TECHNISCHE UNIVERSITÄT DRESDEN

Query-Time Data Integration

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl. Medien-Inf. Julian Eberius
geboren am 28. September 1987 in Dresden

Gutachter:	**Prof. Dr.-Ing. Wolfgang Lehner** Technische Universität Dresden Fakultät Informatik, Institut für Systemarchitektur Lehrstuhl für Datenbanken 01062 Dresden
	Prof. Dr.-Ing. Erhard Rahm Universität Leipzig Fakultät Mathematik und Informatik, Institut für Informatik Abteilung Datenbanken 04109 Leipzig
Tag der Verteidigung:	10. Dezember 2015
	Dresden, im Oktober 2015

ABSTRACT

Today, data is collected in ever increasing scale and variety, opening up enormous potential for new insights and data-centric products. However, in many cases the volume and heterogeneity of new data sources precludes up-front integration using traditional ETL processes and data warehouses. In some cases, it is even unclear if and in what context the collected data will be utilized. Therefore, there is a need for agile methods that defer the effort of integration until the usage context is established.

This thesis introduces *Query-Time Data Integration* as an alternative concept to traditional up-front integration. It aims at enabling users to issue ad-hoc queries on their own data as if all potential other data sources were already integrated, without declaring specific sources and mappings to use. Automated data search and integration methods are then coupled directly with query processing on the available data. The ambiguity and uncertainty introduced through fully automated retrieval and mapping methods is compensated by answering those queries with ranked lists of alternative results. Each result is then based on different data sources or query interpretations, allowing users to pick the result most suitable to their information need.

To this end, this thesis makes three main contributions. Firstly, we introduce a novel method for *Top-k Entity Augmentation*, which is able to construct a top-k list of consistent integration results from a large corpus of heterogeneous data sources. It improves on the state-of-the-art by producing a set of individually consistent, but mutually diverse, set of alternative solutions, while minimizing the number of data sources used. Secondly, based on this novel augmentation method, we introduce the *DrillBeyond* system, which is able to process Open World SQL queries, i.e., queries referencing arbitrary attributes not defined in the queried database. The original database is then augmented at query time with Web data sources providing those attributes. Its hybrid augmentation/relational query processing enables the use of ad-hoc data search and integration in data analysis queries, and improves both performance and quality when compared to using separate systems for the two tasks. Finally, we studied the management of large-scale dataset corpora such as data lakes or Open Data platforms, which are used as data sources for our augmentation methods. We introduce *Publish-time Data Integration* as a new technique for data curation systems managing such corpora, which aims at improving the individual reusability of datasets without requiring up-front global integration. This is achieved by automatically generating metadata and format recommendations, allowing publishers to enhance their datasets with minimal effort.

Collectively, these three contributions are the foundation of a Query-time Data Integration architecture, that enables ad-hoc data search and integration queries over large heterogeneous dataset collections.

ACKNOWLEDGEMENTS

First of all, I want to thank my advisor Professor Wolfgang Lehner for giving me the opportunity to work in his group, pursue my research and write this thesis. He encourages and maintains an atmosphere of traditional academic freedom, giving people the liberty and time to think, and to develop their work. This can, by all accounts, not be taken for granted in today's research landscape. At the same time, he brings discussion, contacts, and all kinds of opportunities into the group, and is always there to support his students. I'm grateful that I could spend four years with the colorful group of individuals he assembled.

Which brings me to the next point: my colleagues. I want to express my thanks to all of them, starting with those that I worked most closely with. From our first steps in Open Data and do-it-yourself mashup research, to our final battles with the intricacies of Web data integration, the members of the original junior research group Edyra, Katrin Braunschweig and Maik Thiele, have contributed the most to my thesis. Though the word "Open Data" quickly became the target of some ridicule by the group's jesters (but what was not?), we persevered and produced respectable results that do not need to be hidden from anybody. Thanks to Ulrike Fischer and Benjamin Schlegel, with whom I shortly shared the best room of the building in a friendly and easy-going atmosphere. I want to give special thanks to Martin Hahmann, with whom I shared an office for most of the second half of my stay in the group. He is easy to laugh with but also available for more serious discussion about all aspects of life. Next, I want to thank all other colleagues that frequently had to endure my need for communication (or ranting), even though they had work of their own to do: Claudio Hartmann, Hannes Voigt, Robert Ulbricht, and Tim Kiefer. I also want to thank all other colleagues that at some point helped creating the fun and creative atmosphere in the group: Ahmad, Annett, Bernd, Bernhard, Dirk, Elena, Frank, Gunnar, Ines, Johannes, Juliana, Kai, Kasun, the two Larses, Laurynas, Matthias, Patrick, Peter, Rihan, Steffen, Thomas, Till, Tobias, Tomas, Ulrike Schöbel, and Vasileios. I want to further thank the students that contributed to my doctoral work, Christopher Werner, Patrick Damme, Mihail Maxacov, Paul Peschel and Markus Hentsch.

I am very grateful to Professor Erhard Rahm for co-refereeing this work. I also want to thank one of his doctoral students, Eric Peukert, who, in his function as my Diploma advisor, introduced me to the science aspects of computer science, and sent me on my way to a doctoral work in this group. Special thanks also to Esther Schiwek for interdisciplinary co-working while writing this thesis and for providing medical background information. I want to thank Maik Thiele one more time, for tirelessly and timely reviewing and proof-reading this thesis, offering feedback and always encouraging me when I had doubts.

Finally, I want to thank my family, especially my parent and my brothers, for giving me the background and the opportunities that led me to undertake this work in the first place, and for their continued support throughout the years. They never had any doubt that I would do it, and supported me in any way they could. I want to thank my own new family, my children Maia and Lorenz for making me laugh and giving my work purpose, and most importantly my fiancée Katharina. Especially in the last months of writing this thesis, she kept everything around me running, and backed me up in every way possible to allow me to finish this thesis.
For that I will always be grateful.

Julian Eberius
October 15, 2015

CONTENTS

INTRODUCTION

While the term *Big Data* is most often associated with the challenges and opportunities of today's growth in data volume and velocity, the phenomenon is also characterized by the increasing *variety* of data (Laney, 2001). In fact, data is collected in more and more different forms, and from increasingly heterogeneous sources. The spectrum of additional data sources ranges from large-scale sensor networks, over measurements from mobile clients or industrial machinery, to the log- and click-streams of ever more complex software architectures and applications. In addition, there is more publicly available data, such as social network data, as well as Web- and Open Data. More and more organizations strife to efficiently harness all forms and sources of data in their analysis projects to gain new insights, or enable new features in their products. While generating value out of these novel forms of data is not trivial and requires new methods, algorithms, and tools, their potential is universally recognized (Labrinidis and Jagadish, 2012). In fact, Big Data technologies and methods are being applied across many application domains, from business (Manyika et al., 2011), over natural sciences (Marx, 2013) to health care (Ferrucci et al., 2013).

New opportunities always come with their own set of novel challenges. In the case of Big Data they comprise the entire data life cycle, from acquisition to interpretation, and introduce a wide range of new problems, including issues of scale, timeliness, privacy and visualization, just to name a few (Jagadish et al., 2014). One specific challenge related to the *variety* aspect of Big Data, or in other words, the heterogeneity and multitude of data sources utilized today, is *data integration*. In this thesis, we focus on this data integration aspect, especially in relation to changing data management practices. These include analytical processes that concern a larger variety of non-traditional data users and contexts, that are based on ad-hoc information needs, and are not easily mapped to traditional analytical architectures.

The structure of this introductory chapter is as follows: In the next section, we will briefly introduce the notion of agile data integration, discuss current developments and identify open challenges. In Section 1.2, we will present a novel query-paradigm for agile data integration, called *Query-time Data Integration*. Furthermore, we will sketch the individual contributions and present a high-level architecture that serves as an outline for this thesis.

1.1 CHALLENGES IN AGILE DATA INTEGRATION

Data integration is a common problem in data management, which deals with combining data of different origins and making it usable in a unified form. In general, it is a laborious and mostly manual process that has to be performed ahead-of-time, i.e., before queries on the combined data can be issued. Due to its complexity, it is usually performed by experts, for example ETL and integration specialists. In the era of Big Data, with an increasing variety of data sources, the traditional challenges in data integration, that is to say bridging the heterogeneity and ambiguity between data sources at schema- as well as instance level, are only becoming more complicated (Dong and Srivastava, 2013).

At the same time, data-driven approaches are applied in increasing numbers of contexts, involving more and more user groups outside of traditional IT. Novel *agile* approaches to data management, such as MAD (Cohen et al., 2009), are complementing or replacing the static processes of data warehouse infrastructures. It is increasingly recognized that organizations

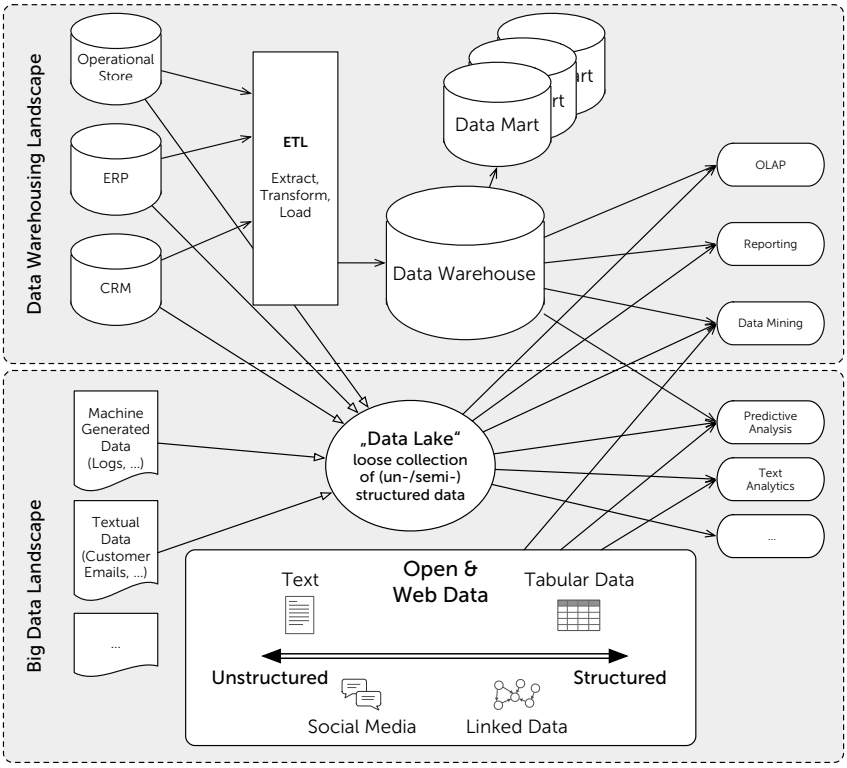


Figure 1.1: The growing Big Data analytics landscape

profit from enabling domain experts to perform their own data management and analysis tasks without or with less involvement of IT personnel (McAfee and Brynjolfsson, 2012). This direct access to data and methods is especially critical because these user's information needs are often *ad-hoc* or *situational* (Marchionini, 2006), or require the use of heterogeneous or unstructured data that is not integrated in a data warehouse. The title "data scientist" is becoming a standard term for highly trained data professionals that perform these kinds of agile analytics as a contrast to the more traditional "business analyst", who is mainly concerned with classical BI (Davenport and Patil, 2012). In addition, a new class of users, sometimes called *data enthusiasts* (Morton et al., 2014), is becoming increasingly prominent. These users also want to utilize data to support decisions or illustrate a story, but are lacking in formal education that is the hallmark of the data scientist. They are another driving factor towards the development of *self-service analytics and integration*.

