed under a very common token, e.g. *St, Ave.* For instance, the pair (*sa.'730 Grand Ave',ta.'Grand Ave, Nr. 730'*) in Figure 5.5 has been indexed under two tokens, *Grand* and *Ave.* The first entry will have a higher weight because there are four occurrences of *Ave* in Figure 5.2, and only three of *Grand.* Notice that the proposed weight is not intended to be a normalised similarity measure between two strings, but rather to identify the pairs of values that are more likely than others to be valid examples for synthesis algorithms.

$$\sigma = \theta(sa.x, ta.y) \times \max_{1 \leq k \leq M} (IDF_{rk} \times \phi(sa.x, ta.y)) \tag{5.1}$$

Token	**Value pairs**
Grand	[(*sa.*'730 Grand Ave',*ta.*'Grand Ave, Nr. 730')]
Ave	[(*sa.*'730 Grand Ave',*ta.*'Grand Ave, Nr. 730'), (*sa.*'21 Duke Ave',*ta.*'Grand Central Ave, Nr. 331')]
Park	[(*sa.*'44 Park Rd',*ta.*'Park Rd, Nr. 44')]

Figure 5.5: Examples of paired source and target values from a matching column pair.

$$\theta(X, Y) = \frac{|X \cap Y|}{\min(|X|, |Y|)} \tag{5.2}$$

$$IDF_{rk} = \log\left(\frac{N}{n_{rk}}\right) \tag{5.3}$$

To illustrate the behaviour of Equation 5.1 in practice, Table 5.2 shows examples of encountered cases and their respective coefficient values. In the table, the first column depicts the token (i.e., key) under which the pair from the second column has been stored in the index. The last three columns are the values for θ (i.e., overlap coefficient), IDF_{r_k} (i.e., the IDF of the key token) and ϕ (i.e., the string similarity). For both θ and ϕ, we only consider alphanumeric characters, i.e., non-separators. In computing ϕ, the key token is also removed from the strings, unless the result is the empty string. All coefficients are computed using lowercase strings.

Of particular interest in Table 5.2 is row 2. While the two strings represent the same address, the key token is very common, and, therefore has a small IDF. The same pair also appears in a bucket identified by the token *morgan*, which seems to be less frequent than *st*, and, hence has a higher IDF (note that in Equation 5.1 we only consider the maximum product of IDF and ϕ). The difference in string lengths in the third row leads to a relatively small ϕ, but the high overlap and token IDF compensate for that. The last two rows are false positives, i.e., pairs that should not be considered examples. Here, the dissimilarity of strings is evident from the low value of at least one coefficient. In general, high values for θ and/or ϕ denote strong evidence of validity, while a combination of low/mid-range values suggests false positives. Whenever the dissimilarity is not caught by Equation 5.1, we rely on Algorithm 5.4 to eliminate false positives, as described in the next section. The final example set, returned by Algorithm 5.3 for the two columns in Figure 5.2, includes three pairs, i.e. the set of distinct pairs in

Token r_k	**Value pair**	θ	IDF_{r_k}	ϕ
2016	*(2016-01-04,04/01/2016)*	1.0	1.27	1.0
st	*(71 Morgan St,Morgan St, Nr. 71)*	1.0	0.32	0.60
mckeon	*(Howard P. Mckeon (R-Calif), Mckeon)*	1.0	2.67	0.55
robert	*(Kruzel, Robert,Robert L Danley Jr)*	0.5	3.84	0.36
neal	*(Tatum O'Neal,neal_h_brian)*	0.33	2.55	0.29

Table 5.2: Weight examples

Figure 5.5, each one having an assigned weight that is used in the selection process described next.

5.5.2 Incremental examples selection

Algorithm 5.2 returns sets of examples that can be used for synthesising transformation programs, but, as with the FD–based technique, there is no guarantee that effective transformations will be produced from these examples. Furthermore, given its approximate nature, Algorithm 5.2 can generate a high number of example pairs, many of which are not valid example instances, i.e., do not represent similar concepts. This means that validation techniques such as the one proposed in Section 5.4.2 would fail. Finally, the large number of examples generated by Algorithm 5.2 can drastically increase the synthesis time. To address all these potential caveats, we describe a selection technique that aims to refine the example set produced by Algorithm 5.2 by selecting only the smallest subset for which there is evidence that it will produce effective transformations. We define the evidence as the effectiveness of the synthesis algorithms, using the selected subset, to produce a transformation program that will correctly transform the original set of examples generated by Algorithm 5.2.

More formally, given an example set $E = \{e_1, e_2, ..., e_n\}$ produced by Algorithm 5.2, with $e_i = (e_i^{in}, e_i^{out})$, the incremental selection technique, described in Algorithm 5.4, returns an example set $F = \{f_1, f_2, ...f_m\}$, with $F \subseteq E$ and $m \leq n$, such that when F is used as the input to a synthesis algorithm produces a set of transformation expressions $T = \{\tau_1, \tau_2, ..., \tau_p\}$ such that $\forall e_i \in E, \exists \tau_j \in T$ with $\tau_j\,(e_i^{in}) = e_i^{out}$.

The objective of Algorithm 5.4 is twofold: (i) to purge pairs of values returned by Algorithm 5.2 which are not equivalent; and (ii) to minimise the set of examples by purging redundant pairs (recall the representativity and non–redundancy

Algorithm 5.4 Examples selection

```
1: function EgsSelection(Egs)
2:     Egs ← Sort(Egs)
3:     minEgs ← InitEgs(Egs)
4:     p ← Synthesise(minEgs, Egs \ minEgs)
5:     while p ≠ ϵ do
6:         if GetFormat(p.in) ∉ GetFormats(Egs \ minEgs) then
7:             return REJECT Egs
8:         end if
9:         minEgs ← minEgs ∪ {p}
10:        p ← Synthesise(minEgs, Egs \ minEgs)
11:    end while
12:    return minEgs
13: end function
```

desiderata from Section 5.1). Regarding (ii), recall from Chapter 4 that a synthesised program is more likely to correctly transform all the test values if the example pairs cover all of the formats existing in the test data. But if there are too many example pairs covering the same format, this will exponentially increase the cost of the synthesis. In Algorithm 5.4, we consider two examples to be *redundant* if they cover the same format.

To illustrate the technique in practice, consider the pair of source and target columns depicted in Figure 5.6. The attribute *sb* contains full names of U.S. politicians, while *tb* contains person last names. The objective is to generate examples for synthesising a transformation program that extracts the last names in *sb*. For this purpose, we start by applying Algorithm 5.2, which produces a set of example pairs, nine of which are represented in Figure 5.7, sorted by their corresponding weights. Next, Algorithm 5.4 follows the steps described below:

Sort, in line 2: The set of example pairs returned by Algorithm 5.2 is sorted into descending order based on the weights. The result of this step is shown in Figure 5.7.

Initialise Examples, in line 3: The algorithm starts by selecting the example pair with the highest weight, *for each format present in the example set*. Specifically, the pairs depicted in Figure 5.7 describe three formats presented here using the notation introduced in Section 4.4.1, i.e., rows 1–3 (`A W A`), rows 4–7 (`A W A P A`), and rows 8-9 (`A W U P W A`). Note that, while the pairs with the same format are grouped together for clarity, this may not be the case in a real world scenario. Note also that both the punctuation and space count as token types

#	**sb**
1	Jim K. Lee
2	Billy W. Tauzin
3	Clarence Thomas
4	Joe Ackerman–Specter
5	Tommy Thompson
6	George W. Bush
7	Alphonso Jackson
n	...

#	**tb**
1	Benjamin-Stock
2	Bush
3	Millender-Cramer
4	Jackson
5	Thomas
6	Lee
7	Ackerman
m	...

Figure 5.6: Examples of source and target records from a matching column pair.

#	**sb**	**tb**	**W**
1	Condoleezza Rice	Rice	2.21
2	Michael Leavitt	Leavitt	2.16
3	Clarence Thomas	Thomas	2.09
4	Juanita Millender–Cramer	Millender-Cramer	1.76
5	Joe Ackerman–Specter	Ackerman	1.34
6	Tammy Liu–Vitter	Liu-Vitter	1.32
7	Walter Ben–Stock	Benjamin-Stock	0.84
8	George W. Bush	Bush	0.81
9	Joe K. Pitts	Pitts	0.79

Figure 5.7: Examples of source and target records aligned by Algorithm 5.2.

rather than separators (as was the case for the tokenization step in the previous section) because the goal in this case is to identify covered format descriptors. The three examples from Figure 5.8 have the highest weights for their respective formats.

Synthesise, lines 4 and 10: This method, illustrated in Algorithm 5.5, takes as input a set of example pairs (viz., the pairs with the highest weights as returned by the *InitEgs* method), and a set of test pairs (viz., the pairs with lower weights than the ones considered as examples). Then, a transformation program is synthesised using *SynthEdit* on the example set (line 2 in Algorithm 5.5). The resulting program is then tested against each pair from the test set. If the result of a transformation is different from the expected test output (line 4 in Algorithm 5.5), and if there has not been a previous test value describing the same format which was correctly transformed (line 5 in Algorithm 5.5), then that failing pair is returned. The $IsFirst(p)$ method at line 5, checks if pair p has the highest weight for the format it describes.

Source Value	Target Value
Condoleezza Rice	Rice
Juanita Millender-Cramer	Millender-Cramer
George W. Bush	Bush

Figure 5.8: Pairs of source–target value pairs in the initialisation phase.

Algorithm 5.5 The *Synthesise* method of Alg. 5.4

```
1: function SYNTHESISE(Egs,TestEgs)
2:     τ ← FlashFill(Egs)
3:     for all p ∈ TestEgs do
4:         if τ(p.in) ≠ p.out then
5:             if IsFirst(p) then
6:                 return p
7:             end if
8:         end if
9:     end for
10:    return ϵ
11: end function
```

Increment, in line 9: Every time the *Synthesise* method returns a failing pair, the pair is added to the previous set of examples. The intuition is that the initial set of examples did not cover the format described by the failing pair, therefore, by taking it as an example, another format is covered.

Halting Condition, in line 6: Algorithm 5.4 stops when there are no more failing test pairs, or when the halting condition is met, viz., all the pairs describing a failing format have been used as examples. This means that there are not enough examples for that format, the algorithm returns and the example set is rejected.

To continue with the example in Figure 5.7, when *Synthesise* is called in line 4 in Algorithm 5.4, the pairs in Figure 5.8 are used as the example set to synthesise a transformation program τ, which is then tested against the rest of the values in Figure 5.7. Notice that for row 5 of Figure 5.7, the transformation synthesised from the examples depicted in Figure 5.8 fails, i.e., $\tau(\textit{Joe Ackerman–Specter}) \neq \textit{Ackerman}$, because *Ackerman* is not the last name. The pair at row 4 has the highest weight from the test pairs describing the same format f, meaning that f has not been covered by the previous example set, so the pair is added to the example set at line 9 in Algorithm 5.4, and a new iteration starts.

Of particular interest is row 7 in Figure 5.7. Notice that, given the values exemplified so far, there is no transformation program that can transform *Walter*

Ben-Stock to *Benjamin-Stock*. If the previous pair (row 6), which has a higher weight, is correctly transformed, i.e., τ(*Tammy Liu-Vitter*) =*Liu-Vitter*, then the pair in row 7 is ignored as a *false-positive* that was returned by Algorithm 5.2, and the algorithm continues (towards a successful result in this case). This result includes rows 1, 4, 5, and 8 of Figure 5.7, i.e., the minimal set of examples that produces a transformation program that correctly transforms the rest of the pairs. Otherwise, if τ(*Tammy Liu-Vitter*) $\neq$*Liu-Vitter* and τ(*Walter Ben-Stock*) $\neq$*Benjamin-Stock*, the halting condition is met, i.e., there are no more pairs for the format described by these two failing pairs, and the example set generated by Algorithm 5.2 is rejected.

We argue that there is no need to perform 10–fold cross–validation on the returned set of examples, as was done in Section 5.4.2. The intuition is that if Algorithm 5.4 returns a valid set of examples, there is enough evidence to consider that a synthesis algorithm can return a transformation program that correctly transforms the source values covered by the seen examples. Furthermore, if it were applied on the output of Algorithm 4, cross–validation would fail because it requires at least two example pairs per format in the candidate example set. In other words, the validation technique requires *redundant* example candidates, which is exactly what Algorithm 5.4 tries to avoid.

5.6 Evaluation

In this chapter, we evaluate the hypothesis that the process of transforming formats in a data lake using *SynthEdit* (discussed in Chapter 4) can be automated by replacing the user–provided examples with example generation algorithms presented above. In a scenario in which information from multiple, heterogeneous data sets is to be integrated, it falls on data scientists to identify values that need to be transformed in a source dataset, and their corresponding versions in a target dataset. Our approach removes the user from this process by automating the identification of examples for use by program synthesis.

To illustrate and evaluate the methods proposed in this chapter, firstly, we use open government Web–data to assess the effectiveness of Algorithm 5.1 at identifying candidate column pairs that can contribute input–output examples when their values are aligned. We do not perform this experiment for Algorithm 5.2 because this would be equivalent to evaluating the effectiveness of the *Match*

function which is not the focus of this chapter.

Secondly, we evaluate the effectiveness of the validation technique discussed in Section 5.4.2, in conjunction with Algorithm 5.1 for identifying problematic column pairs. We do not perform this experiment for Algorithm 5.2 because, as mentioned before, the cross–validation technique is not applicable in the case of the latter algorithm.

Next, we evaluate the effectiveness of using *SynthEdit* to synthesise transformations informed by examples generated automatically using Algorithms 5.1 and 5.2. We also use Algorithm 5.4 to decide on the most informative subset of example instances, showing that the example selection technique is effective in reducing the number of examples.

Finally, we evaluate the efficiency of the algorithms discussed in this chapter when applied on datasets of various sizes.

5.6.1 Data repositories used in evaluation

The motivation for the techniques explored in this chapter is the desire to remove the need for manual specification of examples when using synthesis algorithms. To asses the efficiency and the effectiveness of the methods we selected a collection of 9 source–target pairs of related datasets from Web–based data lakes, e.g., `data.gov.uk`, `data.ny.gov`, etc.. The source dataset contains values to be transformed and the target dataset provides values in the desired format. The datasets have been chosen to reflect multiple transformation types accommodated by *SynthEdit*, as described in Chapter 4, e.g., punctuation removal, (sub)token extraction/deletion/permutation/insertion, case alteration, or combinations of two or more such transformation types.

Table 5.3 describes the domain, location, cardinality and arity of the source and target datasets used in experiments.

Note that only the first five pairs of datasets are used to evaluate the FD–based examples generation approach from Section 5.4, since those were the datasets for which the state–of–the–art data profiling algorithms (e.g., HyFD [PN16]) were successful.

Each pair of source–target datasets contributes one or more pairs of matching columns that contain overlapping instance values with dissimilar format representations. We delegate the $Match$ function used in the proposed algorithms to COMA 3.0 Community Edition (`sourceforge.net/projects/coma-ce/`) [ADMR05],

Domain	Source/Target URLs	Cardinality	Arity
Food Hygiene	ratings.food.gov.uk	12,323	7
	data.gov.uk	61	10
Lobbyists	data.cityofchicago.org	7,542	12
	data.cityofchicago.org	210	19
Doctors/Addresses	www.nhs.uk	7,696	19
	data.gov.uk	17,867	3
Building Permits	app.enigma.io	10,000	13
	data.cityofchicago.org	1,245	13
Citations	dl.acm.org	1,072	16
	ieeexplore.ieee.org	2,000	11
US Politicians	opensecrets.org	620	9
	govtrack.us/congress	11,870	10
Restaurants	app.enigma.io	25,000	19
	data.ny.gov	21,000	18
Movies	imdb.com	75,000	4
	imdb.com	15,000	5
Employment	data.cityofchicago.org	31,000	8
	data.cityofchicago.org	12,000	7

Table 5.3: Data sources used in evaluation

a state–of–the–art schema matching tool. Note that only the generation of examples is claimed as contribution in this chapter and, hence, evaluated in this section. Although they are relevant in practice, we do not separately evaluate the off-the-shelf components we use, viz., COMA and HyFD. They have been evaluated directly by their authors and/or in comparative studies (see [ADMR05] and [PN16]).

5.6.2 Reported measures

We report on the following characteristics of the proposed examples generation techniques:

1. **Effectiveness**

   (a) of Algorithm 5.1 in identifying candidate column pairs, i.e., pairs where (i) the members are in a matching relationship, (ii) represent dependants in two functional dependencies, and (iii) contain overlapping values that denote the same real–world concepts. To this end, we

report the *precision* and *recall* of the algorithm applied on datasets where there are known suitable candidates.

(b) of the transformation programs synthesised by *SynthEdit* from the example instances produced by Algorithms 5.1 and 5.2. To this end, we report the *precision* and *recall* of the result of the transformations synthesised from the generated examples, applied on previously unseen values from the source dataset.

2. **Efficiency** in generating examples given various dataset pairs. To this end, we report the running times of Algorithms 5.1, 5.2, and 5.4 when applied to datasets from various domains. In the case of Algorithm 5.4, we only run it on the results of Algorithm 5.2 since the output of the latter often includes the output of Algorithm 5.1. In practice, the example selection technique can be applied on the result of Algorithms 5.1 or 5.2.

3. We also analyse the effectiveness of the cross–validation technique in identifying problematic column pairs, i.e., pairs that cannot be taken as examples for synthesising transformations. In this case, we report the fraction of candidate column pairs returned by Algorithm 5.1 that are valid by a 10–fold cross validation process, and discuss the causes of failure.

For the purposes of computing precision and recall for (**1 (a)**) above we define:

- **TP**. A pair of columns that share values representing the same concept and that is present in the output of Algorithm 5.1 is a true positive.
- **FP**. A pair of columns with values representing different concepts and that is present in the output of Algorithm 5.1 is a false positive.
- **FN**. A pair of columns that share values representing the same concept and that is not part of the output of Algorithm 5.1 is a false negative.

A ground truth was manually created for the domains in Table 5.3, in which a column from the source and a column from the target create a pair in the ground truth if they represent the same concept.

Similarly, following the definitions of precision and recall of format transformations introduced in Section 4.6.2, for the purposes of computing precision and recall for (**1 (b)**) above, we define:

- **TP**. An input string, not included in the examples from which a transformation program has been synthesised, that is correctly transformed is a true positive.
- **FP**. An input string, not included in the examples from which a transformation program has been synthesised, that is incorrectly transformed by that transformation program is a false positive.
- **FN**. An input string, not included in the examples from which a transformation program has been synthesised, that is left unchanged by that transformation program is false negative.

A ground truth was manually, in which a random selection of 300 instance values from the source column, not part of the generated examples, were transformed and the results manually evaluated.

As usual, precision p is defined by:

$$p = \frac{TP}{(TP + FP)}$$

Recall r is defined by:

$$r = \frac{TP}{(TP + FN)}$$

All experiments have been run on a 2.60 GHz Intel Core i7-4720HQ CPU with 12 GB RAM.

5.6.3 Effectiveness of FD based scheme

In this section we evaluate the results of Algorithm 5.1 using an off–the–shelf matcher to identify column level matches between a source dataset and a target dataset, and an off–the–shelf profiling tool to identify functional dependencies (FDs). We then use these two types of relationships to generate input–output examples for the required transformations using Algorithm 5.1. The resulting sets of examples are then validated using a 10–fold cross–validation process, as described in Section 5.4.2. Note that, in practice, the first two steps mentioned above are necessary only if the input data doesn't explicitly contain pre–discovered matching correspondences or functional dependency candidates, e.g. a relational database that contains explicit FDs for its relations. Finally, the validated pairs

are input to a synthesis algorithm and the resulting transformations are applied on new source values.

Experiment 5.1. ***Can Algorithm 5.1 identify candidate column pairs?***

Representative examples of candidate columns pairs that are identified by Algorithm 5.1, together with a source and corresponding target instance values, are given in Table 5.4. The last column illustrates the number of unique example pairs generated by Algorithm 5.1 for each case.

The results of this experiment are presented in columns *Precision* and *Recall* of Table 5.5, for the data sets in Table 5.4. In general, both precision and recall are high. In the case of the Building Permits domain, there are three attributes in each dataset representing a cost (see row 9 in Table 5.4 for an example). Although there are nine possible source column–target column alignments, Algorithm 5.1 was able to identify the correct ones and returned no false positives. Food Hygiene precision is *0.86* due to a bad match returned by COMA. An example of this match is on the second row of Table 5.4. The two columns have the same name (*AddressLine*), but different semantics. The precision and recall for Citation are reduced by one bad match returned by COMA (represented in the last row of Table 5.4) and two missing matches (i.e. two pairs of columns that were not reported as matches by COMA).

Experiment 5.2. ***Is the validation process successful at identifying problematic column pairs?***

A column pair is problematic if we cannot synthesise a suitable transformation. Note that in Experiment 5.1 we measured the effectiveness of Algorithm 5.1 at identifying correct column pairs. Here we check whether the alignment of values, i.e. the result of the selection in lines 8–10 in Algorithm 5.1, denotes suitable examples for synthesising a correct transformation. To evaluate this we run a 10–fold cross validation task on each pair of columns from Experiment 5.1. We now discuss the candidate pairs for which validation has failed, and have the cases for which validation has passed to be discussed in the next experiment. A pair of columns is considered to pass the validation step if all transformations are correct across all iterations.

The fraction of the candidate column pairs that pass validation is reported in the last column of Table 5.5. Column pairs have failed validation for the following reasons: (i) the matched columns have different semantics, and thus incompatible

#	Domain	Source semantics	Source example	Target semantics	Target example	egs.
1	Food Hyg.	Rating Date	2015-12-01	Rating Date	01/12/2015	57
2	Food Hyg.	Building Name	Royal Free Hospital	Street	Pond Street	59
3	Food Hyg.	Business Name	Waitrose	Business Name	Waitrose	60
4	Lobby	Person Name	Mr. Neil G Bluhm	Person Name	Bluhm Neil G	206
5	Lobby	Phone	(312) 463-1000	Phone	(312) 463-1000	191
6	Docs./Addrs.	Address	55 Swain Street, Watchet	Street	Swain Street	41
7	Docs./Addrs.	City	Salford	City	Manchester	28
8	Build Perm.	Address	1885 Maud Ave	Address	Maud Ave, Nr. 1885	26
9	Build Perm.	Cost	6048	Cost	$6048.00	22
10	Build Perm.	Issue Date	2014-06-05	Issue Date	06/05/2014	26
11	Citations	Author Names	Sven Apel and Dirk Beyer	Author Names	S. Apel; D. Beyer	56
12	Citations	Date	" "	Date	2-8 May 2010	32
13	Citations	Year	2011	Year	2011	56
14	Citations	Num. of pages	10	Start page	401	56

Table 5.4: Resulting column-pairs with example values.

values, for which no transformation can be produced, which is the case for two of the eight column pairs that fail validation (for example see row 2 and row 14 in Table 5.4); (ii) the matched columns have the same semantics, but *SynthEdit* has failed to synthesise a suitable transformation, which is the case for two of the column pairs that fail validation (as an example, consider the lists of author names from row 11 in Table 5.4); (iii) there are issues with the specific values in candidate columns, which is the case for four of the column pairs that fail validation (as an example, consider the missing information from row 12 in Table 5.4 the and inconsistent values in row 7 in Table 5.4).

It is important to note that, in practice, the effectiveness of the off–the–shelf tools that we used here can be impacted by characteristics of the data such as the ones exemplified above. This evaluation shows that the validation method we employ is able to filter out example sets that otherwise would produce invalid transformations or no transformations at all.

Experiment 5.3. ***Does the synthesis algorithm produce effective transformation programs from the example pairs generated by Algorithms***

Domain	Candidates	Precision	Recall	Valid
Food Hyg.	6	0.83	1.00	5/6
Lobby	9	1.00	1.00	9/9
Docs./Addrs.	2	1.00	1.00	1/2
Build Perm.	12	1.00	1.00	12/12
Citations	7	0.86	0.75	1/7

Table 5.5: Results of Experiments 5.1 and 5.2

5.1?

To evaluate this, in Table 5.6 we report the precision and recall of the transformations synthesised from the examples generated in the previous experiments. The missing row numbers are the cases in Table 5.4 that failed validation.

Of the 28 validated cases from Table 5.5, all but six are identity transformations, i.e. the source and target values are the same (e.g., rows 3, 5 and 13 in Table 5.6). This can often happen in practice. For instance, row 13 represents the year for a publication which is most commonly represented as a four–digit number. In such cases, *SynthEdit* was able to identify that the transformation is only a copying operation. Of the six cases where the values are modified, the precision and recall are both *1.0* in two cases (rows 1 and 10 in Table 5.6), confirming the results of the evaluation performed in Chapter 4 according to which *SynthEdit* is effective in such simple cases. For row 8, the transformation is more complicated given that the street name does not always have a fixed number of words, and the street number can have a different position. In this case, Algorithm 5.1 was able to identify enough examples to cover most of the existing formats. There were problems with the transformations generated in three cases. For row 4, a few source values do not conform to the usual pattern (e.g. the full stop is missing after the salutation). For row 9, not all source values are represented as integers, giving rise to incorrect transformations. For row 6, similarly to row 9, the examples do not cover all the relevant address formats, i.e. 41 examples are used to synthesise a program to transform a rather large number of values (approx. 7,700).

In summary, the technique evaluated above proves to be effective in scenarios where certain conditions are met. The most important of these is that the source and target contain overlapping information on some of the tuples, i.e. the left–hand sides of the functional dependencies used in Algorithm 5.1. Another important condition is for *SynthEdit* to be able to synthesise transformations

#	Source semantics	Source example	Target semantics	Target example	Prec.	Rec.
1	Rating Date	2015-12-01	Rating Date	01/12/2015	1.0	1.0
3	Business Name	Waitrose	Business Name	Waitrose	1.0	1.0
4	Person Name	Mr. Neil G Bluhm	Person Name	Bluhm Neil G	0.91	0.90
5	Phone	(312) 463-1000	Phone	(312) 463-1000	1.0	1.0
6	Address	55 Swain Street, Watchet	Street	Swain Street	0.57	1.0
8	Address	1885 Maud Ave	Address	Maud Ave, Nr. 1885	0.89	1.0
9	Cost	6048	Cost	$6048.00	0.95	1.0
10	Date	2014-06-05	Date	06/05/2014	1.0	1.0
13	Year	2011	Year	2011	1.0	1.0

Table 5.6: Results of Experiment 5.3

from the pairs of generated examples. The evaluation shows that as long as these conditions are met, we can dispense with user intervention in this stage of the cleaning process by synthesising and applying transformations automatically.

5.6.4 Effectiveness of weighted scheme

In this section we evaluate the effectiveness of Algorithms 5.2 and 5.4 using all nine pairs of datasets from Table 5.4. As was the case with the FD–based proposal, we hypothesise that the task of providing examples by the user, which is often required in current data format transformation tools, can be partly replaced by Algorithms 5.2 and 5.4. We use the same complementary tools as before, i.e. COMA and HyFD. We analyse the effectiveness of Algorithm 5.4 when applied to the output of a previous run of Algorithm 5.2. Note that, in practice, Algorithm 5.2 should not be used without a subsequent selection of its output, i.e., Algorithm 5.4, since, subject to the degree of similarity between the string values of matching pairs of columns, many of the generated examples may not reflect a valid transformation: the source and target strings may be similar to some degree, but not represent the same concept. The ethos of Algorithm 5.4 is to identify and eliminate such examples, together with other redundant instances.

Experiment 5.4. ***Does the synthesis algorithm produce effective transformation programs from the example pairs generated by Algorithms 5.2 and 5.4?***

#	**Source sem.**	**Target sem.**	**Source**	**Target**	**Alg.2**	**Alg.4**	**Prec**	**Rec**
1	Date	Date	2015-12-01	01/12/2015	57	1	1.00	1.00
2	Pers. name	Pers. name	Mr. Neil G Bluhm	Bluhm Neil G	206	25	0.64	0.75
3	Addr.	Addr.	55 Swain Street, Watchet	Swain Street	2819	N/A	N/A	N/A
4	Addr.	Addr.	1885 Maud Ave	Maud Ave, Nr. 1885	1051	14	0.86	1.00
5	Cost	Cost	6048	$6048.00	257	3	0.98	1.00
6	Pers. name	Pers. name	Pete Stark (D-Calif)	Stark	473	34	0.62	1.00
7	Pers. name	Pers. name	Pete Stark (D-Calif)	Pete	279	25	0.70	1.00
8	Pers. name	State	Pete Stark (D-Calif)	Calif	23	6	0.91	1.00
9	Addr.	Addr.	41 Page Avenue, Delhi	Page Avenue	1401	N/A	N/A	N/A
10	Date	Date	2014-06-23 00:00:00+00	06/23/2014	1148	1	1.00	1.00
11	Pers. name	Username	Jim Brown	jim_brown	1872	N/A	N/A	N/A
12	Pers. name	Username	Perry Lang	lang.perry	1680	17	0.71	1.00
13	Pers. name	Pers. name	OUTTEN, MIA G	Mia G Outtent	8903	22	0.87	0.98
14	Addr.	Addr.	2440 N Cannon Drive	Cannon Dr	21	4	0.80	1.00

Table 5.7: Results of Experiment 5.4

For each matching column pair returned by the *Match* function, we first run Algorithm 5.2, the result is then passed to Algorithm 5.4, then, using the output from the latter and *SynthEdit*, a transformation program is synthesised and applied to 300 random source values that have not been used in the examples. We manually evaluate the output of each transformation and report the precision and recall for each domain, as shown in Table 5.7. The table shows fourteen cases of transformations, together with the semantics of source and target values, an example instance, the number of examples generated by Algorithm 5.2, the number of examples selected by Algorithm 5.4, and the precision and recall of applying the transformations synthesised from the selected examples.