However, conventional data infrastructures assume controlled ETL processes with well-defined input and target schemata, that define the data pipelines in the organization. Consider the upper part of Figure 1.1, which represents this "traditional" architecture. The data sources typically are operational databases and common enterprise IT systems, such as Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) systems. Traditionally, the data sink in such an architecture has been the warehouse or data marts, whose schemata define what is immediately queryable for an analyst. If there is an ad-hoc information need that can not be satisfied with the current schema because external information has to be in-

tegrated, an intricate process has to be followed. Because the warehouse is a crucial piece of infrastructure it is highly controlled: ad-hoc integration is not a feature that it is designed for.

Still, as discussed above, organizations today aim at generating value from all available data, which includes novel internal, but also increasingly external sources. Consider the lower part of Figure 1.1, which depicts key changes to the traditional architecture. First, it includes new data sources that are coming into focus of enterprise analytics. These range from semi-structured machine generated data such as logs of servers or production equipment, to purely textual data such as user complaints or product feedback. It also depicts the growing amount of external data sources discussed above, such as Web- and (Linked) Open Data, as well as social media data. The figure further incorporates a novel component that is gaining importance, which is a central but schema-less repository for semi- or unstructured data sets. It is labeled "*data lake*" (Mohanty et al., 2015; O'Leary, 2014) in Figure 1.1, though the term for this component is not completely established yet. The introduction of such a repository is often necessitated by the mentioned variety of data sources. New datasets are now often collected at a rate that surpasses the organizational capabilities to integrate them with a central schema. In many cases instant integration is not even the desirable, as the future use cases of the data is not known.

So while a new wealth of data is available, the integration of a large variety of sources is still a complicated, laborious and mostly manual process that has to be performed by experts, that is required before queries on the combined data can be issued. This obviously collides with the aim of enabling domain users to generate value from Big Data directly with little IT support. It is well established in traditional warehousing that the majority of the effort in a complex analysis project is actually the integration of heterogeneous data sources (Dasu and Johnson, 2003), which has to be performed before any actual analytical methods can be applied. This limits the ability of non-expert users to include more data into their analysis tasks. Without additional tool support, the effort of data integration will likely prevent those users from taking advantage of the wealth of possible data sources available today.

Put succinctly, we identify two trends: first, we observe an *increasing availability of valuable, but heterogeneous data sources*, and second, there is an *growing demand for self-service BI and ad-hoc integration*, driven by the increasing number of different user groups and usage contexts for data. So while the first trend enables more and more situations in which data can potentially be enhanced and enriched through integration with external data sources, it also introduces additional complexity. The second trend, however, demands that the tools and processes of data integration become simpler, and able to cater to a larger audience.

1.2 QUERY-TIME DATA INTEGRATION

In the previous section, we discussed several trends in today's data management landscapes. One is the ever increasing amount and diversity of available data sources, both inside organizations and outside. Organizations are stockpiling data on a level of quantity and variety not seen in the past. At the same time, developments such as the Open Data trend and sophisticated technologies for Web data extraction lead to higher availability of public data sources.

The second trend is broadening spectrum of data users. While data processing and analysis

traditionally were the domain of IT departments and BI specialists, more and more users from other departments are becoming active data users and require easy access to all valuable data sources, both inside and outside organizational boundaries. In addition, the focus of data use has broadened: Traditional BI processed focused on defined data flows, typically from source systems to a data warehouse, and aimed at producing well-defined reports or dashboards. However, a new type of investigative data analysis, often associated with the role of the *data scientist*, increases demand for ad-hoc integration of the variety of data sources mentioned above.

Still, even with all these new sources for data, the focus of most analysis task is on the respective core data of each organization. This most valuable data will still be stored in controlled data warehouse environments with defined schemata. Therefore, new ad-hoc data integration techniques need to take this analytics context into account.

From these observations, we will now derive requirements for a novel data management architecture focused on fulfilling the need for self-service, ad-hoc integration of heterogeneous data sources with traditional data management systems.

1.2.1 Requirements

This thesis aims at aligning the mentioned trends, increasing *availability of heterogeneous data sources* and increasing demand for *self-service, ad-hoc integration*. This entails enabling users to tap the power of vast amounts of diverse data sources for their analysis tasks. Specifically, in this thesis we will introduce methods and systems aimed at fulfilling the following requirements:

Exploratory and Ad-hoc Data Search and Integration The novel data sources discussed above do not lend themselves to classical data integration with centralized schemata and expert-defined mappings and transformations. Data from Web and Open sources, as well as from heterogeneous management platforms such as data lakes, should be integrated dynamically at the time it is needed by a user. The volume and variety of data sources in these scenarios makes single, centralized integration effort infeasible. Instead of consolidating all available sources as soon as they are available, sources should be retrieved and integrated based on a specific information need. In such a scenario, data search takes the place of exhaustive schemata covering all available information. Throughout this thesis, we will discuss these argument in detail, and show how exploratory analysis over large numbers of heterogeneous sources can be supported with the right methods and systems.

Minimal Up-front User Effort When developing methods for answering ad-hoc information needs, an important factor is the minimization of up-front costs for the user. In other words, our ambition is to minimize the user effort necessary before first results become available. Specifically, we want to reduce the reliance on IT personnel and integration experts, or the need to search data repositories or master integration tools before external sources can be included in a data analysis scenario. In the ideal case, a user should be able to declaratively specify the missing data and be presented with workable answers automatically.

Further, this goal should not only apply to the users that search information. In a heterogeneous information landscape, there will also be users concerned with the management of the dataset corpora, e.g., a maintainer on an Open Data platform. As discussed above, modern data

management aims at attracting, storing and managing all potentially useful data that emerges, even if there is no clear use case established initially. This, however, implies that it is infeasible to expend significant effort for the preprocessing of all data. Therefore, just moving the effort from the querying user to the publisher or manager of the queried data is not a solution. Instead, we need additional support for managing large, heterogeneous dataset repositories that do not require up-front investment on the part of the data owner.

In this thesis, we will therefore introduce techniques for supporting both data users as well as data managers.

Trustworthy Automated Integration The two goals introduced above amount to reducing user effort and time-to-result in data search and integration. To answer an ad-hoc query that depends on external sources, a traditional manual and expert-driven integration process is not ideal. As a consequence, this thesis will propose novel methods for automating these highly involved processes. However, exact data integration has been called an "AI-complete" problem (Halevy et al., 2006), and is generally considered not to be automatically solvable in the general case. In fact, all proposed methods return results with varying confidence values, instead of perfect answers, requiring human validation in many cases. The same can be said for information retrieval techniques, on which automated data source search methods are based. In retrieval systems such as search engines, the uncertainty and ambiguity in user query and data sources is handled by returning ranked result lists instead of single answers.

Consequently, methods for automatically answering ad-hoc queries based on external sources will also have to deal with uncertain results. In this thesis, we will therefore introduce methods that automatize the data search and integration processes as far as possible, while also facilitating user understanding and verification of the produced results.

System Integration Finally, ad-hoc integration queries should not introduce an isolated, new class of systems into existing data management architectures. As discussed above, the wealth of new data sources collected by organizations or found on the Web promises opportunities. However, most data analysis tasks will still focus on the core databases inside organizations, with ad-hoc integrated sources supplementing, not replacing, them. Therefore, methods developed to support ad-hoc integration can not be deployed in a vacuum, but need to work hand in hand with systems managing core data.

In conclusion, for both usability and performance reasons, it is necessary to integrate the novel facilities to be developed in this thesis with the relevant existing data analysis systems.

To fulfill these requirements, we will propose a new query model, *Query-time Data Integration*, in Section 1.2.2, and a new data integration systems architecture implementing it in Section 1.2.3. This query-time integration architecture will also form the outline of this thesis' contributions, which will be discussed in the following chapters.

1.2.2 Query Model

We will now introduce our notion of *Query-time Data Integration*. Consider Figure 1.2, which gives a high-level overview of a manual process for an ad-hoc database query that depends

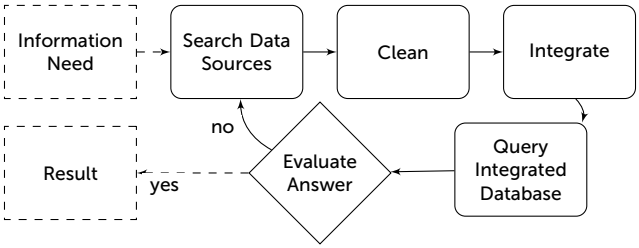


Figure 1.2: Iterative ahead-of-time integration for an ad-hoc query

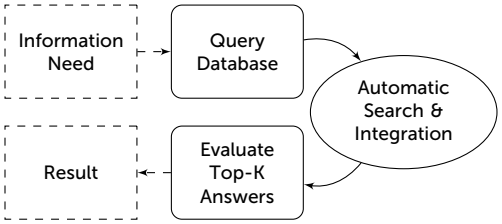


Figure 1.3: An ad-hoc query with top-k query-driven data integration

on data from yet unknown external sources. Before the user can pose the actual query, data sources that contain relevant data for integration need to be identified manually, for example through a regular search engine. Then, the data has to be extracted and cleaned, i.e., converted into a form that is usable in a regular database. In a next step, it needs to be integrated with the existing database, which includes mapping of corresponding concepts, but may also include instance-level transformations. Only after this process is finished, the original query can be issued. Still, the result may not be what the user originally desired, or the user may want to see the query result when using a different Web data source. In this case, another iteration of the process is necessary.

In this thesis, we propose *Query-time Data Integration* as an alternative concept to ahead-of-time data integration. We aim at allowing a user to issue ad-hoc queries on a database while referencing data to be integrated as if it were already defined in the database, without providing specific sources or mappings. We call such database queries *Open World Queries*, since they are defined over data not contained in the database. The goal is to enable users to specify information needs requiring external data declaratively, just as if only local data was used, without having to integrate data up-front. The database system then automatizes the process of retrieving and integrating possible Web data sources that could be used to answer these queries.

However, a fully automatic retrieval and integration process is not feasible. The underlying methods, information retrieval and automatic matching, both work with top-k results or provide uncertain answers with confidence values. We argue that database query results based on such automatic methods should therefore also reflect the uncertainty of the basic methods, as well as the ambiguity of the query and the uncertainty of the data sources themselves.

To cope with these various sources of uncertainty, we propose to extend the concept of providing top-k alternative answers, well known from general search engines, to structured database queries. Under this paradigm, the system will not respond to Open World Queries

```
select n_name, gdp, avg(o_totalprice)
from nation, customer, orders
where
    n_nationkey=c_nationkey
    and c_custkey=o_custkey
    and gdp > 10.0
group by n_name, gdp
order by gdp desc
```

Figure 1.4: Exemplary Open World SQL query, ad-hoc integrated attribute *gdp* highlighted

with a single, perfect result, as it would be the case with normal database queries. Instead, it should produce a ranked list of possible answers. Each result should be structured according to the user query, but based on different possible data sources and query interpretations. The user can then pick the result most suitable to his or her information need, instead of having to iterate manually. This alternative process is depicted in Figure 1.3.

Let us briefly discuss a specific application of the query-time integration paradigm, which we will call *Open World SQL* queries. We will use this example to illustrate our novel query paradigm here, but will defer detailed discussion of Open World SQL queries to Chapter 3. In short, in an Open World SQL query over a regular relational database, the user is allowed to reference arbitrary additional attributes not defined in this database. An exemplary query is shown in Figure 1.4. This query, based on the TPC-H benchmark schema, aims at analyzing sales total by country. However, it also includes an additional attribute "gdp", which is relevant to the user's analysis as a filter and order criterion, but is not defined in database schema.

We will now consider how the two integration processes introduced above and shown in Figures 1.2 and 1.3 could be used to answer such a query. With ahead-of-time integration the user would be required to manually search for data sources that can possibly provide the missing attribute. Then these sources would need to be integrated with the existing database schema. This is not only laborious, but may even be impossible in some scenarios, such as when working with a data warehouse, where strict integration processes must be adhered to. As illustrated in Figure 1.2, only after the integration step, the original query can be executed and the results inspected, which in turn may reveal the need to integrate other sources if the result is not satisfactory.

On the other hand, with the query-time integration model introduced in this thesis, the user can just specify the query as if the data was already integrated, i.e., by submitting the SQL query as shown in Figure 1.4. The undefined attributes will be automatically materialized as part of regular query processing, by retrieving and integrating external sources able to provide them dynamically. As depicted in Figure 1.3, the result of this process is a list of top-k answers, from which the user can choose the most suitable one. These alternative results can be based on different query interpretations or different data sources. This approach produces immediate integration results, with much less required effort from the user, which makes it ideal for ad-hoc queries. We will discuss the advantages of this query paradigm in more detail in the following chapters of this thesis. Before that, however, we will introduce a high-level system architecture for enabling Query-time Data Integration.

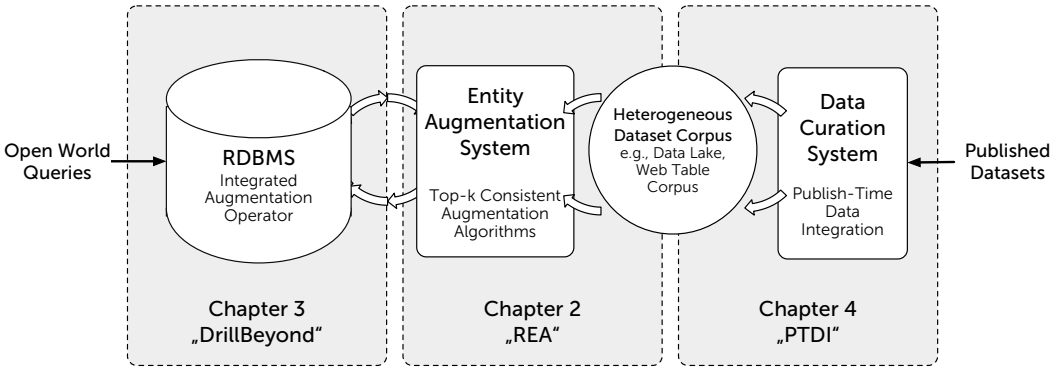


Figure 1.5: Query-time data integration architecture and thesis structure

1.2.3 Architecture and Thesis Contributions

This section introduces our proposed architecture for a query-time integration architecture. It also serves as the outline for the rest of this thesis, and gives a high-level overview of our contributions.

Consider Figure 1.5, which depicts the architecture we are going to introduce, and also marks which chapters in this thesis will discuss each component. Our proposed architecture tackles the problem of ad-hoc integration of heterogeneous sources from several sides. We assume a large collection of external data sources, i.e., a pool of potentially useful but independent and heterogeneous sources, that could be used to fulfill ad-hoc information needs. For the purpose of this high-level architecture, we do not make many assumptions about the nature of this corpus. For example, it could be a corpus of Web tables, i.e., data-carrying tables extracted from the open Web, which we will discuss in more detail in Chapter 2, or a collection of datasets published on a Open Data platform, which we will discuss in Chapter 4. Another example are corporate *data lakes*, also discussed further in Chapter 4, which are systems used to store the semi- or unstructured data that is not immediately integrated in regular ETL processes and data warehouses.

To capitalize on such a loosely coupled corpus of data sources, we propose three systems that complement each other, and together enable our vision of Query-time Data Integration. The first is a system, called *REA*, enables *top-k entity augmentation*, which forms the basic building block for ad-hoc data integration. The second system, called *DrillBeyond*, is an extended RDBMS which can process Open World SQL queries as introduced in Section 1.2.2. The third component is a data curation system based on the novel *Publish-time Data Integration* paradigm for managing heterogeneous dataset collections.

The novel methods and algorithms introduced in these three systems form the main contributions of this thesis, which we will sketch in the following.

REA - Consistent and Diversified Top-k Entity Augmentation In this thesis, we introduce novel algorithms for producing consistent integration results from a large corpus of possible data sources. Specifically, we discuss the *entity augmentation* problem, which aims at extending a given set of entities with an additional, user-requested attribute that is not yet defined for

them. This attribute is typically materialized by automatically retrieving and integrating relevant data sources. We discuss existing approaches and identify their weaknesses with respect to problems such as *Attribute Variations*, *Unclear User Intent*, *Trust & Lineage*, *Exploratory Search* and *Error Tolerance*. We then present a novel approach to the problem, based on an extended version of the set cover problem, called *top-k consistent set covering*. We introduce several algorithms for solving this problem, as well as an implementation called *REA*, which utilizes *Web tables*, i.e., a large corpus of data tables extracted from a general Web crawl, as its data source. We show that our technique improves on the state-of-the-art by producing a set of individually consistent, but mutually diverse, set of alternative solutions instead of a single answer. Our evaluation is based on the Dresden Web Table Corpus, a large-scale corpus of 125M data tables we extracted from a public Web crawl and made available to the research community.

DrillBeyond - Open World SQL queries We then propose a RDBMS/IR system hybrid system called *DrillBeyond*, that allows querying and analyzing a regular relational database while using additional attributes not defined in said database. Using the methods developed previously in this thesis, the original database is then augmented at query time with Web data sources providing those attributes. To this end, our system tightly integrates regular relational processing with data retrieval and integration operators that encapsulate our novel augmentation techniques. This integration allows DrillBeyond to avoid problems we identified with using separate systems for augmentation and query processing, such as *loss of context information* and *unused optimization potential*. We describe challenges in processing this hybrid query type, such as *efficient processing of multi-result SQL queries*, and present how our novel DrillBeyond system solves them. Further, we propose a multitude of methods for optimizing the run-time of this new type of query in the RDBMS-context, and show how usability, efficiency and accuracy are improved when compared to using separate systems.

PTDI - Data Curation at Publish-time As introduced above, we assume a large collection of heterogeneous, individual datasets as the data source for our augmentation techniques. We discuss challenges of managing such collections, and introduce *data curation systems* as a complementary system type to database and data warehouses. We present the results of our survey of governmental Open Data platforms as an example of data curation systems. From its results we derive our concept for improving reusability of datasets managed by data curation systems, called *Publish-time Data Integration*. It aims at improving the reusability of individual datasets, without forcing integration with a global schema. This is achieved by generating recommendations, such as metadata or data format improvements, at publish-time, so that instant feedback of the publisher can be used for verification. We introduce and evaluate two exemplary operations for a publish-time integration-enabled data curation system, *attribute name recommendation* and *relation-extraction for partially structured documents*, to show the concept's viability.

1.2.4 Outline

The organization of this thesis into chapters is shown in the architecture overview in Figure 1.5. In Chapter 2, we will first introduce our novel concept of *top-k entity augmentation using consis-*

tent set covering and its implementation in the *REA* system. We will discuss entity augmentation as a basic operation for ad-hoc data integration, illustrate weaknesses of current methods, and introduce and formalize our concept for solving them. We will then evaluate the *REA* system, with special focus on how well it tackles the problems we identified earlier. The chapter is concluded with review of related work and comparison to our contribution.

In Chapter 3 we will transfer the concept of top-k augmentation into the realm of relational database management. We will show how integration of entity augmentation and query processing can improve usability, efficiency and accuracy when compared to using separate systems for data integration and query processing. We will describe DrillBeyond, its hybrid query processing model and the optimizations we developed, and evaluate with respect to efficiency and accuracy. This chapter likewise concludes with a discussion of related work.

Finally, in Chapter 4, we will introduce the concept of *data curation systems*, and present the results of our survey of governmental Open Data platforms as an example of such systems. We will then introduce the *Publish-time Data Integration* paradigm, and evaluate two specific operations that fit into this framework, attribute name recommendation and extraction of data from partially-structured documents. We will again conclude with a review of related work, both for the overall idea, and for the two specific operations.

Finally, we will summarize this thesis and discuss opportunities for future work in Chapter 5.

TOP-K ENTITY AUGMENTATION

In the previous chapter, we discussed the need to support ad-hoc information needs in analytical scenarios with tools for exploratory data search and lightweight integration. Specifically, we want to enable a user working on a dataset to effortlessly retrieve and integrate other relevant datasets from a large pool of heterogeneous sources. One compelling type of query in this context are so-called *entity augmentation queries*, or *EAQ*. These queries are defined by a set of entities, for example a set of companies, and an attribute that is so far undefined for these entities, for example "revenue". The result of the query should then associate each of the entities with a value for the queried attribute, by automatically retrieving and integrating data sources that can provide them. We will call this attribute the *augmentation* attribute, and the system processing such queries Entity Augmentation System, or *EAS*. The user has to specify the augmented attribute just by a keyword, while the EAS decides on how to lookup data sources, how to match them to the existing data, and possibly how to merge multiple candidate sources into one result. This makes entity augmentation queries both powerful and user-friendly, and thus interesting for exploratory data analysis. Effectively, an EAS aims at automating the process of data search, as well as the following integration step.

In principle, any type of data source could be used for answering entity augmentation queries. For example, a large ontology such as YAGO (Suchanek et al., 2007) could answer some EAQ rather easily, though only if the augmentation attribute is defined in this knowledge base. In recent related work, several systems that process such queries on the basis of large Web table corpora have been proposed, for example *InfoGather* (Yakout et al., 2012; Zhang and Chakrabarti, 2013), the *WWT* system (Pimplikar and Sarawagi, 2012; Sarawagi and Chakrabarti, 2014), and the *Mannheim Search Join Engine* (Lehmberg et al., 2015). An advantage of methods using tables extracted from the Web is that they offer more long tail information, and do not rely on the queried attribute being defined in a central knowledge base. Methods based on Web tables as their data source can, in principle, also be used with any other large collection of heterogeneous data sources. The techniques for automatic data search and integration introduced there could be applied to *data curation systems* (Chapter 4), or enterprise *data lakes* (Mohanty et al., 2015; O'Leary, 2014) as well. In fact, many challenges, such as identifying relevant datasets in light of missing or generic attribute names, bridging semantic or structural differences, or eliminating spurious matches, have already been tackled by existing augmentation systems, which we review in Section 2.6.