Of the fourteen cases exemplified in Table 5.7, Algorithm 5.4 rejected three example sets: rows 3, 9, and 11. For rows 3 and 11, the halting condition was met, while in the case of row 9, *SynthEdit* was unable to synthesise a transformation

program due to a high degree of heterogeneity in the format representation of values. For the rest of the matching column pairs, recall fell below *0.98* only in the case of row 2. This was due to the fact that person names often contain common tokens which can lead to highly weighted false positives (i.e., pairs of strings that, although similar to some extent, do not depict valid transformations) being returned by Algorithms 5.2 and 5.4. This is also true for precision. In fact, all of the cases with precision lower than *0.80* are of person names that exhibit the same characteristic. In the case of precision, the lower values were a consequence of the ambiguity challenge discussed in Section 4.5 of Chapter 4, i.e., the generated examples were not descriptive enough to clearly distinguish a single transformation from all the possible ones.

Notice that Table 5.7 shows large differences between the number of example pairs generated by Algorithm 5.2 and the pairs selected by Algorithm 5.4 across most of the cases exemplified. This suggests that many of the pairs generated by Algorithm 5.2 are covering a relatively small number of formats. For instance, in the case of row 1, all source values of the 57 example pairs generated by Algorithm 5.2 are describing a single date format. Therefore, Algorithm 5.4 returned a single pair.

Consistently with results discussed in Section 4.6, the recall values in Table 5.7 are close to 1.00. This means that *SynthEdit* transformed most of the strings on which it has been tested. It follows that Algorithm 5.2 successfully identified example instances for most of the format representations in the source columns, and that Algorithm 5.4 successfully selected representative example instances for each such format.

5.6.5 Efficiency evaluation

In this section we provide a comparative study of the two schemes for generating examples presented in this chapter by analysing the computational cost of each technique, i.e., Algorithms 5.1, 5.2, and 5.4, the computational cost of transformation synthesis using examples generated by each method, and the consequences (if any) of removing examples through Algorithm 5.4.

Experiment 5.5. ***How efficient is the generation of examples?***

Table 5.8 shows results on cases from Table 5.4 for which examples have been generated using both schemes. The other cases from Table 5.4 have either been

#	**Source/ Target card**	**Alg.5.1**	**Alg.5.2**	**Alg.5.4**	**Alg.5.1(s)**	**Alg.5.2(s)**	**Alg.5.4(s)**
1	12,323/61	57	57	1	0.006	0.2	0.61
2	7,542/210	206	206	25	0.005	0.15	9.9
4	10,000/1,245	26	1,051	14	0.006	12.5	5.1
5	10,000/1,245	22	257	3	0.005	0.15	0.76

Table 5.8: Results of Experiment 5.5

used only with the weighted technique because the conditions for the FD–based approach were not met, or the generated examples failed the validation step in the FD–based approach. We report the row identifier in Table 5.4 on the first column, while the rest of the columns represent the source/target cardinality, the number of example pairs returned by Algorithm 5.1, the number of example pairs returned by Algorithm 5.2 and Algorithm 5.4, and the time in seconds required to generate the examples by each algorithm, resp. Note that we only analyse cases for which the validation stage has been passed (for the FD–based scheme) and for which the selection stage returned at least one example pair (for the weighted scheme).

The complexity of Algorithm 5.1 is dominated by the number of matching relationships of the source and target and by the number of functional dependency candidates for each dataset (lines 3 and 5-6). In practice, the number of matches and FDs tends to be small, meaning that the time required to run Algorithm 5.1 is dominated by the *SQL select* query at lines 8–10. Conversely, the complexity of Algorithm 5.2 is dominated by the number of records of the two datasets, i.e., the more instance values two columns have, the more entries the index will contain, which translates into more pairwise string similarity comparisons. The first two rows of Table 5.8 show that if we do not use functional dependencies and resort to string matching, generating the same number of examples takes 30 times longer. Often (e.g., last two rows), the pairs generated by Algorithm 5.2 contain false positives, i.e. example instances for which the two strings do not represent the same thing. To mitigate such cases we employ Algorithm 5.4, the complexity of which is dominated by *SynthEdit*. The number of seconds required to refine the examples generated by Algorithm 5.2 using Algorithm 5.4 is reported in the last column of Table 5.8. The selection of examples for row 2 took longer than for row 4 because of a greater format heterogeneity in the case of the former, which leads

to multiple iterations for Algorithm 5.4, i.e., multiple runs of *SynthEdit*.

Experiment 5.6. ***How efficient is the synthesis process given the generated examples?***

The benefit of minimising examples can be observed in Figure 5.9, where the time it takes *SynthEdit* to synthesise a transformation program, is reported. We compare synthesising transformations from the examples generated by Algorithm 5.1, on the one hand, and by Algorithm 5.2, refined by Algorithm 5.4, on the other. Once again, the results are consistent with those obtained in Chapter 4. It can be observed that eliminating redundant example instances substantially improves synthesis time, especially when there are many examples per format: the fact that for row 2 Algorithm 5.4 reduced 206 examples to 25 suggests that there are almost 9 examples per format.

With respect to the transformations synthesised, for cases 1 and 2 in Table 5.8, the transformation resulting from the examples returned by Algorithm 5.1 was equivalent to the transformation obtained from the examples returned by Algorithm 5.4. For the last two cases, the transformation synthesised from the examples returned by Algorithm 5.4 was more complex: it included more conditional branches. This is unsurprising, as the number of candidate pairs generated by Algorithm 5.2 contains more formats that the ones from Algorithm 5.1.

In summary, the experiments in this section, together with the ones in Section 5.6.3, show that the FD–based approach efficiently generates more effective example pairs, as long as FDs can be discovered. This approach can lead to an increase in synthesis cost due to redundant example instances. The problem of too many examples instances that cover the same transformation cases can be solved in practice by employing the example selection technique in Algorithm 5.4. For cases where FDs are not available, the weighted solution can prove viable at the expense of effectiveness. For example, row 2 of Table 5.8, where the number of example instances has been reduced from 206 to 25, represents a case where the weighted scheme proves less effective than the FD–based scheme: the transformation precision decreased from 0.91 when 206 example instances were used, to 0.64 when only the 25 instances selected by Algorithm 5.4 were used. A closer investigation showed that, through selection, only one example instance per format representation survived, but, in some cases, *SynthEdit* needed more examples per format representation to overcome the ambiguity issue discussed in Section 4.5.

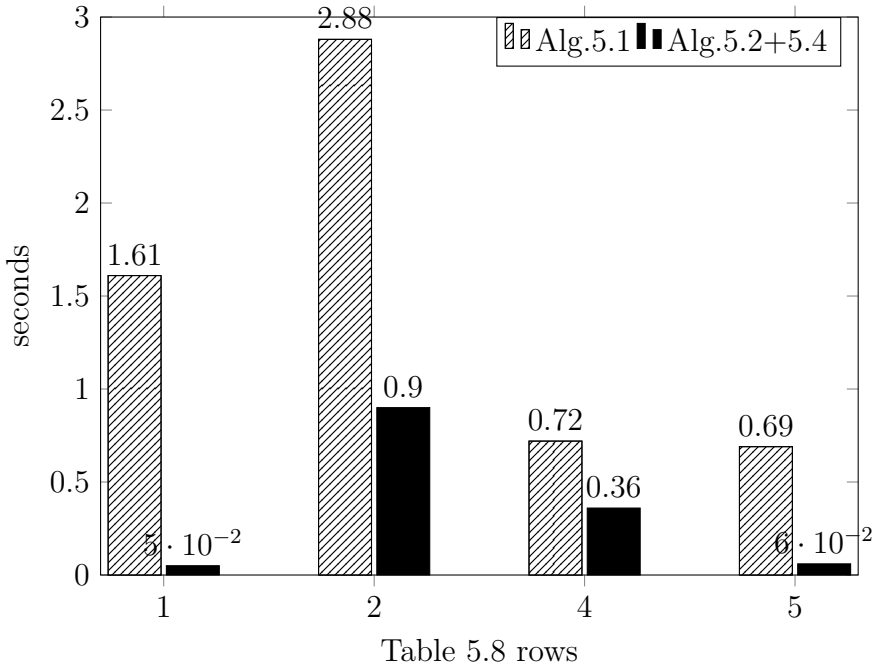


Figure 5.9: Average synthesis time of SynthEdit measured on 100 runs.

5.7 Summary and conclusions

This chapter has focused on discussing procedures that enable the application of format transformations (Chapter 4) on data lakes. Specifically, these proposals addresses the assumption of Chapter 4 that the provision of examples for program synthesis is performed by the user. In large–scale scenarios, where there are many datasets to transform, or where the instance values of a column to be transformed exhibit multiple format representations that would each require at least one example instance, we argued that such manual intervention is too costly. Consequently, in this chapter, we proposed the automation of the example provisioning task and only relying on the user to provide one or more datasets where examples should be searched for. These datasets, which in our approach act as the targets, can, for example, be part of the *data context*. This concept was introduced in Section 2.3 of Chapter 2 as a collection of data values that supports the wrangling process by stipulating the domain of the data that is prepared for analysis, or that exemplifies the entities that are expected to appear in the resulting dataset, together with their representation.

We proposed two approaches for the automatic generation of examples with the following characteristics:

- The first approach employs matching relationships and hypothesised functional dependencies to align the instance values of matching source and target columns. The result can be construed as example instances for synthesising transformation programs that can normalise the representation of the rest of the values of the source. The use of postulated FDs, followed by 10–fold cross validation, leads to a set of valid examples w.r.t. the transformations needed, therefore addressing the validity desideratum from Section 5.1.

- When FDs are not identifiable, a second approach relies on string similarities to perform the same type of value alignment between source and target instance values.

- Given the approximate nature of string similarity pairing, a selection method was proposed to filter out inconsistent and redundant pairs. The result is a subset of source–target value alignments that cover most of the format representations present in the unfiltered version. Instead of a cross-validation step, the selection method uses a synthesizer to ensure the validity of the resulting set of examples, i.e., that a transformation program can be synthesised from the resulting examples. In addition, the selection algorithm is designed with the representativity and non–redundancy desiderata in mind, i.e., it searches for representative examples for each format descriptor encountered in the unfiltered set of examples and it aims to eliminate redundant example instances.

- Lastly, both approaches perform automatic examples generation, and, therefore, address the need for automation expressed in the last desideratum of Section 5.1.

The experiments conducted on data selected from various Web–based data lakes showed that, given a source S and target T datasets, the FD–based examples generation approach is efficient and effective when the following conditions are met:

- $\exists\, S.sa, T.ta$, two columns from S and T, respectively, such that $S.sa$ *matches* $T.ta$;

- $\exists\, S.sx, T.tx$, two columns from S and T, respectively, such that $S.sx$ *matches* $T.tx$ and $S.sx \rightarrow S.sa$ and $T.tx \rightarrow T.ta$;

- $S.sx$ and $T.tx$ share some values;

- The values of $S.sa$ and $T.ta$ are amenable to homogenisation, i.e., their values are regularly structured.

When FDs are not available, the second approach, based on string–similarities, exhibited comparable effectiveness at the expense of efficiency, when compared with the FD–based proposal. Moreover, the weighted proposal requires the existence of an example selection step that proved capable of reliably reducing the set of examples instances to a minimal subset for which there is enough evidence to inform the synthesis of a transformation program.

Overall, the two examples generation approaches presented in this chapter, prove effective in alleviating the burden of manual example provisioning for performing format transformations, thereby meeting objective O_3 of Chapter 1.

Chapter 6

Enabling data profiling in data lakes

Chapters 3–5 have described our research contributions to the vision of cost–effective data wrangling. However, and as discussed in Chapter 2, cost–effective data wrangling comprises more tasks than the ones covered in this thesis. For instance, the transformed and cleaned datasets have to be merged into a unified view through data integration before being delivered to consumer applications. Such tasks assume knowledge about the structural relationships intra– and inter–data sets.

Given a collection of datasets, specialised data profiling algorithms have been devised to identify semantic relationships (e.g., functional dependencies [Cod71] and inclusion dependencies [CFP82]) when they exist. While functional dependencies can be an important input to schema normalisation and to data cleaning (as shown in Chapter 5), our focus in this chapter is on inclusion dependencies, which are essential in data integration. When data integration is performed on collections of heterogeneous datasets (e.g., data lakes), the assumptions made by algorithms for inclusion dependency (IND) discovery do not always hold. Thus, our aim in this chapter is to contribute to enabling the profiling of data lakes, with a focus on INDs discovery. We approach this task by combining elements specific to data discovery and format transformation, thus showing that the two main topics of this thesis, which are valuable in themselves, can also be used together to support another task of data wrangling.

Our aim in this chapter can be summarised as follows:

Given a collection of datasets with potentially differently formatted values, identify inclusion dependencies between attributes from distinct datasets.

State–of–the–art INDs discovery algorithms (e.g., BINDER [PKQN15], FAIDA [KPD+17], SINDY [KPN15]) on their own cannot always achieve our declared aim, since such algorithms assume normalised representations of instance values and string equality between the values of the members of an IND. One possible solution to this problem is described in Section 6.4. Prior to that, in Section 6.1, we present our motivation and our desiderata for the proposed solution. We discuss the technical background in Section 6.2, and define the boundaries of our approach in Section 6.3. We close the chapter with an empirical evaluation of the proposed technique in Section 6.5, before presenting the main conclusions in Section 6.6.

6.1 Motivation and desiderata

In a nutshell, an inclusion dependency (IND) between two attributes implies that for all database states, the set of values of one attribute must be a subset of the values of another attribute, often from a different table. The task of identifying INDs often implies two challenges, viz., (i) how to contend with inconsistencies in the format representation of instance values that are equivalent in meaning, and (ii) how to remain efficient given the ever increasing size of the datasets, in terms of number of records as well as attributes. Most of the state–of–the–art proposals for IND discovery focus on the second challenge, and seek to devise scalable techniques that cover the potentially huge space of IND candidates. In so doing, they assume that the first challenge does not arise, i.e., they assume homogeneous representations across all instance values. Consequently, IND discovery algorithms are not easily applicable on data lakes characterised by format inconsistencies between similar instance values.

Motivated by the need for data profiling in wrangling data from data lakes, we propose one approach for enabling the use of IND discovery algorithms on data with inconsistent format representations. To this end, we rely on the contributions reported for dataset discovery, in Chapter 3, and for automatic format

transformation, in Chapters 4 and 5, and seek to devise a prepossessing step to the IND discovery algorithms with the following properties:

- It should enable the discovery of new INDs, previously missed by the used algorithms due to inconsistencies in the format representations of instance values.
- It should enable the identification of most, if not all, shared values between the members of an IND, irrespective of their format representation.

The two desiderata above are motivated by the need for accuracy in identifying all the shared instance values between the composing attributes of an IND candidate: (i) if we are looking for full inclusion of the set of values of the first attribute in the set of values of the second attribute, then missing a single shared value due to a different representation amounts to discarding the IND candidate; (ii) if we are interested in only a fraction of the values of the first set to be included in the second value set, then missing shared values due to different formatting amounts to discarding the IND candidate if the overlap between value sets drops below the expected minimal fraction. In both cases, discarding an IND candidate could lead to missing a valid join opportunity between the two members of the dependency.

In this chapter we are interested in the second case above, viz., *partial* INDs, which we define in the next section.

6.2 Background and related work

The goal of this chapter is to introduce an approach to facilitate the discovery of INDs between the attributes of datasets in a data lake. Firstly, we briefly explore the notion of inclusion dependencies and the state–of–the–art available for discovering them.

6.2.1 Inclusion dependencies

Inclusion dependencies are a fundamental type of structural metadata, originally devised in the context of relational databases [CFP82]. Intuitively, an IND states that a combination of attributes from one table is a subset of the values of another

attribute combination, often from a different table. Formally, INDs are defined as follows:

Definition 6.1. *Given S and T, two datasets with schemas $s = (s_1, \ldots, s_k)$ and $t = (t_1, \ldots, t_m)$, respectively, let $\bar{s} = s_{i_1}, \ldots s_{i_n}$ and $\bar{t} = t_{j_1}, \ldots t_{j_n}$ be two n–ary attribute combinations with attributes from s and t, respectively. An inclusion dependency, $\bar{s} \subseteq \bar{t}$, states that $\bar{s}$ is included in $\bar{t}$ if for every tuple $r_s \in S$, there is a tuple $r_t \in T$, such that $r_s[\bar{s}] = r_t[\bar{t}]$.*

In Definition 6.1, $\bar{s}$ is called the *dependant* column combination, and $\bar{t}$ is called the *referenced* column combination. In this chapter, we focus on column combinations with only one dependant and only one referenced, often represented using attribute names, e.g., $s_i \subseteq t_j$. This is because the format transformation algorithm used to normalise format inconsistencies in attribute values was devised with single values in mind. Supporting n-ary INDs is a topic for future work.

Furthermore, since it is more likely for two attributes s_i and t_j from a data lake to share some, rather than all, values, in this chapter we focus on *partial* inclusion dependencies, defined with respect to an overlap coefficient $ov(s_i, t_j) \in [0, 1]$ between the extent of the dependant s_i and the extent of the referenced t_j. The overlap coefficient is defined as follows:

$$ov(s_i, t_j) = \frac{|[\![s_i]\!] \cap [\![t_j]\!]|}{|[\![s_i]\!]|} \tag{6.1}$$

where $|[\![s]\!]|$ represents the cardinality of the extent of attribute s.

From Equation 6.1 it follows that, given an IND $s_i \subseteq t_j$, with $ov(s_i, t_j) < 1$, $\exists\ t_j \subseteq s_i$, with $ov(t_j, s_i) < 1$. In other words, two attributes with overlapping instance values but not in a full inclusion relationship can spawn two partial IND candidates, each with an overlap coefficient given by Equation 6.1.

6.2.2 State–of–the–art

In practice, INDs are important for many tasks such as data integration [MHH+01], [ZHO+10], schema design [LV00], query optimisation [Gry98], and integrity checking [CTF88]. In this section we briefly mention some of the IND discovery algorithms available in the literature, which we use as "black–boxes" on datasets

preprocessed by the technique proposed in this chapter. As such, any improvement in the effectiveness of IND discovery in this chapter originates from combining existing IND discovery algorithms with the proposed preprocessing step, and not from modifying or devising new algorithms.

Early IND discovery proposals (e.g., [KL03]), made use of SQL–statements to check the validity of IND candidates generated based on instance value overlap. Later approaches, such as SPIDER [BLN06], focused more on efficiency by using inverted indexes and sort–based merging to identify pairs of attributes with overlapping extents. Again, aiming at efficiency, FAIDA [KPD+17] and SINDY [KPN15] use hash–based approximate methods and map–reduce techniques, respectively, to speed-up the identification of IND candidates. The latter of the two discovers partial INDs as well.

Lastly, BINDER [PKQN15] construes the task of IND discovery as a divide–and–conquer problem of efficiently identifying pairs of attributes with overlapping instance values and discovers both unary and n–ary INDs.

Both partial and full INDs are important in identifying potential joining opportunities between datasets from a data lake. Consequently, the IND discovery algorithm we use this chapter is SINDY [KPN15], but our proposed preprocessing step enables the identification of INDs between differently formatted attributes can be used in conjunction with most IND discovery proposals.

On the task of performing IND discovery on differently formatted datasets, *Auto–Join* [ZHC17], which we covered in Section Chapter 5, aims at discovering transformation–accompanied join opportunities. Their hypothesis is that similarity–join operations can be converted to equi–join operations through the use of automatically synthesised transformations. To this end, they propose a space of transformation operators, less expressive than our transformation language proposed in Chapter 4, that is searched for the suitable combination, i.e., transformation, that would equalise the values of given join attributes. A pre–search for suitable pairs of joinable attributes performs an all–against–all q–gram–based comparison that can be costly when many input datasets are used.

6.3 Overview and contributions

In this section we provide an overview of our method which we call $T\text{–}IND$, for transformation–informed inclusion dependency discovery. As the name suggests,

the intuition is that, given a collection of datasets with heterogeneous formatting of similar values, one can use format transformations to normalise the value representation of attributes before searching for INDs. Therefore, we characterise our main contribution in this chapter as follows: *a preprocessing step to the identification of INDs that combines data–relatedness discovery and automatic format transformations to mitigate the impact of heterogeneous format representations on inclusion dependency discovery.*

Since the envisaged usage of INDs in the context of data wrangling is for creating join opportunities that can eventually contribute to merging the wrangled datasets into an unified view, we are mostly interested in INDs between join attributes.

Figure 6.1 illustrates a three–step approach to T–IND discovery, where steps **(1)** and **(2)** represent the contribution claimed in this chapter to a state–of–the–art IND discovery, e.g., an algorithm such as SINDY, in step **(3)**. Overall, the intuition is to enrich the search space of attributes, among which INDs are searched, with transformed variants of (some) of the original attributes.

With respect to each component from Figure 6.1, firstly, we assume that T–IND discovery is not performed on the entire data lake, but on a subset of datasets for which we already know that potential INDs exist, i.e., there is evidence of pairs of attributes with overlapping values. This assumption is necessary because both IND discovery and examples generation tasks incur a performance cost when the scale of the input matches the potential scale of a data lake. The cost of IND discovery is high, as most IND discovery algorithms perform a pair–wise processing of attributes given at input. Similarly, the potential cost of example generation results from the need to align similar instance values given a pair of attributes, as discussed in Chapter 5.

The subset of datasets on which to apply T–IND discovery can result from dataset discovery, as discussed in Chapter 3, i.e., viewing IND discovery as a data preparation task, is natural to firstly identify the relevant inputs from the data lake, through dataset discovery, and then apply preparation processes on those inputs alone.

With the subset of relevant datasets to profile in hand, resulted from a dataset discovery task, we rely on already created LSH indexes to efficiently identify pairs of attributes for which there is evidence of overlapping values (step **(1)** in Figure 6.1). Each such *(dependant, referenced)* pair is passed to an example generation

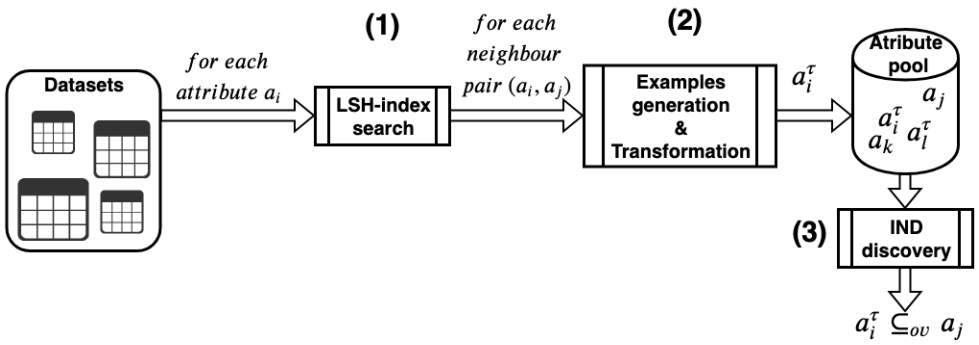


Figure 6.1: Simplified illustration of the constituent steps of a T–IND discovery process.

and transformation process (as described in Chapter 5) that may reformat the dependant attribute (step **(2)** in Figure 6.1).

Under the requirements of example generation, pairs of source and target attributes have to be identified beforehand. In such a pair, the source represents the attribute to be transformed, and the target represents the attribute that exemplifies the expected result of the transformation. Recall that, in Chapter 5, we relied on schema matching techniques for identifying such pairs. While taking the same approach here would be a viable option, in this chapter we rely on the LSH–based indexes, created as part of data discovery. Specifically, we use the index that groups attributes based on their instance values, $I_{\mathbb{V}}$). The advantages of using an LSH index are twofold: (i) as per Theorem 1 from Chapter 3, LSH–based similarity discovery is more efficient than the all–against–all string comparison often employed by schema matching algorithms such as COMA++ [DR02], and (ii) assuming dataset discovery is one of the earliest tasks in data wrangling, and performed once on the entire data lake, LSH–based indexes can be reused when T–IND discovery task is to be performed.

6.4 Transformation–informed IND discovery

In this section we discuss each step in Figure 6.1 and brought together in Algorithm 6.1. The process assumes a given collection of already discovered related datasets, call it $\mathcal{D}$, an already created value–based LSH index, call it $I_{\mathbb{V}}$, that groups attributes of datasets in $\mathcal{D}$ that share instance values, and a state–of–the–art IND discovery algorithm that takes as input a collection of attributes (together

with their sets of distinct instance values) and returns a collection of INDs. Then, Algorithm 6.1 can be applied to discover the collection of transformation–informed inclusion dependencies between attributes of datasets from $\mathcal{D}$.

Intuitively, T–IND discovery considers each attribute a_i of each datasets from $\mathcal{D}$ (line 3). For each such attribute, an index lookup is performed (line 4) to identify the neighbouring attributes of a_i for which, as per the design principles of $I_{\mathbb{V}}$, there is evidence that their extents share values with the extent of a_i. For each such neighbouring attribute na_i that is also part of some dataset from $\mathcal{D}$ (line 5), a set of example pairs is generated using the technique introduced in Section 5.5 of Chapter 5 (line 6). The resulting example instances are used to synthesise a transformation program τ using *SynthEdit* (line 7), which is then used to transform each instance value of a_i (line 8). The result is a new attribute a_i^τ that is a normalised version of a_i, and that is added to an attribute pool (line 9), together with the neighbour na_i that acted as the target in the synthesis process. Finally, the resulting attribute pool, which now contains attributes with mostly normalised extents, is searched for partial INDs using an off–the–shelf IND discovery algorithm (line 13), e.g., SINDY.

With respect to Algorithm 6.1 we observe the following:

- The LSH lookup process in line 4 considers only attributes with non–numerical values, since $I_{\mathbb{V}}$ does not contain numerical attributes (as discussed in Section 3.4 of Chapter 3). In order to include numerical attributes as well, a matching task would have to be performed before data profiling, and the results of the former used in preprocessing the input for the latter. In this chapter, we only consider non–numerical attributes.

- The transformation step in line 8 modifies each instance value of a_i whose format representations are covered by the previously generated examples. Otherwise, the values are left unchanged.

- T–IND discovery relies on the off–the–shelf IND discovery algorithm used at line 13, the compute the overlap coefficient between the attributes contained in the pool, and to compare the coefficient against a pre–configured threshold.

- If a_i is not amenable to normalisation, then a_i^τ will be equivalent to a_i. It follows that the INDs that do not require a format transformation and that

Algorithm 6.1 T–IND discovery

Input: All attributes of all datasets from $\mathcal{D}$, $I_{\mathbb{V}}$
Output: A collection of T–INDs

```
1: function TIND_discovery
2:     tinds ← {}; attribute_pool ← {}
3:     for all a_i ∈ D do
4:         for all na_i ∈ I_V.lookup(a_i) do
5:             if na_i ∈ D then
6:                 eg ← examples_generation([[a_i]], [[na_i]])
7:                 τ ← SynthEdit(eg)
8:                 a_i^τ ← τ(a_i)
9:                 attribute_pool ← attribute_pool ∪ {a_i^τ, na_i}
10:            end if
11:        end for
12:    end for
13:    tinds ← IND_discovery(attribute_pool)
14:    return tinds
15: end function
```

would have been discovered without normalising the values beforehand (by applying SINDY directly on the set of attributes from $\mathcal{D}$) are subsumed by the final collection of T–INDs. This is why, in line 9, the original a_i is not added to the attribute pool. Furthermore, if only few of the dependant values require a format transformation to become equivalent to the corresponding referenced values, while others are already equivalent, the resulting T–IND would be characterised by a greater overlap coefficient than the IND discovered without pre–normalising the values.

- Assuming there are attributes in the datasets of $\mathcal{D}$ whose values are amenable to transformations and, therefore, can be normalised to some format representation of another attribute, new INDs, undiscoverable in the absence of a normalisation process, would potentially be part of the final collection of T–INDs.

The last two observations above suggest the emphasis of our design on the two desiderata mentioned in Section 6.1: to improve the overlap of some of the partial INDs discoverable without pre–format–transformation as well, and to discover new partial INDs that would not be identified without normalising the values

beforehand. We now describe the evaluation carried out to ascertain the extent to which these goals have been met.

6.5 T–IND discovery evaluation

The goal of this chapter is to enable the discovery of inclusion dependencies on input attributes that exhibit format representation inconsistencies. To this end, we propose preprocessing the input to a traditional IND discovery algorithm, as discussed in Section 6.4.

To evaluate our contributions, we use data from Web–based data lakes to compare the results of applying a state–of–the–art IND discovery algorithm (SINDY) to datasets from multiple domains, and the results of applying a T–IND discovery process, i.e., the same IND discovery algorithm is preceded by data discovery and format transformation, on the same datasets. Specifically, we compare the number of INDs discovered using SINDY on the set of original attributes from our input datasets and the number of INDs discovered using SINDY on a set of attributes preprocessed as in Algorithm 6.1. We also analyse and report on the INDs discoverable only when using T–IND discovery, the INDs discoverable when using any of the two approaches but characterised by a greater overlap coefficient when using T–IND discovery, and on the INDs that are missed by T–IND discovery but discoverable when using SINDY.