Though solving these fundamental issues of data integration remains the most important factor for the success of an EAS, we argue that several other challenges are left unanswered in entity augmentation research. These challenges, which we will motivate and discuss in Section 2.1, are *attribute variations*, *unclear user intent*, query result *trust and lineage*, *exploratory search* and *error tolerance*. From these challenges, we will derive design requirements for a novel entity augmentation method in Section 2.2, namely generating *minimal consistent results* and *diversified top-k augmentation*. Next, in Section 2.3, we will map the entity augmentation problem to an extended version of the *Set Cover* problem, which we call *Top-k Consistent Set Cover*, and derive multiple abstract augmentation algorithms based on this general framework. We will then introduce our *REA* system in Section 2.4, which is a Web table-based implementation of our proposed augmentation method and algorithms. In Section 2.5, we will present an extensive evaluation of the REA system, demonstrating the advantages of our method. Finally, we will survey related work in Section 2.6 and then summarize and discuss our findings

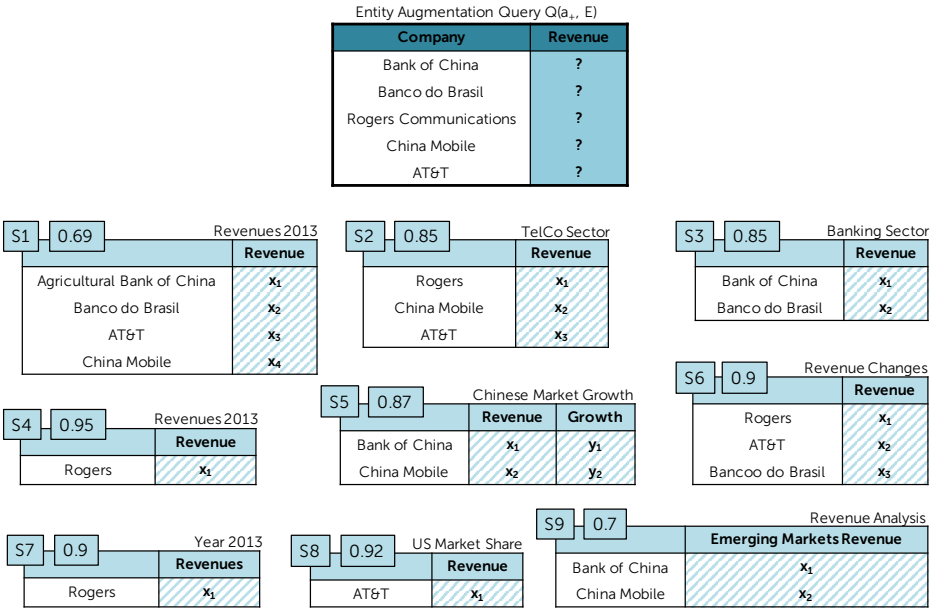


Figure 2.1: Example augmentation scenario: query table and candidate data sources

in Section 2.7.

Parts of the material presented in this chapter have already been published in (Eberius et al., 2015b).

2.1 MOTIVATION AND CHALLENGES

In this section, we will discuss an exemplary entity augmentation scenario based on a heterogeneous table corpus with partially overlapping sources. Our example scenario is depicted in Figure 2.1, with the query represented as a table on the top, and the available candidate data sources below it. The query table consists of a set of five companies, and the augmentation attribute "*revenue*". The candidate data sources depicted below the query table vary in coverage of the query domain, the exact attribute they provide, and their context. They are further annotated with their relevance with respect to the query, which is depicted as a numeric score on each source. We will discuss how exactly this set of candidate sources and the associated relevance scores can be computed in Section 2.4.

For now, let us consider ways in which an augmentation result could potentially be constructed from these sources. Existing methods, having computed the set of candidate sources and relevance scores, compose their query result on a per-entity basis (see Section 2.6). Though more complex algorithms exist, consider as an introductory example an algorithm that picks, for every queried entity, the respective value from the source with the highest relevance score. In our example, this naïve algorithm would pick values from the sources $S7$ for "Rogers", $S8$

for "AT&T", $S5$ for "Bank of China" and "China Mobile", and finally $S3$ for "Banco do Brasil", using the highest ranked available source for each queried entity. This means that the algorithm picks a large number of data sources, almost one distinct source for each entity. More sophisticated methods for pruning candidate tables and picking values from the remaining candidates can be used, such as correlating or clustering sources, mapping sources to knowledge bases, or quality-weighted majority voting (again, see Section 2.6). However, these methods do not fundamentally change the fact that the integration is performed on a by-entity basis, which leads to a large number of distinct sources being used to construct the result. It has been argued that in data integration, and especially selection of sources to integrate, "less is more" (Dong et al., 2013). Choosing to many sources not only increases integration cost, but may even deteriorate result quality if low quality sources are added. For example, adding $S8$ to the example result made the result inconsistent: In contrast to the other sources, this one is, in fact, concerned with US revenue only. This is just one intuitive example of a problem introduced by inadequate source selection and combination in the augmentation process. In the following, we will identify five specific challenges that are insufficiently solved with existing by-entity fusion methods.

Trust and Lineage Our first argument is concerned with the usability of an entity augmentation result. In many application domains, the user can not blindly trust an automatic integration result, no matter how sophisticated the method used is. Rather, the result will serve as a starting point in a semi-automatic integration process, where the user will manually check the sources proposed by the system, correct errors made by the automated integration techniques, and even switch some of the sources for others. Choosing a large number of sources therefore increases the users verification effort. We argue that existing fuse-by-entity models diminish *trust* and hinder understanding of data *lineage*, two properties that are important in the overall process of working with data, because the number of distinct sources they fuse is not considered in source selection. In this chapter, we therefore investigate methods that produce not only consistent augmentation results from several sources, but *minimal* augmentations, i.e., augmentations that use a minimal number of sources to facilitate the usage of the result. In the running example, one such result would be $S2$, $S3$, as it only uses two sources to augment all entities, even though the sources' average score is slightly worse than the score of the naïve solution introduced above. To summarize, when coercing a large number of data sources into one result, properties important for data analysis such as transparency, lineage and trustworthiness of the result are diminished.

Attribute Variations Another problem that we identified with related work is the underlying assumption of a single true value for any combination of entity and augmentation attribute. This single-truth idea, however, does not reflect the complex realities. For example, the augmentation attribute in our scenario was given simply as "revenue" . However, the concept is actually more complex, with many variants such as "US revenue" or "emerging markets revenue" ($S8$ and $S9$) and derived attributes such as "revenue growth" ($S5$ and $S6$). Furthermore, many types of values come with temporal or spatial constraints, such as different years of validity, or may have been measured in different ways, for example using different currencies. Therefore, we argue that even when only sources of high relevance are picked, they may

be correct and of high quality when considered on their own, but still do not form a consistent result for the overall query domain when combined. The differences in precise semantics can, in most cases, not be decided based on the extracted attribute values, but on the level of data sources, for example by considering the context of a table. Even though better methods for creating consistent augmentations for some important dimensions such as time and unit of measurement have been proposed (Zhang and Chakrabarti, 2013; Sarawagi and Chakrabarti, 2014), source consistency is not considered as a general dimension in entity augmentation so far. To summarize the argument: due to the existence of subtle *attribute variations*, the notion of source consistency needs to be taken into account when combining several sources to a single augmentation result.

Unclear User Intent Extending our argument based on the intricacies of attribute variations, we will discuss an additional challenge: the problem of *unclear user intent*. In entity augmentation queries, the information need of the user is relatively underspecified, especially when compared to queries on a database with a defined schema. Even though entity augmentation also operates on structured data, for example on a large-scale Web table corpus, the user is still forced to pose queries on data whose extent or schema is unknown to him or her. Forming precise queries may therefore not always be possible, especially in the presence of attribute variations. In turn, the entity augmentation system may not always be able to pinpoint the exact attribute the user is interested in. To give one example, in the scenario in Figure 2.1 it is unclear whether the user is interested in any specific year, or just in the highest ranked sources. In this example, a solution based on the sources $S1, S4$ may be more useful than the higher-ranked solution $S2, S3$ proposed above, because both sources explicitly state a certain year of validity. However, which solution the user would prefer can not be decided from the query alone.

Exploratory Search Even if the user can specify a precise query, in the case of situational or ad-hoc analysis, they may not even yet know which attribute variant is the most relevant. In those situations, the underspecified nature of entity augmentation queries may even be turned into an advantage. For example, a user may want to stumble on new aspects of his or her information need, similarly to the way it may happen with a Web search engine. In the example scenario, the user may have queried only for "revenue", but a solution showing "revenue growth" based on sources $S5, S6$ may, in some situations, give the ongoing analysis process a new perspective or direction. However, such serendipitous query answering is not supported with current augmentation systems. One partial exception is the Mannheim Search Engine (Lehmberg et al., 2015), which allows so-called unconstrained queries, in which the system returns augmentations for all attributes it can retrieve from all available sources. However, this may leave the user with an unfocused, large and hard-to-comprehend result that is not connected to the information need at hand. In other words, the *exploratory* nature of entity augmentation queries is not done justice in current approaches.

Error Tolerance Finally, we note that all existing augmentation systems are based on techniques for automated schema matching, entity resolution and automated relevance estimation. All these components by themselves have a certain margin of error that, no matter how so-

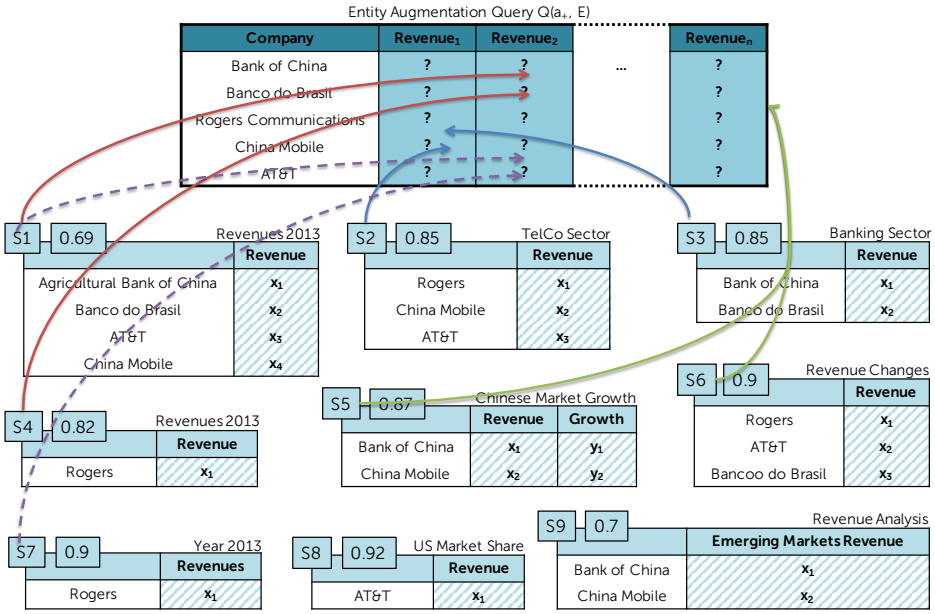


Figure 2.2: Example of a top-k augmentation scenario: possible covers

phisticated the underlying methods become, can never be fully eliminated. In combining these systems to higher-level service such as entity augmentation, the individual errors will even multiply. Still, none of the existing systems, while of course striving to optimize their precision in various ways, offer explicit support for *tolerating* possible errors.

2.2 REQUIREMENTS

Having introduced and discussed five significant open challenges in entity augmentation, we now want to discuss *requirements* for a novel entity augmentation method that alleviates these challenges. First, let us discuss the challenges *lineage* and *attribute variants*. We already discussed that using a large number of sources impedes the user's understanding, and therefore trust of the fused query results. Further, we discussed that, to detect and correctly exploit the existing variants of the queried attribute, we need to take consistency between sources into account. In this chapter, we therefore investigate methods that produce both augmentation results that are both *consistent* and *minimal*, i.e., augmentations that use a minimal number of distinct sources, with each of them representing the same attribute variant. To determine this consistency of datasets, it is possible to measure the similarity between their attribute names, compare global properties of their value sets, or analyze the associated metadata such as dataset titles and context. We will discuss our notion of consistency in more detail and also give formal definitions in the following sections.

Now let us consider two further challenges: *unclear user intent* and *error tolerance*. Both of

these challenges result from various forms of uncertainty: The first from uncertainty about the user intent, the second from uncertainty of the utilized basic methods such as schema matching and entity resolution, and the third from uncertainty in the sources. Information retrieval systems solve this problem by presenting not one exact, but a top-k list of possible results. For example, errors in relevance estimation are tolerable, as long as some relevant documents are included in the top-k result list. Unclear user intent or ambiguities in the query keywords can be resolved by returning documents based on multiple interpretations of the query, instead of focusing exclusively on the most likely interpretation (Agrawal et al., 2009). Furthermore, the challenge of *exploratory search* can also be solved in a top-k setting by means of result list diversification, as is common practice in Web search (André et al., 2009) or recommender systems (Ziegler et al., 2005). We argue that for entity augmentation a similar argument can be made: It is advantageous to provide not only one solution, but allow the user to choose from a ranked list of alternative solution. In other words, in this chapter we aim at extending entity augmentation to *diversified top-k entity augmentation*.

Let us return to a new version of the running the example, shown in Figure 2.2, which illustrates the idea of top-k entity augmentation. In this figure, the query asks for k augmentations for the attribute "revenue", while the colored arrows map sources to potential alternative results. As an initial example, instead of returning only one result based on $S2, S3$ as discussed above and depicted with blue arrows in the example, one alternative would be a result based on $S1, S4$, colored red in the example. It has a worse average score, but has clearly marked year in both sources, which may be more useful for the user on manual inspection, because of the clearly marked year information in the context.

Another aspect are attribute variations, which, due to the exploratory nature of entity augmentation queries, may also be of interest to the user. An example would be a third result based on $S6$ and the second column of $S5$, depicted with green arrows in the example, which represents changes in revenue instead of absolute revenue. Yet another exploratory result could be comprised of just $S9$, which does not cover all entities, but might give the users analysis a new direction.

Note however, that we want to generate solutions that are real alternatives, such as the three examples above. Because of copying effects on the Web (Li et al., 2013a; Dalvi et al., 2012b), using only the most relevant sources for creating all k augmentation results however, would lead to many answers being created from structurally and semantically similar sources. Furthermore, because we fuse results from several sources, naïve solutions would use the same, most relevant, sources multiple times in various combinations, leading to *redundancy* in the result list. In the example, one such redundant result would be $S1, S7$, shown with dashed purple arrows in Figure 2.2. It reuses the highly relevant source $S1$ that was used in the $S1, S4$ result, but in combination with another source $S7$. This source however, differs only superficially from source $S4$, which makes the results $S1, S4$ and $S1, S7$ very similar. We want to avoid slight variations of one solution that, for example, only differ in a single data source, and also avoid creating multiple solutions from very similar datasets, as this would add little information to the top-k result. A meaningful top-k list needs to consider a *diverse* set of sources, exploring the space of available data sources while still creating correct and consistent answers. This has been recognized a long time ago in information retrieval (Carbonell and Goldstein, 1998), but has recently been explored for structured queries (Demidova et al., 2010; Hasan et al., 2012),

and even for Web table search (Tam et al., 2015), which is highly related to entity augmentation. We will define the notion of result diversity for our specific problem, as well as our means to achieve it, in the following sections.

To conclude this section, we want to summarize the requirements we derived for our novel entity augmentations method. For this chapter, we have two goals for our novel entity augmentation method: We aim at providing a *diversified top-k* list of alternative results, which are composed of *consistent and minimal* individual results. Note that the two aspects, construction of minimal consistent single results, and providing meaningful alternative solutions, are complementary to each other. Based on these insights, throughout the rest of the chapter, we make the following contributions:

- We propose new entity augmentation algorithms that construct multiple minimal and consistent augmentations for a given entity set and attribute. For that purpose, we formalize the problem of top-k entity set augmentation by extending the classic weighted Set Cover problem to the *Top-k Consistent Set Cover* problem.
- We propose a greedy algorithm that picks consistent data sources to construct individual augmentations, while maximizing the diversity between the results to provide meaningful alternative solutions.
- We improve on this first algorithm with a genetic set covering approach that naturally models the creation of a diverse set of individually strong solutions.
- We implement the algorithms in a new Web table retrieval and matching system called REA, and evaluate the system using the DWTC[1], a corpus of 125 million Web tables which we extracted from a public Web crawl. We measure the effects of our proposed algorithms on precision, coverage and runtime, but also on the new dimensions consistency and diversity of the top-k query results, and show substantial improvements over baseline methods.

2.3 TOP-K CONSISTENT ENTITY AUGMENTATION

In this section we will describe our novel method of top-k consistent entity augmentation. Initially, we will formalize augmentation queries and the optimization objectives of our method in Section 2.3.1. We then introduce top-k consistent *set covering* as an abstract framework for solving our problem in Section 2.3.2, and introduce multiple specific augmentation algorithms based on this notion in Sections 2.3.3 to 2.3.5.

2.3.1 Entity Augmentation Queries

We will now formalize our notion of top-k Consistent Entity Augmentation, and introduce the optimization objectives that we aim at. First, consider a general entity augmentation query definition.

[1] http://wwwdb.inf.tu-dresden.de/misc/dwtc

Definition 2.1 (Entity Augmentation Query). *Let $E(a_1, ...{}_a n)$ denote a set of entities with attributes $a_1, ..., a_n$, and a_+ denote the augmentation attribute requested by the user. Then, an augmentation query is of the form:*

$$Q_{EA}(a_+, E(a_1, ..., a_n)) = E(a_1, ..., a_n, a_+)$$

In other words, the result of such a query is exactly the set of input entities with the augmented attribute added. To create this new set of entities, the EAS has to retrieve values for attribute a_+ for each entity $e \in E$, which we will denote v_e. These values will be retrieved from a corpus of data sources $\mathcal{D}$ managed by the EAS, from which it selects a relevant subset D for each query Q_{EA}.

Sources $d \in D$ can provide a_+ values for some subset of E, denoted $\mathrm{cov}(d)$, i.e., they cover E only partially in the general case. Individual values provided by a source d for an entity e are denoted $d[e]$ with $e \in \mathrm{cov}(d)$. Given a heterogeneous corpus of overlapping data sources, an augmentation system will potentially retrieve multiple values for an entity e, the set of which we will denote by $V_e = \bigcup_{d_i \in D} d_i[e]$. Finally, the EAS assigns each data source a relevance score $rel : D \to [0, 1]$ with respect to the query. To determine this relevance score, various measures can be combined. Examples include the similarity between the queried attribute name and the respective attribute's name in the data source, or the quality of the match between the queried instances and those found in the data source. In addition, global measures for the quality of the data source can be integrated. For example, in the case of Web tables as sources, the PageRank of the source page with respect to the query can be included. We will discuss the specific relevance features used in our implementation of Top-k consistent set covering in Section 2.4.

As we described in Section 2.1, most systems from literature assume that they can reconcile the set of values V_e into a single correct augmentation value v_e for each entity. Consider the visualization in Figure 2.3, in which the vertical axis shows the queried entities e_i, the boxes depict sources d_i, with their vertical extent representing $\mathrm{cov}(d_i)$, the set of entities they provide a value for. The horizontal axis plots the sources' relevance score $\mathrm{rel}(d_i)$, i.e., sources to the left have been scored higher than sources to the right. The by-entity fusion approach found in related work is visualized in Figure 2.3 for the first three entities: each one is considered separately by fusing all candidate values into one result.

Definition 2.2 (By-Entity Fusion). *Given the set of retrieved values $V_e = \bigcup_{d_i \in D} d_i[e]$ for each entity $e \in E$, and a fusion method $\mathcal{F} : 2^{\mathcal{V}} \to \mathcal{V}$ operating on the augmentation attribute's domain $\mathcal{V}$, the By-Entity Fusion approach processes an entity augmentation query as:*

$$Q_{EA}(a_+, E) = \{\mathcal{F}(V_e) \mid e \in E\} \tag{2.1}$$

An example for such a fusion method would be majority voting, or clustering values and then picking a representative from the highest ranked cluster as in (Yakout et al., 2012) and (Lehmberg et al., 2015).

As motivated in Section 2.2, our notion of an entity augmentation query differs in several aspects: First, instead of individual values, it picks subsets of sources that cover E, and second, it returns an ordered list of alternative solutions. In other words, its basic result is a list of top-k alternative *selections of sources*.

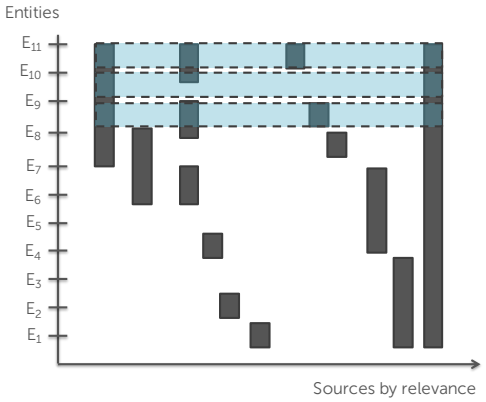


Figure 2.3: Augmentation using By-Entity Fusion

Definition 2.3 (Top-k Source Selections)**.** *Given a set D of relevant data sources, and a number k of augmentations to create, a top-k Source Selection is defined as:*

$$Q_{EA}(a_+, E, k) = [c_1,\ ...,\ c_k \mid c_i \subset D \wedge \mathrm{cov}(c_i) = E] \tag{2.2}$$

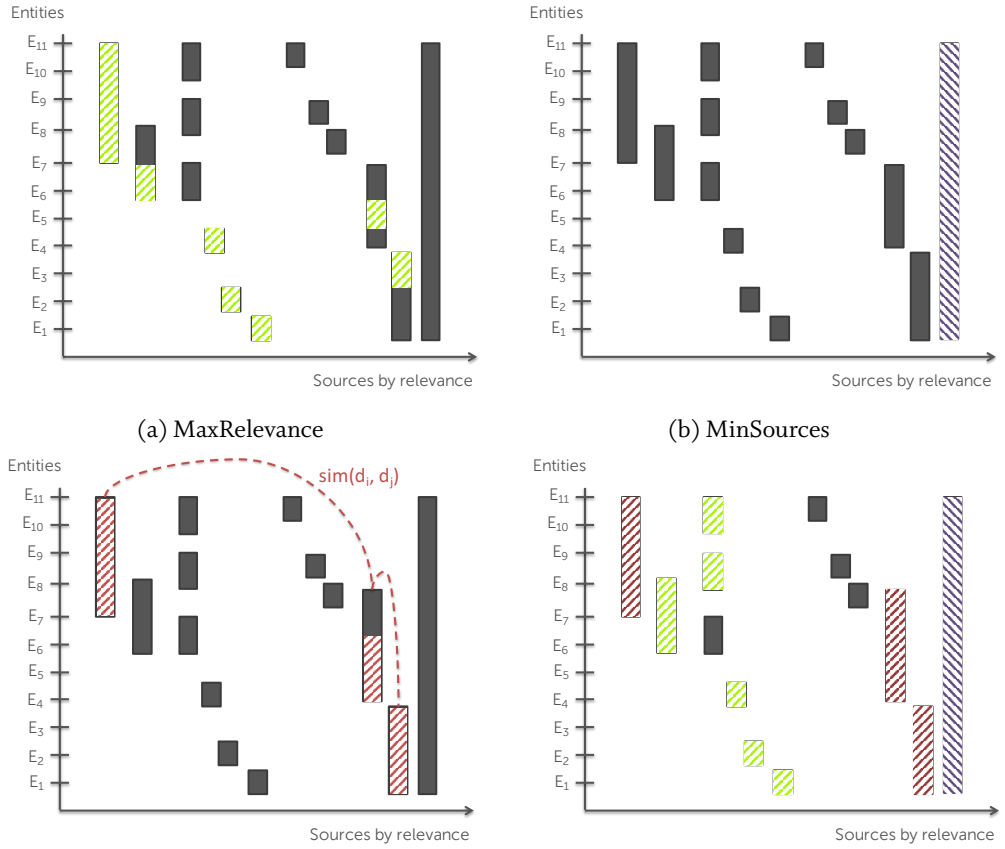
We call one such set c_i a *cover* or an *augmentation*, and the list of these augmentations the query result.