Finally, we briefly discuss the viability of the obtained INDs with respect to performing join operations and we touch on the cost of performing T–IND discovery in terms of the time required to run the pre–format–transformation steps.

6.5.1 Data repositories used in evaluation

The motivation for T–IND discovery is given by the need to identify INDs even when the input datasets exhibit format representation heterogeneity among their attributes. To asses the proposed method, we require input datasets that contain information from similar areas, and whose values are formatted differently (at least on some of the attributes). Consequently, we explored government–specific open data lakes, e.g., `data.gov.uk`, `public.enigma.com`, `opendata.cityofnewyork.us`, `data.parliament.uk`, etc., and extracted 95 datasets from 10 different domains, with attributes characterised by various levels of format heterogeneity.

Table 6.1 describes the domain, number of datasets, average cardinality and arity, and total number of attributes of the data used in the evaluation, with the last three columns referring to non–numerical attributes.

In Table 6.1, datasets from the same domain include attributes with similar information formatted differently that hinder the discovery of INDs. Table 6.2 illustrates some examples of such instance value pairs. Note that, in order for IND discovery algorithms to discover an IND between attributes with such values, each dependant value has to be transformed to its corresponding referenced value (or vice–versa). This is precisely what the preprocessing step proposed in this chapter aims to do.

Note also that Table 6.2 illustrates transformation needs specific to partial join attributes. While it may not be always the case that attributes paired by the LSH–index do represent viable join opportunities, in the experiments from this section, the vast majority of reported INDs refer to pairs of joinable attributes.

6.5.2 Experimental setup and reported measures

In performing the experiments we use a publicly–available implementation of SINDY (`https://github.com/sekruse/sindy`) for inclusion dependency discovery. For LSH–indexing we used a value–based index created using the techniques described in Section 3.4 of Chapter 3. For example, generation and transformation synthesis we used the weighted scheme described in Section 5.5 of Chapter 5 and the *SynthEdit* algorithm from Chapter 4, respectively. We used 0.7 for both the overlap threshold value in SINDY, i.e., two attributes have to have an overlap coefficient of at least 0.7 in order to be considered an IND candidate, and the similarity threshold in LSH, i.e., two attributes are grouped under the same bucket in the LSH index if their Jaccard similarity is estimated to be at least 0.7.

We run SINDY and Algorithm 6.1, separately, on each domain mentioned in Table 6.1, and we analyse and report the count of the following types of INDs:

- The INDs that are discovered by SINDY and the INDs that are discovered by Algorithm 6.1.
- The INDs that are discovered by Algorithm 6.1 alone.
- The INDs that are discovered by SINDY with no preprocessing of the input attributes.

Domain	# datasets	Avg. cardinality	Avg. arity	Total arity
Texas governors	5	37	4	15
California gov.	4	42	5	19
US presidents	12	53	5	63
NY gov.	10	56	4	32
Hospital finance	13	443	13	189
UK parliament	7	671	13	91
UK local elections	9	1,241	11	112
NY businesses	10	2,645	13	128
Ocean companies	15	6,588	11	181
Building permits	10	6,989	29	233

Table 6.1: Data sources used in evaluation

Domain	Dependant	Referenced
Texas gov.	Sam Houston	Gov. Sam Houston
California gov.	John McDougall; 1818 March 30 1866; (Age 48)	McDougall, John
US presidents	John Adams (1735-1826)	J. Adams
NY governors	Gov. Hugh Leo Carey	Hugh L. Carey
Hospital finance	726 Fourth Street, Marysville	Fourth Street
UK parliament	Grant Shapps	shappsg@parliament.uk
UK local elections	Edmonton Gill	GILL
NY businesses	6717 4TH AVENUE, BROOKLIN	4th Av.
Ocean companies	Sea Star LTD.	Sea Star
Building permits	EUGENE BROLLYN	Brollyn, Eugene

Table 6.2: Format transformation examples

- The INDs that are discovered by SINDY and Algorithm 6.1 but are characterised by a greater overlap coefficient in the latter.

Note that we do not analyse or report INDs that appear in both cases but are characterised by a smaller overlap coefficient in the result of Algorithm 6.1 because we have not encountered such INDs. For this to happen it would require the transformation to modify some source instance values that were previously equivalent in meaning and format representation to some target instance values.

6.5.3 Evaluation results

Experiment 6.1. ***How many INDs and how many T–INDs are discovered?***

Domain	INDs	T–INDs	New INDs	Improved INDs	Lost INDs
Texas gov.	4	4	1	0	1
California gov.	6	6	0	1	0
US presidents	108	154	46	8	0
NY governors	38	46	22	2	14
Hospital finance	1,182	1,228	46	113	0
UK parliament	205	189	12	6	28
UK local elections	38	30	2	1	10
NY businesses	23	24	4	2	3
Ocean companies	308	388	123	20	43
Building permits	235	330	116	21	21

Table 6.3: Results of Experiment 6.1

The purpose of this experiment is to show the benefit, if any, of applying Algorithm 6.1 over the standalone use of SINDY, i.e., the benefit of performing LSH index–based candidate selection and format–transformation before searching for INDs.

Table 6.3 reports the number of INDs discovered by SINDY (the *INDs* column), the number of INDs discovered when applying T–IND discovery (the *T–INDs* column), the number of INDs resulting only when using T–IND discovery, i.e., missed by SINDY, (the *New INDs* column), the number of INDs with a greater overlap coefficient that result from T–IND discovery (the *Improved INDs* column), and the number of INDs discovered only when using SINDY, i.e., missed by T–IND discovery, (the *Lost INDs* column).

In most of the cases, a close analysis reveals that the generated INDs represent viable join opportunities, e.g., both the dependent and the referenced denote the same entities and uniquely identify the records they appear on: governor names, business names, addresses, etc. For instance, some datasets from *California governors* domain identify their records best on a column, *Governor*, that contains the governor name, his date of birth, and his age, e.g., *Gray Davis; December 26 1942 (age 73)*, while in other datasets the *Governor* attribute simply contains the person's name, e.g., *Davis, Gray*. A transformation that extracts the governor name from the first example can enable the discovery of an IND between the transformed first attribute and the second attribute. Similarly, some datasets from the *UK parliament* domain, identify members of parliament by their names, e.g., *Gavin Strang*, while other datasets include an attribute with the email address, in addition to the name attribute, e.g., *strangg@parliament.uk*. Consequently, in

addition to an IND between name attributes, a transformation that derives the email address from the name can enable the discovery of an IND between email address attributes.

Of particular interest is the *Hospital finance* domain where there are more than $1,000$ INDs discovered. Many such INDs (more than 50%) represent dependencies between attributes denoting calendar dates, e.g., ("10 $March$ 2008", "The 10 of $March$, 2008"). While the transformation synthesised for such cases seems to correctly equalise the value representation between the dependant and the referenced attributes, such a pair is, arguably, less viable to be used as a join path than the pair of attributes denoting hospital addresses or hospital administrator names of the same domain.

T–IND discovery proves able to identify INDs that are otherwise missed by SINDY, sometimes identifying as much as 50% more INDs, e.g., *Building permits* and *NY governors* domains. This increase is given by the existence of attributes with similar meanings but different representations between the datasets of the respective domains. For example, in some cases New York governor names are formatted differently, e.g., *George Clinton (1777 - 1795)* and *Clinton, George*. Similarly, some building addresses in the *Building permits* domain are denoted using a single attribute, e.g., *733 Front Street*, while others are expressed using multiple attributes, e.g., the house number *733*, and the street name *Front St*. A transformation that extracts different address components from an address string can inform, in this case, two INDs that can only be discovered through T–IND discovery.

Conversely, Algorithm 6.1 can result in the loss of as much as 36% of the INDs discovered by SINDY, e.g., *NY governors* domain. This happens because, while SINDY computes the exact overlap for each combination of two attributes given at input, T–IND discovery relies on LSH partitioning and considers only the attribute pairs whose members are part of the same bucket. Furthermore, LSH (with *MinHash*) is known ([SL15], [ZHC17]) to be less effective when the size distribution of the input sets is skewed or when the sizes are very small. A close verification of the INDs missed by Algorithm 6.1 showed that the component attributes contained only a few distinct values, e.g., the attributes denoting a political party in *UK elections*, *NY governors*, or *Texas governors* datasets contain between 2 and 4 distinct values, e.g., *Republican*, *Democrat*, *Conservative*, *Labour*. Similarly, the attribute denoting the city name in *NY businesses* contains only

one value: *New York*. A possible approach to mitigate such behaviour is to rely on LSH indexing for identifying overlapping but distinctly formatted attributes, and on all–against–all overlap computation for attributes with overlapping and equivalently represented values.

Finally, in some cases, Algorithm 6.1 discovers stronger evidence of overlap between attribute values, e.g., up to 10% of the INDs discovered by SINDY have been identified by Algorithm 6.1 as having a greater overlap coefficient, e.g., *Hospital finances* domain. This happens because, through format transformation, more values of the dependant attribute become equal to values of the referenced attribute. A close analysis of these cases shows that the average overlap improvement per domain is often less than 10%. Most of the time, the dependant values that become equal to some referenced value through transformation, contain extra characters/punctuation signs that are not present in the latter value. For example, a dependant attribute denoting the type of organisation a hospital represents in the *Hospital finances* domain contains values such as *Non Profit Corp* . The corresponding referenced attribute contains the same type of values but few of them include a *dot* at the end, e.g, *Non Profit Corp.* . Some of these cases involving extra/missing characters are solved through transformations and, therefore, the overlap coefficient of the resulting IND increases.

Experiment 6.2. ***What is the cost of performing the preprocessing steps on the input attributes ?***

The purpose of this experiment is to evaluate the cost incurred by SINDY and T–IND discovery in terms of the time required to discover the INDs.

Table 6.4 reports the time, in seconds, spent by SINDY and by Algorithm 6.1 searching for INDs, and the percentage of the time, relative to the latter, that corresponds to generating examples and transforming the values. We focus on example generation and transformation since this is the most costly task when the size and number of input attributes increases.

When there are few input attributes and few INDs to be discovered (e.g., less than 40), both SINDY and T–IND discovery take less than 20 seconds to run (e.g., *NY businesses* and *UK elections* domains). In contrast, more input attributes combined with a high cardinality lead to a dramatic increase in the time required by T–IND discovery to run, while the time required by SINDY remains at no more than 5 seconds, e.g., *Building permits* domain. Whenever T–IND discovery takes more than a few seconds to run, e.g., domains such as *US presidents* or

Domain	**SINDY(s)**	**Alg. 6.1(s)**	**Transformation time(%)**
Texas gov.	1.46	3.24	10.01%
California gov.	1.82	2.91	6.2%
US presidents	1.35	121.28	98.36%
NY governors	1.93	298.39	98.93%
Hospital finance	2.38	172.34	97.15%
UK parliament	2.07	601.45	99.38%
UK local elections	2.65	6.48	34.22%
NY businesses	3.02	10.54	45.27%
Ocean companies	3.76	1,187.84	99.4%
Building permits	5.41	4,035.96	99.75%

Table 6.4: Results of Experiment 6.2

Ocean companies, more than 98% of the running time is spent generating examples and transforming values, while the rest is split between LSH–index lookup and running SINDY.

A close analysis of the above behaviour shows that, as expected, given the evaluation results from Chapter 5, examples generation becomes expensive when the input attributes contain many values with a high degree of source format inconsistencies, i.e., when there are many formats used to represent the source values. For example, domains such as *Ocean companies* or *Building permits*, each with an average cardinality of nearly $7,000$ records, are characterised by many source formats. The consequence is that, in such cases, examples generation can require many iterations, each one performing program synthesis and validation against some of the values (see Section 5.5 of Chapter 5 for details). For instance, in the case of *Building permits*, T–IND discovery took more than one hour to discover 50% more INDs than SINDY, with examples generation involving, on average, more than 100 iterations per pair of attributes with overlapping instance values and more than 500 such pairs. Recall, from Chapter 5, that each iteration uses a reduced set of example instances to synthesise a transformation program and to transform all dependant values. At each subsequent iteration, the previous set of examples is incremented and the synthesis/transformation tasks are performed again. The loop continues until there is evidence of successful transformation of dependant values or of the impossibility to synthesise a transformation.

In summary, the experiments of this section show that selecting attribute pairs for IND discovery and pre–normalising those attributes can be effective, with as much as 50% more INDs discovered than the simple use of an IND discovery

algorithm, many of which represent viable join opportunities. The flip side is that LSH–based selection and pre–normalisation of attributes come at an efficiency cost that can be too high if there are many values to transform with a high degree of format heterogeneity.

This drawback can, potentially, be alleviated through horizontal scaling of the example generation process. For instance, the processes of generating examples and transformation, given a pair of attributes, is independent from the generation of examples and transformation of another pair of attributes. This means that the inner loop of Algorithm 6.1 can be easily parallelized.

6.6 Summary and conclusions

This chapter has proposed an approach in which techniques specific to data relatedness discovery are combined with automatic format transformation of instance values to enable the discovery of partial inclusion dependencies, even when the values from the input datasets exhibit formatting inconsistencies. We materialised this technique in a three–step process (Figure 6.1), where the first two steps constitute the claimed contributions: LSH–based attribute selection and automatic format transformation, respectively, while the third step is the application of an existing state–of–the–art IND discovery algorithm. The resulting technique, formalised in Algorithm 6.1, exhibits the following characteristics:

- It considers the input as a collection of attributes that is updated in the first two steps with normalised versions of dependant attributes, therefore making possible the discovery of INDs that would otherwise be missed due to format inconsistencies. This is in line with the first desideratum discussed in Section 6.1.

- Through the format normalisation process, the number of shared values (and the overlap coefficient) between a dependant and a referenced attribute can increase, thereby, addressing the second desideratum discussed in Section 6.1.

The impact of the above characteristics has been assessed using open government data from multiple domains. The results showed that, compared with a standalone use of an IND discovery algorithm, our technique is promising in identifying more INDs, some of which are characterised by higher degrees of overlap

between dependant and referenced attributes. In contrast, our proposal in this chapter can lead to missing INDs, otherwise discoverable using state–of–the–art algorithms. Often, such INDs are between attributes with few distinct values and skewed cardinalities that suffer from known weaknesses in LSH and MinHash.

For attributes with many instance values and many format representations, effectiveness comes at a performance cost represented by the time required to normalise the values before searching for INDs.

Overall, in this chapter, we showed that data relatedness discovery and automatic format transformation are two data preparation tasks that can can be effectively combined with the goal of improving the results of other individual wrangling steps, such as data profiling. Since the proposed preprocessing step relies on transformation synthesis algorithms, the improvements in IND discovery shown in this chapter are possible if the conditions for performing format transformation synthesis, discussed in Chapter 4, are met. When this is the case, the proposal of this chapter leads to the fulfilment of objective O_4 from Chapter 1.