Definition 2.4 (Cover/Augmentation)**.** *A cover is an ordered subset of D that covers E, i.e., $c = [d_i, ..., d_x]$ with $\bigcup_{d \in c} \mathrm{cov}(d) = E$. If multiple data sources provide values for a distinct e, i.e, if $\exists_e(e \in \mathrm{cov}(d_i) \cap \mathrm{cov}(d_j))$, the augmented value for e is decided by the order of the sources in c, i.e., $v_e = d_i[e]$ with $i = min(\{i \mid e \in d_i \wedge d_i \in c\})$.*

As discussed in Section 2.2, the aim is to enable the user to choose the most promising augmentation from the ranked list of alternatives. This leads to the question of how to construct these subsets $c_i \subset D$, in order to create a valuable list of alternatives for the user. We will now introduce the individual dimensions of this problem, *relevance*, *minimality*, *consistency* and *diversity*, discussing exemplary baseline strategies that optimize for one of each dimension.

Relevance One naïve baseline strategy, which we call *MaxRelevance*, is depicted in Figure 2.4a, using the same form of visualization as Figure 2.3. Starting from the highest ranked data source, it picks all the values it provides for entities that do not have a value yet, then continues to the next most relevant source, according to the relevance function introduced above. While this strategy obviously maximizes the average relevance of the created augmentation, a large number of distinct sources might be picked. This makes it harder for the user to understand the query result and assess its quality, and also has a high chance of producing inconsistencies between sources. Generally speaking, for a set of covers C, the *Relevance* objective is:

$$\arg\max_{C} \sum_{c \in C} \sum_{d \in c} \mathrm{rel}(d) \qquad \textit{(Relevance Objective)}$$



(a) MaxRelevance

(b) MinSources

(c) MaxConsistency

(d) NoReuse

Figure 2.4: Strategies for creating augmentations

Minimality A naïve approach to solve this last problem would be to prioritize data sources with large coverage of the queried entities. This strategy, called *MinSources*, is illustrated in Figure 2.4b. As is illustrated in this particular example, while solutions created this way use a minimal number of distinct sources, the other objectives, such as relevance, can be arbitrarily bad. The general *Minimality* objective for a set of covers C is:

$$\arg\min_{C} \sum_{c \in C} |c| \qquad \text{(Minimality Objective)}$$

Consistency Next, consider the strategy *MaxConsistency*, in which sources are chosen based on a measure of source similarity, i.e., a function $\text{sim} : D \times D \rightarrow [0, 1]$, which is depicted using dashed arrows in Figure 2.4c. This function captures our notion of attribute variant consistency as discussed in Section 2.1. It is calculated from measures such as the similarity between the two data source's attribute names, their value sets, and by comparing the associated metadata such as dataset titles and context. We will introduce specific features used for our source similarity functions in Section 2.4. Utilizing such a function to guide source selection will increase the overall consistency of the created augmentations, but will create augmentations that are

not necessarily minimal nor highly relevant.

$$\arg\max_{C} \sum_{c \in C} \sum_{d_i, d_j \in c} \text{sim}(d_i, d_j) \qquad \textit{(Consistency Objective)}$$

Diversity In addition, we will have to devise a method that is able to create multiple meaningful alternative solutions. An naïve solution would be to create one cover c from sources D, and then iteratively set $D' = D \setminus c$ and create the next cover from D'. This approach, called *NoReuse*, is illustrated in Figure 2.4d. It has two problems: Firstly, each data source can be used in only one alternative, even though several combinations of good sources might be possible. Secondly, just prohibiting reuse of specific datasets does not necessarily lead to diversified solutions, as there may be data sources so that $\exists_{d_i,d_j} | d_i \neq d_j \wedge \text{sim}(d_i, d_j) \approx 1.0$. This occurs, for example, due to frequent copying-effects on the Web (Li et al., 2013a; Dalvi et al., 2012b). Effectively, this strategy would could create multiple very similar covers, even though no dataset is ever used more than once.

In general, the objective function for the *Diversity* of a set of covers C is:

$$\arg\min_{C} \sum_{c_i, c_j \in C} \text{sim}_{\mathcal{A}}(c_i, c_j) \qquad \textit{(Diversity Objective)}$$

In other words, we aim at minimizing the pairwise similarity of covers in the query result. Since here the similarity function $\text{sim} : D \times D \rightarrow [0, 1]$ is used to compare covers, and not datasets, we involve an aggregation function $\mathcal{A}$, such as *average* or *max*. This lifts the similarity function to the domain of covers $\text{sim}_{\mathcal{A}} : C \times C \rightarrow [0, 1]$.

We considers these four dimensions to be the decisive factors for a useful top-k entity augmentation result. What we therefore need, is a strategy that creates complete covers, while jointly optimizing for relevance, minimality, consistency and diversity.

Definition 2.5 (Top-k Consistent Entity Augmentation)**.** *A top-k Consistent Entity Augmentation query produces a top-k Source Selection (Definition 2.3) that is optimized with respect to the relevance, minimality, consistency and diversity objectives.*

In the next section, we will introduce our algorithmic approach to processing top-k Consistent Entity Augmentation queries.

2.3.2 Ranked Consistent Set Covering

We propose a new approach for constructing entity set augmentations by modeling it as an extended form of the *Weighted Set Cover Problem*, one of Karp's original 21 NP complete problems (Karp, 1972).

Definition 2.6 (Weighted Set Cover)**.** *Given a universe of elements U and a family of subsets of this universe S, each associated with a weight w_i, the Weighted Set Cover problem is to find a subset $s \subset S$ with $\bigcup_s = U$, such that $\sum_{i \in s} w_i$ is minimized.*

Intuitively speaking, the aim is to *cover* all elements in U using sets from S with minimal cost. In our problem domain, the algorithm input consists of a set of entities E that are to be augmented, corresponding to U in the original problem, and a set of data sources $D = \{d_1, ...d_n\}$, as retrieved and matched by the underlying entity augmentation system, corresponding to S. The relevance score assigned to each datasource by $\mathrm{rel}(d)$ (Section 2.3.1) is used in place of the weights w.

So far, we could trivially map our problem to the well known Set Cover problem. Specifically, the *Relevance* and *Minimality* objectives defined in Section 2.3.1 correspond closely to the objective $\sum_{i \in s} w_i$ in the set cover problem. Still, there are some crucial differences: In contrast to the original problem, where only a *single* minimal cover is required, the output we aim for is a ranked list of covers, denoted $C = [c_1, ..., c_n]$. Furthermore, as illustrated in Section 2.3.1, the entity augmentation use case does not only require small covers with high individual relevance, but *consistent* covers, as defined in the *consistency* objective. And lastly, we also introduced the *diversity* objective, i.e., the covers should not consist of the same or similar datasets throughout the top-k list, but be complementary alternatives.

We will now incrementally develop our proposed algorithms for top-k consistent set covering. We start from the well known greedy algorithm for the Weighted Set Cover problem, which, given a universe U, a set of sets S with weights w, and a set of yet uncovered elements F, iteratively picks the set:

$$\arg\min_{S_i \in S} \frac{w_i}{|S_i \cap F|} \qquad \textit{(Greedy Set Cover Algorithm Step)}$$

The algorithm chooses sets S_i until $F = \emptyset$, at which point a complete cover has been formed. Although the greedy algorithm does not create optimal covers, it is still the most frequently employed algorithm for the set covering problem. It fact, it has been shown that the algorithm, achieving an approximation ratio of $H(s') = \sum_{k=1}^{n} \frac{1}{k}$ is essentially the best possible polynomial-time approximation algorithm for the set cover problem (see Section 2.6.4).

Coverage and Relevance We therefore also initially base our algorithm on the greedy set covering algorithm. With an initially empty cover c and a free entity set $F = E$, we can use the original greedy Set Cover algorithm to produce an ordered subset of D, by picking in each iteration the dataset d that maximizes:

$$\arg\max_{d_i \in D} \mathrm{rel}(d_i) \cdot |\operatorname{cov}(d_i) \cap F| \tag{2.3}$$

until $F = \emptyset$. Note that we maximize scores instead of minimizing weights as this is more intuitive for the problem domain.

An augmentation constructed in this way would roughly correspond to a middle ground strategy between the *MaxRelevance* and *MinSources* strategies discussed in Section 2.3.1. This implies, however, that it can potentially create augmentations from very heterogeneous data sources.

Cover Consistency To counteract this effect, we explicitly model consistency between the datasets that make up a cover. We utilize the similarity function between datasets $\mathrm{sim} : D \times$

$D \rightarrow [0, 1]$, as defined in Section 2.3.1, which models the consistency between data sources. We give a specific instance of this function for Web tables in Section 2.4. Given an initially empty cover c and an aggregation function $\mathcal{A}$ such as *average* or *max*, we can greedily generate covers using consistent datasets by picking in each iteration the dataset d that maximizes:

$$\arg\max_{d \in D} \ \mathrm{rel}(d) \cdot |\operatorname{cov}(d) \cap F| \cdot \mathrm{sim}_{\mathcal{A}}(d, c) \tag{2.4}$$

This means we encourage picks of data sources that are similar to data sources that were already picked for the current cover. We assume as a special case that $\mathrm{sim}_{\mathcal{A}}(d_i, \emptyset) = 1$, which implies that the first data source chosen will be the same as in regular set covering. Subsequent choices on the other hand will be influenced by already selected sources. This also implies that datasets with a low relevance or coverage, that are not picked initially, may still be chosen in a later iteration, if they fit well with those chosen so far. Since we require $|\operatorname{cov}(d) \cap F|$ to be greater than zero, the algorithm will still make progress with every step, as only datasets that provide at least one new value can be selected.

Using objective function 2.4, the algorithm picks datasets to create covers that are not only highly relevant to the query, but also fit well together according to $\mathrm{sim} : D \times D$. However, using only this objective function, there is still no intuitive way of creating useful top-k augmentations. The naïve approach re-running the same algorithm with $D \setminus c$ as the set of candidate data sources would not lead to useful alternative solutions, as discussed in Section 2.3.1.

Top-k Results and Diversity Let C denote the set of previously created covers, with $|C| \geq 1$. This set could be initialized with a single cover created, for example, using the greedy algorithm and objective function 2.4. Our core idea is to perform consecutive runs of the greedy algorithm using the same input datasets, with each run placing greater emphasis on datasets that are dissimilar to datasets picked in previous iterations, i.e., dissimilar to datasets in $\bigcup_C$. Implementing this idea naïvely however, for example by dividing function 2.4 by $\sum_{d_i \in \bigcup_C} \mathrm{sim}(d, d_i)$ does not yield the expected results. While the second iteration might then choose datasets from a different part of the similarity space than the first iteration, the term becomes increasingly meaningless with more iterations as $\bigcup_C$ grows. This is because newly picked datasets are compared to a larger and larger subset of the candidate set D, leading to an increasingly uniform values for $\sum_{d_i \in \bigcup_C} \mathrm{sim}(d, d_i)$.