Chapter 7

Conclusions and future work

Data preparation has been called "janitorial" work many times (e.g., [NYt], [Datc]), mainly due to its repetitive, often mundane, handcrafted tasks that require active user involvement for long periods. At the same time, it has been widely acknowledged that preparing the data for analysis is an essential requirement for extracting value from that data. This thesis set out to investigate what makes data preparation challenging and costly, and how data preparation can become more cost–effective. In this chapter, we recall the stages of this investigation and reflect on the significance of the contributions made. We also look at future work that might advance the area of data wrangling and reflect on whether or not data wrangling is worthwhile.

7.1 Thesis overview

We started with the conviction that data preparation is important, encouraged by the large number of reports and research papers that stress its significance and cite the famous claim that "data scientists spend up to 80% of their time preparing the data for analysis and only 20% of their time analysing it", e.g., [Datc, Data, Datb], [KHP+11, DFA+17, KKA+17]. Then, we chose to focus on a scenario in which data analysis is to be performed on data flowing out of a, potentially large, repository with datasets in various formats, from multiple domains, i.e. a *data lake*. This type of environment and existing work on the subject encouraged us to consider *data wrangling* as the embodiment of data preparation in data lakes. Next, we set out to identify what makes data wrangling costly in terms of time, effort, and user technical knowledge required to perform it

in data lakes. We pinned down two challenges that were insufficiently addressed in the state–of–the–art:

- **The scalability challenge**: Wrangling data becomes more costly when applied on an ever increasing data repository due to the need to run, often different, wrangling processes on many datasets.
- **The heterogeneity challenge**: Wrangling data becomes more costly when applied on data whose textual values are inconsistently formatted due to the labour intensive, often manual, tasks involved in normalising the values.

A review of current trends in data wrangling revealed the opportunities for addressing the above challenges, thereby making data wrangling more cost–effective. We identified the potential for automating two of the many data wrangling sub–processes: the data discovery component, addressing the scalability challenge, and the format transformation component, addressing the heterogeneity challenge. Specifically, we argued that, by automating the discovery of datasets related to a given target dataset, we can limit the application of, potentially costly, wrangling tasks to the relevant input. Similarly, by automating the process of format normalisation, we relieve the user from having to manually identify, potentially many, transformation needs, and from manually writing transformation rules for each case.

Having pinned down the main opportunities that motivate us, we defined the research aim of the thesis:

> *This thesis aims to reduce the cost of data preparation by developing techniques that automate the identification of data relevant for a given wrangling task and the operations necessary to clean and transform the format representation of data values.*

Given this aim, we described in detail techniques to perform data discovery and format transformation automatically. In the next section (Section 7.2), we briefly describe these techniques again, whilst revisiting the main contributions of this thesis. We both discuss the significance of these contributions with respect to the gaps in the state–of–the–art they aim to fill, and assess our success in filling in those gaps. Finally, we discuss some unanswered questions in the area of data wrangling and present a few ideas aimed at advancing the work of this thesis in Section 7.4.

7.2 Main contributions and their significance

The major contributions were introduced in Chapter 1, in relation to the main objectives that steered the research in this thesis. We now review those objectives and contributions, together with their significance with respect to the advances over the state–of–the–art they provide.

7.2.1 Data discovery contributions

On the topic of data discovery and the scalability problem, this thesis sought to identify the types of evidence that enable the identification in a data lake of those datasets that are most relevant for the given task, and to develop techniques that use the evidence to perform cost–effective dataset discovery, i.e., O_1 from Chapter 1.

Representative data wrangling systems such as Wrangler [KPHH11] or Data Tamer [SBI+13], expect their input to be provided by the user and, therefore, they do not address the scalability challenge directly. In other relevant works, e.g., [DFA+17, NZPM18, FAK+18], the problem of data discovery is tackled directly by means similar to our proposal from Chapter 3. However, important characteristics, often exhibited in real–world data lakes, have been identified as not being addressed by these proposals:

- Data lakes often contain data from various sources with different representations of similar concepts. Therefore, the discovery process ought to be lenient with respect to the format representation of textual values. In Chapter 3, we addressed this desideratum by contributing a method for finding related attributes using both schema and instance–level fine–grained features, i.e., based on tokens of instance values rather than full instance values. The evaluation section of Chapter 3 showed that the use of such features is more effective in identifying relatedness than relying on full values, especially in cases when similar entities are inconsistently represented.

- Data stored in data lakes does not obey pre–determined rules/schemas and it often lacks descriptive metadata. Therefore, relatedness evidence ought to be extracted from the data values themselves and based on multiple types of evidence that can be considered together in quantifying the level of relatedness between two datasets. In Chapter 3, we addressed this desideratum by

contributing a framework for measuring relatedness that maps five types of similarity evidence to a uniform distance space, thereby enabling an aggregated view on the notion of relatedness between two datasets, to which each type of evidence contributes. The evaluation section of Chapter 3 showed that such combination of similarity signals identifies the closely related datasets to a given target more effectively than each type of the evidence independently can do.

- Given a target dataset, the discovery of relevant datasets with which to populate it requires example instance values that can prove insufficient to relate all potentially useful candidates from a data lake. Therefore, other means of relating datasets, that can overcome a weak relatedness signal from the used evidence, ought to be considered. In Chapter 3, we addressed this desideratum by contributing a method for relating datasets based on postulated join paths between closely related datasets and less related datasets to a given target. The evaluation section showed that join paths allow for the identification of additional datasets, weakly related to the target, but, nevertheless, with attributes relevant for increasing the target coverage.

The extensive experiments of Chapter 3 compared our proposal for data discovery with state–of–the–art alternatives represented by *TUS* [NZPM18] and *Aurum* [FAK+18]. The results showed that D^3L allows for more effective dataset discovery due to its distance–based framework, which enables each type of evidence used to contribute to measuring the relatedness between two datasets, and to its use of schema and instance–level fine–grained features, which are more lenient in identifying relatedness relationships, especially when similar entities are inconsistently represented. Furthermore, our proposal proved capable of more precise covering of the target dataset due to its use of join paths based on LSH evidence and subject–attributes.

Treating the datasets discovery as an LSH index lookup task and leveraging LSH properties to efficiently compute relatedness measurements, showed that D^3L is also more efficient than *TUS* in both, index construction and discovery tasks, and comparatively efficient with *Aurum*.

Overall, the contributions envisaged in Chapter 1 as necessary for automatic dataset discovery in data lakes have been provided, as reviewed above. Thus, we consider objective O_1 of this thesis attained.

7.2.2 Format transformation contributions

On the topic of format transformation and the heterogeneity challenge, this thesis set out to relieve the user from having to manually identify the need for transformation and to create transformation scripts designed to normalise the textual representation of similar entities. This has been approached by developing algorithms that automatically identify similar entities with different textual representations and use them to learn a sequence of string operations that edit an input string into its corresponding output, thereby addressing O_2 and O_3 from Chapter 1.

Representative data wrangling systems either do not tackle the problem of unnormalised textual representation of values (e.g., Tamer [SBI+13]), or rely on the user to guide and decide the normalisation process (e.g., Data Civilizer [DFA+17]). Other approaches assist the user in developing transformation scripts, but still assume advanced technical knowledge (e.g., Wrangler [KPHH11]).

Fully automated approaches to format transformation have been proposed in the area of spreadsheet transformation (e.g., [Gul11], [Sin16]). Although these proposals fulfil many of the desiderata envisaged in Chapter 4, they lack two important properties that become essential when applying such algorithms on data in a data lake:

- Fast convergence: datasets of large sizes may exhibit multiple transformation requirements, each of which has to be covered by at least one input–output example. This, in turn, may result in a large number of examples that generate a time–cost problem for state–of–the–art synthesis algorithms. In Chapter 4, we address the requirement for fast convergence, while preserving the level of effectiveness achieved by the closest competitor, with (i) an expression language that is expressive enough to represent a variety of string manipulation operations, while being restrictive enough to allow efficient synthesis given a set of input–output examples; and (ii) a synthesis algorithm that, given a collection of input–output string pairs, efficiently converges to the simplest transformation consistent with the given examples.

- Automatic provision of input–output examples: relying on the user to provide more than a few valid and representative input–output examples required to normalise a dataset is unrealistic, especially when there are multiple such datasets to process and the datasets themselves could be large. In Chapter

5, we address this requirement by devising two reliable techniques for the automatic provision of input–output examples for synthesis algorithms that relieve the user from having to manually input such specifications.

The evaluation section of Chapter 4 showed that our proposed synthesis algorithm, *SynthEdit*, performed substantially more efficiently than the closest state–of–the–art alternative, *FlashFill* [Gul11], while achieving better recall and comparable precision. The performance improvement stems mostly from the underlying reliance of *SynthEdit* on automata structures that allow for an efficient coverage of the, potentially large, space of possible transformations consistent with the given examples.

On the topic of employing *SynthEdit* in an automatic setting to normalise datasets from a data lake, the evaluation section of Chapter 5 showed that input specifications, in the form of input–output examples, can be reliably and automatically generated by using either postulated functional dependencies or similar matching of instance values. Furthermore, a potentially high cost of synthesis was shown to be avoidable by automatically pruning redundant examples instances.

Overall, the contributions envisaged in Chapter 1 as necessary for automatic format normalisation in data lakes have been provided, as reviewed above. Thus, we consider objectives O_2 and O_3 of this thesis attained.

7.2.3 Data profiling contributions

Normalised instance values can benefit not only the data analysis process by providing a clean end–product of data wrangling, but other individual wrangling tasks as well. For example, given that data profiling algorithms and, specifically, inclusion dependency (IND) discovery algorithms (e.g., BINDER [PKQN15], FAIDA [KPD+17], SINDY [KPN15]) assume normalised formatting in their input attributes, datasets from a data lake with potentially inconsistent value representations are not a suitable use case for such algorithms. In Chapter 6, we set out to attain objective O_4 from Chapter 1, and address the above mentioned problem by proposing a preprocessing of the input to profiling algorithms. Specifically, we employ data discovery–specific index structures to efficiently identify pairs of attributes with overlapping values, and format transformations to automatically normalise the values of such candidates, so that specialised profiling algorithms can find INDs.

Experiments conducted on real–world datasets showed that, compared with a standalone use of an IND discovery algorithm, our technique is promising for identifying more INDs, some of which are also characterised by higher degrees of overlap between dependant and referenced attributes. For attributes with many instance values and many format representations, the effectiveness comes at the expense of efficiency due to the extra step of normalising the values before searching for INDs.

Overall, in Chapter 6 we show that data preselection and format normalisation can be beneficial for other wrangling tasks, such as data profiling and, thus, we consider O_4 of this thesis attained.

7.3 Impact of contributions

The previous section argued that the empirical evaluations performed in Chapters 3–6 shows that this thesis's contributions fulfil the objectives set in Chapter 1, and successfully cover the limitations identified in the state–of–the–art. In this section, we observe that the reported contributions are being transferred to a commercial product, viz., Data Preparer (`https://thedatavaluefactory.com/`).

Data Preparer is an automated data preparation tool and a materialisation of the vision of cost–effective data wrangling proposed in the VADA project [KKA+17, KAB+19]. The tool brings together several state–of–the–art techniques focused on specific wrangling tasks, e.g., data discovery, data profiling, schema matching and mapping, format transformations, etc. The implementations of D^3L from Chapter 3, and *SynthEdit* from Chapters 4, together with example generation techniques from Chapter 5, are the bases upon which the data discovery and format transformations components of Data Preparer have been built.

The incorporation of the ideas of this thesis into a commercial solution that is driven by wrangling cost reduction, together with the fulfilment of specific desiderata left unaddressed by current state–of–the–art, demonstrate the potential for practical impact of our contributions.

7.4 Unanswered questions and future work

Research on data wrangling is still in its early years, especially in the context of data lakes. In this thesis, we have contributed to the automation of only two of the

tasks involved. This suggests that we are far from exhausting the space of needs in data wrangling. In fact, there are many research questions left unanswered even on the wrangling tasks of interest in this thesis. In this section, we visit some of these questions and briefly propose some ideas on how to tackle the problems they raise that may act as starting points for future work on data discovery, on format transformations, and on data wrangling as a whole.

Can the evidence for discovery be extended with domain knowledge?

One approach to improving data discovery is to incorporate domain knowledge into the relatedness evidence used in drawing the relationships between datasets. Work on this subject has been conducted in [LSC10, VHM+11, PAKS16]. While the ideas stemming from this body of work are relevant for our data discovery scenario in this thesis, they involve the use of external knowledge–bases and aim for a general degree of annotation that often performs poorly, as seen in Chapter 3 where TUS [NZPM18], which considers YAGO as a source of similarity evidence, proved less precise and less efficient than D^3L. One alternative would be to narrow down the scope of the annotations to domain–specific knowledge–bases that would constitute a sixth type of evidence in determining the relatedness between datasets.

Can the relatedness of numerical attributes be grounded on an LSH indexing scheme?

In Chapter 3, the relatedness of numerical attributes is treated as a special case that is not construed as common LSH bucket membership. This is because hashing schemes with LSH properties that can quantify the resemblance of numerical distributions are still on–going research topics. A few proposals have shown promising results under certain conditions, e.g., [GD04] for LSH–based earth mover's distance estimation, and [MFH+17] for LSH–based Jensen–Shannon divergence estimation.

From a practical point of view, an implementation and integration of such LSH schemes into D^3L would represent an interesting next step. Given the restrictions the current distribution similarity estimation proposals convey, e.g., samples of equal sizes, their applicability could be limited. Therefore, from a research point of view, further investigation on fast estimation of numerical distribution similarity measures that avoids all–against–all comparisons would be welcomed.

Can other data wrangling tasks benefit from the discovery methodology?

The focus of Chapter 3 has been data discovery. To that extent, each dataset of the data lake has been vertically partitioned and construed as a collection of attributes (with associated instance values). However, the nearest neighbour search techniques employed in performing attribute–level relatedness discovery construe their input, i.e., the attributes, as merely sets of items. This opens the opportunity for other wrangling tasks, whose inputs can be processed as sets of items, to be suitable use cases for the same methodology. One such example is the entity resolution problem [EIV07] (i.e., duplicate record detection), where, from individual records of a dataset, a set can be derived and indexed under specific LSH structures. Although this has been tried before (e.g., [Chr12]) an additional challenge is to adapt the multi–evidence aggregation scheme used for dataset discovery to the problem of duplicate detection.

Can transformations be applied beyond inconsistency of formats?

On the topic of transformations, in this thesis we have considered the normalisation of textual representation on strings. But the heterogeneity of data consists of more than just formatting, e.g., schema level heterogeneity, lexical heterogeneity, etc.