Instead, we introduce a more complex dissimilarity metric based on individual entities in E and the datasets that were used to cover them in previous iterations. We define a function $coveredBy(e, C)$ which yield the datasets that were used to augment the entity e in covers C created in previous iterations.

$$coveredBy(e, C) = \{d \mid \exists_c \in C : d \in c \wedge e \in \operatorname{cov}(d)\} \tag{2.5}$$

We can then define out final scoring function as

$$\arg\max_{d \in D} \frac{\mathrm{rel}(d) \cdot |\operatorname{cov}(d) \cap F| \cdot \mathrm{sim}_{\mathcal{A}}(d, c)}{\mathrm{redundancy}(d, F, C)} \tag{2.6}$$

where

$$redundancy(d, F, C) = \sum_{e \in F \cap \mathrm{cov}(d)} \mathrm{sim}_{\mathcal{A}}(d, \mathrm{coveredBy}(e, C)) \tag{2.7}$$

In other words, we penalize picks that would cover entities with data sources that are similar to datasets that were already used to cover these entities in previous iterations. By penalizing similarity to previous covers, we avoid using the same similar datasets repeatedly for all covers in the top-k list, but we also do not strictly disallow the re-use of data sources in new combinations. Objective function 2.6 forms the core of our proposed entity augmentation algorithms, which we will introduce in the next sections.

Algorithm 1 Top-k consistent set covering: Greedy

function GREEDY-TOPK-COVERS(k, E, D)
  $C \leftarrow \emptyset$
  $U \leftarrow \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}_{|E| \times |D|}$ ▷ Usage matrix
  while $|C| < k$ **do**
    $c \leftarrow \text{COVER}(E, D, U)$
    for all $(e \rightarrow d) \in c$ **do** ▷ Update Usage Matrix
      $U[e, d] \leftarrow U[e, d] + 1$
    if $c \notin C$ **then** ▷ Remove duplicates
      $C \leftarrow c$
  return C

function COVER(E, D, U)
  $c \leftarrow \emptyset$
  $F \leftarrow E$ ▷ Free set, uncovered entities
  while $|F| > 0$ **do**
    $d \leftarrow \arg\max_{d \in D} \frac{\text{rel}(d) \cdot |\text{cov}(d) \cap F| \cdot \text{sim}_{\mathcal{A}}(d,c)}{\text{REDUNDANCY}(d,D,F,U)}$
    for all $e \in F \cap \text{cov}(d)$ **do**
      $F \leftarrow F \setminus e$ ▷ Update free set
      $c \leftarrow c \cup (e \rightarrow d)$ ▷ Update cover
  return c

function REDUNDANCY(d, D, F, U)
  $r, norm = 0, 0$
  for all $e \in F \cap \text{cov}(d)$ **do** ▷ Coverable by d
    $u \leftarrow U[e]$ ▷ Sources used to cover e
    $r \leftarrow r + \sum_{i=0}^{|u|} u[i] * \text{sim}(d, D[i])$
    $norm \leftarrow norm + \sum u$
  return $\frac{r}{norm}$

2.3.3 Basic Greedy Algorithm

With the scoring function in place, we can construct a greedy consistent set covering, shown in Algorithm 1, that produces consistent individual augmentations, as well as diversified solutions when run with $k > 1$. In Algorithm 1, the function *Greedy-TopK-Covers* produces k covers by calling the function *Cover* k times, while keeping a $|E| \times |D|$ matrix called U as state between the calls. While the *Cover* function performs the basic greedy set cover algorithm with the objective function defined above, the main function updates the matrix U after each iteration by increasing the entry for each entity/dataset combination that is part of the produced cover. The function *coveredBy* used in the *redundancy* term of objective function 2.6 is realized in the algorithm by summing over the matrix row $U[e]$, which records the datasets used to cover e in previous covers. Note that the main function also discards duplicate solutions, which may occur if the influence of the *redundancy* function is not strong enough to steer the search away from an already existing solution. Still, the matrix U is updated even if a solution is rediscovered, so that further choices of the same data sources become more and more penalized, guiding the search into a different part of the solution space.

The greedy approach described above, while being easy to implement and fast to execute, will not necessarily construct the best possible list of solutions, as our evaluation in Section 2.5 will show. This is mainly due to exploring only a small part of the search space, i.e., considering only k different covers. Therefore, we introduce two further algorithms as extensions of the basic framework in the next two sections.

2.3.4 Extension: Greedy*

The first extension of the basic framework is based on the observation that the first k solutions produced by Algorithm 1 may not necessarily be the best solutions. After the first solution has been produced the search is mainly guided by using different datasets for each solution, and thus new combinations of previously used data sets are often not considered in the basic greedy algorithm. One simple extension is called *Greedy**-algorithm, which uses the basic greedy algorithm to create more covers than requested, and then introduces a second phase to the query processing called *Select*, in which the k best solutions are selected from a pool of $s \times k$ possible solutions, as shown in Algorithm 2, with s being the scale factor. This raises the question of how to select the top-k solutions. Following (Drosou and Pitoura, 2010) we evaluated two ways to pick diversified top-k solutions from a pool of possible solution: one again based on a *greedy* approach and one based on *replacement*. The greedy approach, shown in Algorithm 1, iteratively picks the best cover by relevance, consistency and dissimilarity to the already picked covers. In contrast to Algorithm 1, where we use objective functions to estimate the gain of picking a single dataset, in Algorithm 1 we use functions that estimate the gain of selecting a complete cover for the overall result list. These *Cover quality* functions are derived

Algorithm 2 Top-k consistent set covering: Greedy*

function GREEDY*-TOPK-COVERS(k, s, E, D)
  $C \leftarrow$ GREEDY-TOPK-COVERS($k * s, E, D$) ▷ As in Algorithm 1
  $C \leftarrow$ SELECT(k, C)
  return C

function SELECT-GREEDY(k, C)
  $C' \leftarrow []$
  while $|C'| < k$ **do**
    $C' \leftarrow C' \cup \arg\max_{c \in C} \text{rel}(c) \cdot \text{cons}(c) \cdot \text{div}(C' \cup c)$
  return C'

from the four optimization objectives introduced in Section 2.4.

$$\text{rel}(c) = \frac{1}{\sum_{d \in c} |\operatorname{cov}(d, c)|} \cdot \sum_{d \in c} |\operatorname{cov}(d, c)| * \text{rel}(d) \tag{2.8}$$

$$\text{minim}(c) = \begin{cases} 1.0 & |c| = 1 \\ 1.0 - \frac{|c|}{\sum_{d \in c} |\operatorname{cov}(d,c)|} & \text{else} \end{cases} \tag{2.9}$$

$$\text{cons}(c) = \frac{1}{\sum_{d \in c} |\operatorname{cov}(d, c)|} \cdot \sum_{d \in c} |\operatorname{cov}(d, c)| * \text{sim}_{\mathcal{A}}(d, c) \tag{2.10}$$

$$\text{div}(C) = \frac{\sum_{c_i, c_j \in C} \text{dist}(c_i, c_j)}{\binom{|C|}{2}} \tag{2.11}$$

(*Cover quality functions*)

With a slight variation in notation, we here use $\operatorname{cov}(d, c)$ to indicate not the whole coverage of d, but only the set of entities covered by d in the given cover c. In other words, $\operatorname{cov}(d, c)$ measures the share of c supplied by d, i.e., its importance. The div quality function requires a measure of distance between covers, which we define similarly to the *redundancy* term used for the covering algorithm (Equation 2.7). For all entities $e \in E$, it takes the similarity between the two datasets used to cover e in the two covers it compares, averages and then inverts them to get a distance.

$$dist(c_1, c_2) = 1.0 - \frac{\sum_{e \in E} \text{sim}(\text{coveredBy}(e, c_1), \text{coveredBy}(e, c_2))}{|E|}$$

We also evaluated solutions based on replacement, in which the first k covers as produced by the greedy approach are used as the initial top-k list, which is then iteratively refined. Refinement is done by removing the cover least fit according to the scoring functions and replacing it with a new cover from the pool of possible answers until a stop criterion is reached, e.g., when the diversity of the top-k list has fallen under a certain threshold.

In comparison to the *Greedy* algorithm, the *Greedy** approach should find better solutions as it searches a larger portion of the search space, at the cost of a runtime that increases with

the scale factor s, plus some overhead for the selection phase. However, the way it explores the solution space is relatively naïve. In the next section, we will therefore introduce another algorithm that is based on generating a large solution set, and then picking an optimal diversified subset.

2.3.5 Extension: Genetic Algorithm

There is a large body of literature dedicated to the set covering problem, in which various optimization methods apart from the greedy approach are studied (see Section 2.6). The genetic approach seems especially viable for our specific versions of the problem, as it intrinsically generates a pool of solutions from which k can be picked, and both consistency and diversity of the results can be modeled intuitively. Specifically, consistency can be modeled as part of the fitness function, and diversity can be introduced through a suitable population replacement strategy. In general, to apply the genetic framework to a problem, we need to define

- a representation of individuals and their genomes,
- the fitness function,
- cross over and a mutation functions,
- a method for population initialization,
- the strategy for parent selection,
- and finally a mechanism replacement of individuals in each generation.

We will discuss all of these factors in the following. While our approach is inspired partly by (Beasley and Chu, 1996), our problem domain makes different choices for almost all of these decision necessary. Obviously, we can represent an individual's genome as the set of data sources it is comprised of, i.e., as a bit vector where the n-th bit is set if the n-th candidate source was used in the cover. We can then use the objective function 2.6 as the fitness function for individuals. The population is initialized by creating covers with the Greedy algorithm until all candidate sources have been used in at least one cover. The Greedy algorithm is modified in this use case to strongly favor unused data sources in later iterations to quickly produce such an initial population. This can be trivially achieved by assigning an increasing weight to the *redundancy* term over the iterations of the Greedy Algorithm 1. This initialization ensures that all the possible genetic material, i.e., all possible data sources, are represented in the starting population.

The most interesting step is the crossover function, as combining two sets of data sources does not necessarily again yield a valid cover. If the wrong genes are passed to the descendant solution, the solution may not be feasible, i.e., there may be uncovered entities, or if too many genes are passed, the cover may not be minimal. In (Beasley and Chu, 1996), genes shared by both parents are passed on definitively, and those only present in one parent are passed with a probability corresponding to their weight, which leads to potentially infeasible or non-minimal solutions as mentioned above. To correct this, another step of pruning redundant genes and adding random genes from the pool that cover missing entities was proposed.