Recent work on program synthesis has addressed the problem of spreadsheet table transformations (e.g., [HG11]) that allows for automatic structural changes to a table. The proposed algorithm is devised, again, with spreadsheets in mind and suffers from the same efficiency issues as $FlashFill$. There is an opportunity here to apply the methodology used with $SynthEdit$ for performing table transformations, e.g., *split column*, *concatenate columns*, etc., efficiently. The initial challenges would be to devise a language expressive enough to describe table transformations and simple enough to be amenable to synthesis, to determine what constitutes an input–output example, and how to map different such examples to automata states and transitions (recall from Chapter 4 that automata structures are at the core of $SynthEdit$ methodology).

Can data wrangling be applied on semi/un–structured data?

Industry reports (e.g., [Datc]) suggest that 30% of data analysis tasks involve semi–structured and unstructured data, e.g., log files, tweets, user reviews, etc. Often, data lakes can contain a variety of data types ranging from structured and tabular data to unstructured datasets. Since, most of the time, analytical tools expect tabular inputs, wrangling semi/un–structured data from a data lake would also involve a new task of extracting and mapping the most relevant pieces of

information from such a dataset to a tabular format. Moreover, these types of data come with similar challenges to the ones considered in this thesis. For example, relatedness relationships between unstructured data sets need to be discovered as well. The challenge here is that unstructured data does not come with an intrinsic schema that can be leveraged to derive useful similarity signals. Other techniques would have to be identified/devised in order to extract useful evidence from such data, e.g., natural language processing–based entity recognition, e.g., [RCME11].

Can the current proposals be adapted for user–guided data wrangling?

In this thesis, automation has been the basis for cost–reduction in data discovery and format normalisation. However, automatic solutions are unlikely to be able to match the reach or quality of cleaning produced by data scientists. One potential direction for future research would be the incorporation of user feedback in the process of dataset discovery and format transformation. The aim in a user–guided wrangling process would be to help a human interpret the data in the lake, not to serve data to an automated wrangling pipeline. User supervision could come, for example, in the form of corrections made to the end–product that triggers a re–run of specific wrangling tasks, and does not involve advanced technical knowledge. This is, essentially, the vision of cost–effective data wrangling envisioned by the VADA system [KKA+17, KAB+19], and adopted in this thesis. Therefore, incorporating user feedback in the solutions presented in this thesis represent a promising direction for future work.

7.5 Final reflections

The concomitant achievements from the last decades in data management and machine learning have transformed the task of data analysis into an enabler of reliable data–driven business decisions. Initially, the advancement in data management focused mostly on *silo-ed* data models, specific to data warehousing. As data become more and more available, two needs surfaced: the need for a more liberal data management approach, and the possibility of performing data analysis on–demand, often by non–experts. The first need was addressed by data lakes, as discussed in Chapter 2. The second need proved more challenging, as the process of analysis implies clean data that is often hard to acquire. This is how data wrangling emerged as the beacon for self–service data preparation and this is where our contributions from this thesis can make a significant practical

impact. Specifically, data discovery offers the possibility of focusing the wrangling efforts on relevant data for the task at hand, while format transformation opens the possibility for performing basic operations required to clean the data to users without a programming background.

The research in data wrangling is still at the beginning, with numerous challenges still to be addressed. For example, automating other wrangling steps, e.g., entity resolution, schema matching, etc. But the argument has not been closed on data discovery and transformation either. As mentioned above, we consider that future work on unaddressed needs in these two areas (e.g., discovery on non–tabular data, table transformations, etc.) can benefit from the methodology described in the chapters of this thesis.

We conclude this thesis on cost–effective data wrangling by addressing on final question regarding the importance of this area and, consequently, of the work presented in the previous chapters: *Is data wrangling worth the effort?*. This question arises from the fact that most of the research on the topic envisions it as being exploratory in nature: proper data wrangling can sometimes involve a daunting amount of work and time with little to show for in the end. We have seen in Chapter 2 that, through the adoption of the schema–on–read model, there is a cost shift from structuring the data once for data warehouse loading to preparing it every time a new analysis task is conducted. We answer the above question by reiterating an important argument of Chapter 2: there is no better or best with performing data preparation before warehouse loading or after data lake extraction. Just like most things in data analysis, it depends on the use–case. For instance, if the type of analysis is known in advance and results need to be fast and repetitive then schema–on–write might be the answer. Alternatively, if there are a lot of unknowns with the data and constant new sources, schema–on–read is more appropriate.

Finally, we end this thesis with one last argument to support the importance of the work described here: data has often been called "the new oil" [Oil] due to its immeasurable potential for business value. In this context, data wrangling, together with data analysis, represent the foundation that allows organisations to become truly data–driven enterprises.

Bibliography

[AAK10] Parag Agrawal, Arvind Arasu, and Raghav Kaushik. On indexing error-tolerant set containment. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 927–938, 2010.

[AAO+15] Ziawasch Abedjan, Cuneyt Gurcan Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. Temporal rules discovery for web data cleaning. *PVLDB*, 9(4):336–347, 2015.

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, 2007.

[ABLM14] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[ACD+16] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *Proc. VLDB Endow.*, 9(12):993–1004, August 2016.

[ADKC18] Ö Akgün, A. Dearle, G. N. C. Kirby, and P. Christen. Using metric space indexing for complete and efficient record linkage. In *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III*, pages 89–101, 2018.

[ADMR05] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard

Rahm. Schema and ontology matching with COMA++. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 906–908, 2005.

[AGM03] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 337–348, 2003.

[AGN15] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015.

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, January 2008.

[AKP+18] Edward Abel, John A. Keane, Norman W. Paton, Alvaro A. A. Fernandes, Martin Koehler, Nikolaos Konstantinou, Julio César Cortés Ríos, Nurzety A. Azuan, and Suzanne M. Embury. User driven multi-criteria source selection. *Inf. Sci.*, 430:179–199, 2018.

[AMI+16] Ziawasch Abedjan, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. Dataxformer: A robust transformation discovery system. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 1134–1145, 2016.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035, 2007.

[AW18] Hassan H. Alrehamy and Coral Walker. Semlinker: automating big data integration for casual users. *J. Big Data*, 5:14, 2018.

[AZ12] Charu C. Aggarwal and ChengXiang Zhai. *A Survey of Text Clustering Algorithms*, pages 77–128. Springer US, 2012.

[BBC+15] Anant P. Bhardwaj, Souvik Bhattacherjee, Amit Chavan, Amol Deshpande, Aaron J. Elmore, Samuel Madden, and Aditya G. Parameswaran. Datahub: Collaborative data science & dataset version management at scale. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[BBN+05] Alexander Bilke, Jens Bleiholder, Felix Naumann, Christoph Böhm, Karsten Draba, and Melanie Weis. Automatic data fusion with hummer. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1251–1254, 2005.

[BCFE03] Rohan Baxter, Peter Christen, and Centre For Epidemiology. A comparison of fast blocking methods for record linkage. *Proc. of ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 07 2003.

[BCFM98] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 327–336, 1998.

[BCG05] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: Self-tuning indexes for similarity search. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 651–660, 2005.

[BFPK19] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. Synthedit: Format transformations by example using edit operations. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 714–717, 2019.

[BG05] Jose Barateiro and Helena Galhardas. A survey of data quality tools. *Datenbank-Spektrum*, 14:15–21, 01 2005.

[BGHZ15] Daniel W. Barowy, Sumit Gulwani, Ted Hart, and Benjamin G. Zorn. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 218–228, 2015.

[BGMZ97] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Selected Papers from the Sixth International Conference on World Wide Web*, pages 1157–1166, 1997.

[BLMT16] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Trans. Knowl. Data Eng.*, 28(5):1217–1230, 2016.

[BLN06] Jana Bauckmann, Ulf Leser, and Felix Naumann. Efficiently computing inclusion dependencies for schema discovery. In *Proceedings of the 22nd International Conference on Data Engineering Workshops, ICDE 2006, 3-7 April 2006, Atlanta, GA, USA*, page 2, 2006.

[BN05] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 69–80, 2005.

[BN08] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1:1–1:41, 2008.

[BPE+13] Khalid Belhajjame, Norman W. Paton, Suzanne M. Embury, Alvaro A. A. Fernandes, and Cornelia Hedeler. Incrementally improving dataspaces based on user feedback. *Inf. Syst.*, 38(5):656–687, 2013.

[BPFK18] A. Bogatu, N. W. Paton, A. A A Fernandes, and M. Koehler. Towards automatic data format transformations: Data wrangling at scale. *The Computer Journal*, 12 2018.

[Bro93] Ruven E. Brooks. Watch what I do: Programming by demonstration. edited by allen cypher (book review). *International Journal of Man-Machine Studies*, 39(6):1051–1057, 1993.

[Bro97] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, 1997.

[BSG18] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The vadalog system: Datalog-based reasoning for knowledge graphs. *PVLDB*, 11(9):975–987, 2018.

[BT03] Yoah Bar-David and Gadi Taubenfeld. Automatic discovery of mutual exclusion algorithms. In *Distributed Computing, 17th International Conference, DISC 2003, Sorrento, Italy, October 1-3, 2003, Proceedings*, pages 136–150, 2003.

[CFP82] Marco A. Casanova, Ronald Fagin, and Christos H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. In *Proceedings of the ACM Symposium on Principles of Database Systems, March 29-31, 1982, Los Angeles, California, USA*, pages 171–176, 1982.

[CG18] Pravin Chandra and Manoj K. Gupta. Comprehensive survey on data warehousing research. *International Journal of Information Technology*, 10(2):217–224, Jun 2018.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, 2002.

[CHK09] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009.

[Chr12] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9):1537–1555, 2012.

[CHW+08] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[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.

[CMI+15] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1247–1261, 2015.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 109–118, 2001.

[Cod71] E. F. Codd. Further normalization of the data base relational model. *IBM Research Report, San Jose, California*, RJ909, 1971.

[Con99a] W. J. Conover. *Practical nonparametric statistics*. Wiley, 3. ed edition, 1999.

[Con99b] W. J. Conover. *Practical nonparametric statistics*. Wiley, 1999.

[CTF88] Marco A. Casanova, Luiz Tucherman, and Antonio L. Furtado. Enforcing inclusion dependencies and referencial integrity. In *Fourteenth International Conference on Very Large Data Bases, August 29 - September 1, 1988, Los Angeles, California, USA, Proceedings.*, pages 38–49, 1988.

[Data] Data scientist report 2016. `https://visit.figure-eight.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf`. Accessed: 2019-04-01.

[Datb] Data scientist report 2017. `https://visit.figure-eight.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport.pdf`. Accessed: 2019-04-01.

[Datc] Data scientist report 2018. visit.figure-eight.com/WC-2018-Data-Scientist-Report_.html. Accessed: 2019-03-30.

[DEE+13] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. NADEEF: a commodity data cleaning system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 541–552, 2013.

[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.

[DHI12] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, 2004.

[Dix] James Dixon. Data lakes. https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/. Accessed: 2019-03-30.

[DJ03] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley, 2003.

[DLP02] Ido Dagan, Lillian Lee, and Fernando Peteira. Similarity-based methods for word sense disambiguation. 05 2002.

[DM88] B. A. Devlin and P. T. Murphy. An architecture for a business and information system. *IBM Systems Journal*, 27(1):60–80, 1988.

[DR02] Hong Hai Do and Erhard Rahm. COMA - A system for flexible

combination of schema matching approaches. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 610–621, 2002.

[DSFG+12] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 817–828, 2012.

[DSS12] Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: Selecting sources wisely for integration. *PVLDB*, 6(2):37–48, 2012.

[DWa] Data warehouse augmentation. `cdn2.hubspot.net/hubfs/443949/DWAugmentationWhitePaper11.1.16.pdf`. Accessed: 2019-04-03.

[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.

[EMH09] H. Elmeleegy, J. Madhavan, and A. Y. Halevy. Harvesting relational tables from lists on the web. *The VLDB Journal*, 2(1), 2009.

[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.

[Fan08] W. Fan. Dependencies revisited for improving data quality. In *PODS 2008, June 9-11*, pages 159–170, 2008.

[Fan15] W. Fan. Data quality: From theory to practice. *SIGMOD Record*, 44(3):7–18, 2015.

[FG12] Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[FGG+13] Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, and Andrew Jon Sellers. Oxpath: A language for scalable data

extraction, automation, and crawling on the deep web. *VLDB J.*, 22(1):47–72, 2013.

[FGG+14] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Diadem: Thousands of websites to a single database. *Proc. VLDB Endow.*, 7(14):1845–1856, October 2014.

[FGL+16] Tim Furche, Georg Gottlob, Leonid Libkin, Giorgio Orsi, and Norman W. Paton. Data wrangling for big data: Challenges and opportunities. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 473–478, 2016.

[FKM+00] R. Fenk, A. Kawakami, V. Markl, R. Bayer, and S. Osaki. Bulk loading a data warehouse built upon a ub-tree. In *Proceedings 2000 International Database Engineering and Applications Symposium (Cat. No.PR00789)*, pages 179–187, 2000.

[FLM+12] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. Towards certain fixes with editing rules and master data. *VLDB J.*, 21(2):213–238, 2012.

[FMQ+18] R.C. Fernandez, E. Mansour, E.E. Qahtan, A.K. Elmagarmid, I.F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*, 2018.

[GD04] Kristen Grauman and Trevor Darrell. Fast contour matching using approximate earth mover's distance. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), with CD-ROM, 27 June - 2 July 2004, Washington, DC, USA*, pages 220–227, 2004.

[GFSS00] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Ajax: An extensible data cleaning tool. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 590–, 2000.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, 1999.

[GMJB17] E. Grave, T. Mikolov, A. Joulin, and P. Bojanowski. Bag of tricks for efficient text classification. In *EACL*, pages 427–431, 2017.

[GPS17] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1-2):1–119, 2017.

[Gry98] Jarek Gryz. Query folding with inclusion dependencies. In *Proceedings of the Fourteenth International Conference on Data Engineering,Orlando, Florida, USA, February 23-27, 1998*, pages 126–133, 1998.

[Gul10] Sumit Gulwani. Dimensions in program synthesis. In *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 26-28, 2010, Hagenberg, Austria*, pages 13–24, 2010.

[Gul11] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 317–330, 2011.

[Gul12] Sumit Gulwani. Synthesis from examples: Interaction models and algorithms. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, pages 8–14, 2012.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.

[HCG+18] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek R. Narasayya, and Surajit Chaudhuri. Transform-data-by-example (TDE): an

extensible search engine for data transformations. *PVLDB*, 11(10):1165–1177, 2018.

[HCL+08] C. Hsieh, K Chang, C. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, pages 408–415, 2008.

[HG11] William R. Harris and Sumit Gulwani. Spreadsheet table transformations from examples. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 317–328, 2011.

[HGQ16] Rihan Hai, Sandra Geisler, and Christoph Quix. Constance: An intelligent data lake system. In *SIGMOD*, pages 2097–2100, 2016.

[HHK15] Jeffrey Heer, Joseph M. Hellerstein, and Sean Kandel. Predictive interaction for data transformation. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[HHK18] Joseph M. Hellerstein, Jeffrey Heer, and Sean Kandel. Self-service data preparation: Research to practice. *IEEE Data Eng. Bull.*, 41(2):23–34, 2018.

[HHK19] Jeffrey Heer, Joseph M. Hellerstein, and Sean Kandel. Data wrangling. In *Encyclopedia of Big Data Technologies.* 2019.

[HKN+16] Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 795–806, 2016.

[HRO06] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: The teenage years. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 9–16, 2006.

[HSG+17] Joseph M. Hellerstein, Vikram Sreekanti, Joseph E. Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, Mark Donsky, Gabriel Fierro, Chang She, Carl Steinbach, Venkat Subramanian, and Eric Sun. Ground: A data context service. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, 1998.

[Inm] William H. Inmon. a tale of two architectures. `https://release.nl/magazines/Aveq/118597.pdf`. Accessed: 2019-04-09.

[Inm92] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., 1992.

[JACJ17] Zhongjun Jin, Michael R. Anderson, Michael J. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 683–698, 2017.

[JM09] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009.

[JT12] Rajni Jindal and Shweta Taneja. Comparative study of data warehouse design approaches : A survey. *International Journal of Database Management Systems*, 4:33–45, 02 2012.

[KAB+19] Nikolaos Konstantinou, Edward Abel, Luigi Bellomarini, Alex Bogatu, Cristina Civili, Endri Irfanie, Martin Koehler, Lacramioara Mazilu, Emanuel Sallinger, Alvaro Fernandes, Georg Gottlob, John

Keane, and Norman Paton. Vada: An architecture for end user informed data preparation. *Journal of Big Data*, 7 2019.

[KBC+17] Martin Koehler, Alex Bogatu, Cristina Civili, Nikolaos Konstantinou, Edward Abel, Alvaro A. A. Fernandes, John A. Keane, Leonid Libkin, and Norman W. Paton. Data context informed data wrangling. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, pages 956–963, 2017.

[KC04] Ralph Kimball and Joe Caserta. *The data warehouse ETL toolkit*. Wiley, 2004.

[KCH+03] Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee. A taxonomy of dirty data. *Data Min. Knowl. Discov.*, 7(1):81–99, January 2003.

[KHP+11] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.

[KIJ+15] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. Bigdansing: A system for big data cleansing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1215–1230, 2015.

[KKA+17] Nikolaos Konstantinou, Martin Koehler, Edward Abel, Cristina Civili, Bernd Neumayr, Emanuel Sallinger, Alvaro A.A. Fernandes, Georg Gottlob, John A. Keane, Leonid Libkin, and Norman W. Paton. The vada architecture for cost-effective data wrangling. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1599–1602. ACM, 2017.

[KL51] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 03 1951.

[KL03] Ronald S. King and James J. Legendre. Discovery of functional and approximate functional dependencies in relational databases. *JAMDS*, 7(1):49–59, 2003.

[Kot02] Yannis Kotidis. *Aggregate View Management in Data Warehouses*, pages 711–741. Springer US, 2002.

[KP08] Gal Katz and Doron A. Peled. Genetic programming and model checking: Synthesizing new mutual exclusion algorithms. In *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings*, pages 33–47, 2008.

[KP19] Nikolaos Konstantinou and Norman W. Paton. Feedback driven improvement of data preparation pipelines. In *Proceedings of the 21st International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT Joint Conference, DOLAP@EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*, 2019.

[KPD+17] Sebastian Kruse, Thorsten Papenbrock, Christian Dullweber, Moritz Finke, Manuel Hegner, Martin Zabel, Christian Zöllner, and Felix Naumann. Fast approximate discovery of inclusion dependencies. In *Datenbanksysteme für Business, Technologie und Web (BTW2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings*, pages 207–226, 2017.

[KPHH11] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 3363–3372, 2011.

[KPHH12] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2917–2926, 2012.

[KPN15] Sebastian Kruse, Thorsten Papenbrock, and Felix Naumann. Scaling out the discovery of inclusion dependencies. In *Datenbanksysteme für Business, Technologie und Web (BTW),16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 4.-6.3.2015 in Hamburg, Germany. Proceedings*, pages 445–454, 2015.

[KR02] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. Wiley, 2002.

[Kus97] Nicholas Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, 1997.

[LB17] O. Lehmberg and C. Bizer. Stitching web tables for improving matching quality. *PVLDB*, 10(11):1502–1513, 2017.

[LDW00] Tessa A. Lau, Pedro M. Domingos, and Daniel S. Weld. Version space algebra and its application to programming by demonstration. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 527–534, 2000.

[LG14] Vu Le and Sumit Gulwani. Flashextract: a framework for data extraction by examples. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, pages 542–553, 2014.

[LHWY13] X. Ling, A. Y. Halevy, F. Wu, and C. Yu. Synthesizing union tables from the web. In *IJCAI*, pages 2677–2683, 2013.

[LRG+18] Massimo Lusetti, Tatyana Ruzsics, Anne Gohring, Tanja Samardzic, and Elisabeth Stark. Encoder-decoder methods for text normalization. In *Proceedings of the Fifth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2018)*, 08 2018.

[LRNdST02] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93, June 2002.

[LSC10] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3(1):1338–1347, 2010.

[LV00] Mark Levene and Millist W. Vincent. Justification for inclusion dependency normal form. *IEEE Trans. Knowl. Data Eng.*, 12(2):281–291, 2000.

[MFH+17] Xianling Mao, Bo-Si Feng, Yi-Jing Hao, Liqiang Nie, Heyan Huang, and Guihua Wen. S2JSD-LSH: A locality-sensitive hashing schema for probability distributions. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3244–3251, 2017.

[MHH+01] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[Mil95] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[Mit82] Tom M. Mitchell. Generalization as search. *Artif. Intell.*, 18(2):203–226, 1982.

[MJDS07] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 141–150, 2007.

[MKK+08] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proc. VLDB Endow.*, 1(2):1241–1252, August 2008.

[MMP+11] Bruno Marnette, Giansalvatore Mecca, Paolo Papotti, Salvatore Raunich, and Donatello Santoro. ++spicy: an opensource tool for second-generation schema mapping and data exchange. *PVLDB*, 4(12):1438–1441, 2011.

[MNZ+18] Renée J. Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q. Pu, and Periklis Andritsos. Making open data transparent: Data discovery on open data. *IEEE Data Eng. Bull.*, 41(2):59–70, 2018.

[Moh02] Mehryar Mohri. Edit-distance of weighted automata. In *Implementation and Application of Automata, 7th International Conference, CIAA 2002, Tours, France, July 3-5, 2002, Revised Papers*, pages 1–23, 2002.

[MPFK19] Lacramioara Mazilu, Norman W. Paton, Alvaro A. A. Fernandes, and Martin Koehler. Dynamap: Schema mapping generation in the wild. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management, SSDBM 2019, Santa Cruz, CA, USA, July 23-25, 2019*, pages 37–48, 2019.

[MPR02] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[MPR09] G. Mecca, P. Papotti, and S. Raunich. Core schema mappings. In *SIGMOD*, pages 655–668, 2009.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[NH10] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[Nix85] Robert P. Nix. Editing by example. *ACM Trans. Program. Lang. Syst.*, 7(4):600–621, 1985.

[Noy04] Natalya F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70, December 2004.

[NYt] Janitor work. `www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html`. Accessed: 2019-08-21.

[NZPM18] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.

[Oil] Data is the new oil. `www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data`. Accessed: 2020-02-11.

[ORRHG05] Paulo Oliveira, Fatima Rodrigues, Pedro Rangel Henriques, and Helena Galhardas. A taxonomy of data quality problems. *Journal of Data and Information Quality - JDIQ*, 01 2005.

[PAKS16] Minh Pham, Suresh Alse, Craig A. Knoblock, and Pedro A. Szekely. Semantic labeling: A domain-independent approach. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, pages 446–462, 2016.

[Pap19] Paolo Papotti. Schema mapping. In *Encyclopedia of Big Data Technologies.* 2019.

[Pat19] Norman W. Paton. Automating data preparation: Can we? should we? must we? In *Proceedings of the 21st International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT Joint Conference, DOLAP@EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*, 2019.

[PBE+16] Norman W. Paton, Khalid Belhajjame, Suzanne M. Embury, Alvaro A. A. Fernandes, and Ruhaila Maskat. Pay-as-you-go data integration: Experiences and recurring themes. In *SOFSEM 2016: Theory and Practice of Computer Science - 42nd International Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 23-28, 2016, Proceedings*, pages 81–92, 2016.

[PBF+15] T. Papenbrock, T Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome. *PVLDB*, 8(12):1860–1863, 2015.

[PKQN15] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. Divide & conquer-based inclusion dependency discovery. *PVLDB*, 8(7):774–785, 2015.

[PN16] Thorsten Papenbrock and Felix Naumann. A hybrid approach to functional dependency discovery. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 821–833, 2016.

[PSC+15] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J. Miller, and Divesh Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4):300–311, 2015.

[QH19] Christoph Quix and Rihan Hai. Data lake. In *Encyclopedia of Big Data Technologies*. 2019.

[QHV16] Christoph Quix, Rihan Hai, and Ivan Vatov. GEMMS: A generic and extensible metadata management system for data lakes. In *CAiSE 2016*, pages 129–136, 2016.

[Rah11] Erhard Rahm. Towards large-scale schema and ontology matching. In *Schema Matching and Mapping*, pages 3–27. 2011.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001.

[RCIR17] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[RCME11] A Ritter, S. Clark, Mausam, and O. Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, 2011.

[RD00] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23:2000, 2000.

[RG17] Mohammad Raza and Sumit Gulwani. Automated data extraction using predictive program synthesis. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 882–890, 2017.

[RGM14] Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. Programming by example using least general generalizations. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 283–290, 2014.

[RH01] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 381–390, 2001.

[RHH+17] Tye Rattenbury, Joseph M. Hellerstein, Jeffrey Heer, Sean Kandel, and Connor Carreras. *Principles of Data Wrangling.* OâĂŹReilly Media, Inc., 2017.

[RKR97] Nick Roussopoulos, Yannis Kotidis, and Mema Roussopoulos. Cubetree: Organization of and bulk incremental updates on the data cube. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 89–99, 1997.

[RLU14] Anand Rajaraman, Jure Leskovec, and Jeffrey Ullman. *Mining of Massive Datasets.* 01 2014.

[Rou98] Nick Roussopoulos. Materialized views and data warehouses. *SIGMOD Rec.*, 27(1):21–26, March 1998.

[RS10] Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. pages 45–50, 05 2010.

[SAPS19] S. Sampaio, M. Aljubairah, H. A. Permana, and P. Sampaio. A conceptual approach for supporting traffic data wrangling tasks. *Comput. J.*, 62(3), 2019.

[SBI+13] Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, George Beskales, Mitch Cherniack, Stanley B. Zdonik, Alexander Pagan, and Shan Xu. Data curation at scale: The data tamer system. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[SDH08] Anish Das Sarma, Xin Dong, and Alon Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 861–874, 2008.

[SDRK02] Yannis Sismanis, Antonios Deligiannakis, Nick Roussopoulos, and Yannis Kotidis. Dwarf: Shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 464–475, 2002.

[SE13] P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, Jan 2013.

[SG16] Rishabh Singh and Sumit Gulwani. Transforming spreadsheet data types using examples. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 343–356, 2016.

[SGF13] Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. Template-based program verification and program synthesis. *STTT*, 15(5-6):497–518, 2013.

[Sha19] Mehul Shah. Data warehouses are dead, long live data warehousing!

In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*, 2019.

[SI18] Michael Stonebraker and Ihab F. Ilyas. Data integration: The current status and the way forward. *IEEE Data Eng. Bull.*, 41(2):3–9, 2018.

[Sin16] Rishabh Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *PVLDB*, 9(10):816–827, 2016.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 697–706, 2007.

[SL15] Anshumali Shrivastava and Ping Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 981–991, 2015.

[SLH12] M. Slaney, Y. Lifshits, and J. He. Optimal parameters for locality-sensitive hashing. *Proceedings of the IEEE*, 100(9):2604–2623, Sep. 2012.

[SSB+18] Juliano Efson Sales, Leonardo Souza, Siamak Barzegar, Brian Davis, André Freitas, and Siegfried Handschuh. Indra: A word embedding and semantic relatedness server. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018.*, 2018.

[TP10] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *J. Artif. Intell. Res.*, 37:141–188, 2010.

[TSRC15] Ignacio G. Terrizzano, Peter M. Schwarz, Mary Roth, and John E. Colino. Data wrangling: The challenging yourney from the wild to the lake. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015*, 2015.

[Vas09] Panos Vassiliadis. A survey of extract-transform-load technology. *International Journal of Data Warehousing and Mining*, 5:1–27, 07 2009.

[VHM+11] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *PVLDB*, 4(9):528–538, 2011.

[VS99] Panos Vassiliadis and Timos Sellis. A survey of logical models for olap databases. *SIGMOD Rec.*, 28(4):64–69, December 1999.

[WA15] Coral Walker and Hassan H. Alrehamy. Personal data lake with data gravity pull. In *Fifth IEEE International Conference on Big Data and Cloud Computing, BDCloud 2015, Dalian, China, August 26-28, 2015*, pages 160–167, 2015.

[Wik] Worldwide big data market forecast. `wikibon.com/2016-2026-worldwide-big-data-market-forecast`. Accessed: 2019-03-30.

[WK16] Bo Wu and Craig A. Knoblock. Maximizing correctness with minimal user effort to learn data transformations. In *Proceedings of the 21st International Conference on Intelligent User Interfaces, IUI 2016, Sonoma, CA, USA, March 7-10, 2016*, pages 375–384, 2016.

[WLF11] Jiannan Wang, Guoliang Li, and Jianhua Feng. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 458–469, 2011.

[WLF14] Jiannan Wang, Guoliang Li, and Jianhua Feng. Extending string similarity join to tolerant fuzzy token matching. *ACM Trans. Database Syst.*, 39(1):1–45, 2014.

[WT14] Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 457–468, 2014.

[WTSSJ14] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. 08 2014.

[YGCC12] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108, 2012.

[YLDF16] Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. String similarity search and join: a survey. *Frontiers of Computer Science*, 10(3):399–417, Jun 2016.

[ZC09] Luke S. Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*, pages 976–984, 2009.

[ZHC17] Erkang Zhu, Yeye He, and Surajit Chaudhuri. Auto-join: Joining tables by leveraging transformations. *PVLDB*, 10(10):1034–1045, 2017.

[ZHO+10] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. On multi-column foreign key discovery. *PVLDB*, 3(1):805–814, 2010.