Algorithm 3 Top-k consistent set covering: Genetic

function GENETIC-TOPK-COVERS(k, s, E, D)

$U \leftarrow \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}_{|E| \times |D|}$ ▷ Usage matrix

$P \leftarrow$ INIT-GREEDYTOPKCOVERS($k * s, E, D, U$)

for $i \in 0..k * s$ **do**

$c_1, c_2 \leftarrow$ pickRandom(P) ▷ Select Parents

▷ Tournament vs. most similar solution

$w_1, l_1 \leftarrow$ tournament$(c_1, \arg\max_{x \in P \setminus c_2} \text{sim}(c_1, x))$

$w_2, l_2 \leftarrow$ tournament$(c_2, \arg\max_{x \in P \setminus c_1} \text{sim}(c_2, x))$

$c_+ \leftarrow$ COVER($E, c_1 \cup c_2, U$) ▷ As in Alg. 1

MUTATE(c_+) ▷ Randomized Flipping of Genes

FILLANDPRUNE(c_+) ▷ As in (Beasley and Chu, 1996)

$removed \leftarrow \text{min}_{score}(l_1, l_2)$

$P \leftarrow \big(P \setminus removed\big) \cup c_+$

$U \leftarrow$ UPDATEUSAGE($U, c_+, removed$) ▷ As Alg. 1

$C \leftarrow$ SELECT(k, P) ▷ As in Alg. 2

return C

However, this solution does not take consistency of the generated descendant into account. Instead, we again use the basic Greedy approach as a building block, but use the shared genes as starting point, and take the union of genes as the set of candidates, instead of the whole set D. We use the same mutation strategy, i.e., randomly flipping bits in the gene, and subsequently use the prune and fill approach from (Beasley and Chu, 1996) to remove datasets that may have become redundant, and add datasets to fill holes created by mutation. The *fill* operation can be realized using the greedy algorithm with the incomplete cover as the starting points, i.e., by computing $Cover(E, D, U)$ after mutation if necessary. The *prune* operation works by alternating between computing the redundant datasets $rd = \{d \mid cov(c \setminus d) = E\}$, and then removing $\arg\min_{d \in rd} \text{sim}_{\mathcal{A}}(d, c)$ until there is not redundancy in the cover left.

Diversity is achieved through several mechanisms: in the crossover step, since the Greedy approach as a building block uses a global usage matrix U throughout the run of the genetic algorithm, in the mutation step that introduces randomness into the population, and finally in the parent selection and replacement strategy. Here, we select two parents randomly to create a descendant, but before crossover, we pair them with their most similar solution in the population use the better one by score, while the least fit solution is replaced in the next generation. Algorithm 3 gives an overview of our genetic consistent top-k set covering approach and all the mentioned mechanics.

Having introduced our abstract algorithms for top-k diversified and consistent set covering

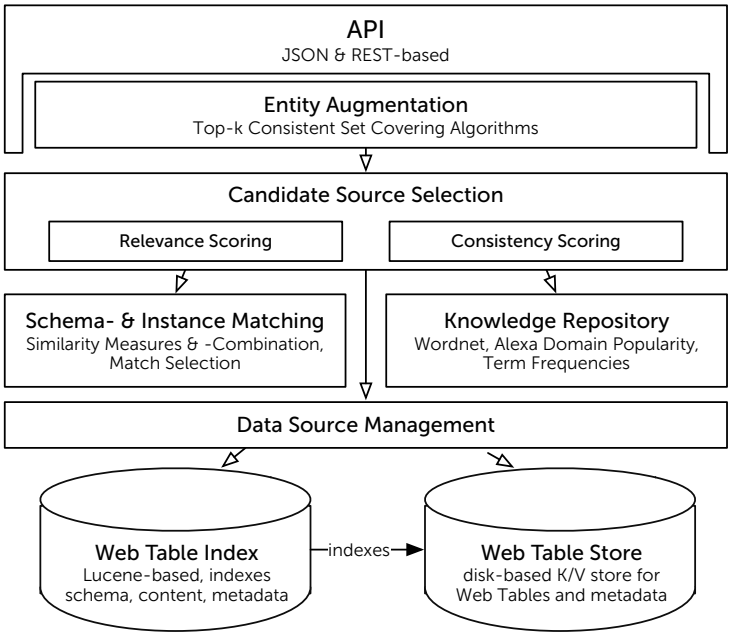


Figure 2.5: REA System Architecture

in this section, the next step is to instantiate them for the problem of Web table-based entity augmentation by introducing the Web table-specific relevance and similarity functions $\text{rel} : D \rightarrow [0,1]$ and $\text{sim} : D \times D \rightarrow [0,1]$. We will therefore now describe our novel Web table-based entity augmentation system REA.

2.4 THE REA SYSTEM

In this section, we discuss our novel entity augmentation system REA[2]. It implements the abstract method of consistent top-k set covering for the specific case of Web table-based entity augmentation. This implementation requires the following components:

- **Candidate Source Selection**: a method for generating the set D of candidate Web Tables for use in the set covering algorithms, given a specific augmentation query Q_{EA}.
- **Source Relevance Estimation**: an instantiation of $\text{rel} : D \rightarrow [0,1]$ for Web tables, i.e., a function that scores a Web table with respect to Q_{EA}.
- **Source Consistency Estimation**: an instantiation of $\text{sim} : D \times D \rightarrow [0,1]$ for Web tables, i.e., a function calculates the consistency of two Web tables with respect to Q_{EA}.

In Figure 2.5 we give an architectural overview of our proposed system fulfilling these requirements. The system consists of a series of layers providing increasingly higher-level services.

[2]http://github.com/JulianEberius/REA

The bottom layer, the *Data Source Management* system, provides storage and indexing facilities for Web tables, enabling the higher layers to retrieve raw Web tables based on keyword matches in data, schema or other metadata. The next layer contains a *Schema- and Instance Matching System* for generating mappings between the query attributes and entities and those found in the Web tables. We will detail these two systems operation in Section 2.4.1. It also contains a *Knowledge Repository* for managing and accessing external knowledge sources that are employed in the retrieval and matching process. Currently it offers synonym lookups via Wordnet[3], Web domain popularity and category information via Alexa Web Services [4] and term frequencies extracted from the DWTC[5] (Eberius et al., 2015a), which we use for *tf-idf* scoring throughout the matching process.

All these lower level services are utilized in the *Candidate Source Selection* component, which orchestrates them to create a candidate dataset D given an augmentation query Q_{EA}. For this task, it implements the rel and sim functions, which we will detail in Sections 2.4.2 and 2.4.3. Using the candidate set D including relevance and consistency scores, the *Entity Augmentation* system then creates the top-k augmentation result as described in Section 2.3.2. Finally, REA includes a JSON-based REST API, which enables other systems to easily integrate with it and pose entity augmentation queries.

2.4.1 Web tables Retrieval and Matching

REA uses uses both relational Web tables, also called "entity-value" tables, and entity-tables, also called "attribute-value" tables, which focus on one entity only. This allows consistent results for domains where relational tables are available, but increases recall in domains where no or only few such tables are ever published.

The retrieval of candidate Web tables works as follows: we build several indices on the corpus of available Web tables, both on attribute names (*AI*), content cells (*CI*), as well as metadata, e.g., page title and most frequent terms in the context (*MI*). For attributes, we optimistically assume that, after basic cleaning, the attribute names are located in the first row for relational tables, or the first column for entity tables. There are more precise ways of analyzing tables to identify attributes and concepts (see Section 2.6), but they are not in focus of this work. We then use a simple but effective approach to identify relational- and entity candidate tables in the corpus: we build not only one attribute query, but one on the first row (*ARI*) and one on the first column (*ACI*). For each entity augmentation query $Q(a, E)$ we then run two queries against the corpus: one query $ARI(a) \wedge CI(E)$ to identify relational candidate tables for the input set of entities, and one $ACI(a) \wedge \forall_{e \in E} MI(e)$ to identify entity candidate tables for each entity.

The respective candidate table sets are then processed in two separate pipelines, which we mostly sketch as many parts correspond to state-of-the-art techniques in schema and instance matching. String distance functions and synonym dictionaries are used to determine where the entities and attributes are located in the table, which enables extraction of values for augmentation. Specifically, we use a measure which we wall *WeightedTokenLevenshtein* (WTL) which

[3] http://wordnet.princeton.edu/

[4] http://aws.amazon.com/awis/

[5] http://wwwdb.inf.tu-dresden.de/misc/dwtc

is similar in style to the one used in (Chaudhuri et al., 2003). The *WTL* function decomposes strings into token sets, weights the tokens according to their inverse document frequency in our table corpus, and then compares the individual tokens using a regular thresholded Levenshtein distance. Matching token pairs contribute their IDF weight to the final matching score for the two original strings.

$$
\begin{aligned}
WTL(a,b) &= \frac{\sum_{(t_1,t_2)}^{tok(a)\times tok(b)} tokSim(t_1,t_2)}{\sum_{t}^{tok(a)\cup tok(b)} idf(t)} \\
tokSim(t_1,t_2) &= \begin{cases} idf(t_1)+idf(t_2) & \text{if } lev(t_1,t_2) \geq th_{sim} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{2.12}
$$

We found that *WTL* outperforms traditional string distances such as Edit distance or nGram distance, especially for multi-word entity names such as "agricultural bank of china" and "industrial bank of china" . This is due to the *IDF* weighting, which places emphasizes on tokens that strongly differentiate strings, and ignores generic terms.

In the case of entity tables, the entity itself is often not present in the table at all, so we attempt to locate it in the URL or page title to verify that the table is about the queried entity. While simplistic, this approach has been shown to be reasonably effective in related work (Yin et al., 2011), and proved sufficient for our needs of retrieving data sources for our covering algorithms. Additionally, the matching step also supplies parts of the scoring information through the distance functions' confidence values, and also through the location of the matches. We detail scoring of candidates in the next section (2.4.2).

The matching step also discards candidates if the attributes or entities do not match closely enough or if no values could be extracted. For example, an index lookup for "Bank of China" might return datasets about the "Industrial and Commercial Bank of China" or the "Agricultural Bank of China", or a column named "Revenue" might be present, but the corresponding content cell might be empty, or not contain a number.

2.4.2 Relevance Scoring

As described in Section 2.3.2, ranked consistent set covering approaches require all data sources to be relevance scored. After the candidates have been processed and filtered as described in the last section, we score them according to the following criteria:

- **Quality of the schema-/instance match:** The confidences returned by the string distance functions, both for matches between the queries' attributes and entities, as described in the previous section.
- **Quality of the metadata match:** We use set distances between top terms in the query and the table's metadata, such as top terms extracted from page context, from the title and URL.
- **Quality of the data source:** A perfect match to an untrustworthy source may be less desirable than an almost perfect one to an established source. There are many techniques to measure trust in a Web page, such as the well-known PageRank algorithm, but these are out of scope for the REA system. We approximate source quality using the popularity

scores returned by the Alexa Web information service available through Amazon Web Services[6].

We use a weighted sum of these scores as the final relevance score of each Web table.

2.4.3 Web table Similarity

The second requirement for applying the algorithms discussed in Section 2.3.2 to Web tables is a similarity function that can be used to calculate consistency of covers, and by extension, diversity of a number of covers.

- **Attribute Similarity:** As mentioned in Section 2.4.1, an entity augmentation query for an attribute "revenue" might return sources with many variants of the attribute, such as "revenue 2010" or "revenue change". Since we are aiming for covers that are consistent internally but diverse between them it makes sense to compare their extracted attribute names, to group similar attribute names into one cover. Specifically, we remove the queried attribute name, spelling variations, expanded synonyms, e.g. "sales" for "revenues", and then compute string distances between leftover tokens of the candidate source to identify similar attribute variants.
- **Value Similarity:** While different data sources may be similar in the attribute name they specify, they may still mean different aspects of the attribute, e.g. change in the attribute instead of absolute values, or use different units. We therefore also compute similarities between the value sets extracted from the data source. Since we aim for supporting analytical queries, we currently only query numeric attributes and can therefore use numeric set similarity measures. In our current implementation we compare the order of magnitude of the value set's averages as a similarity measure. When comparing string valued attribute sets, measures such as average string lengths, length standard deviations, number of words or similar could be used.
- **Metadata Similarity:** We use the Jaccard distance between the term sets extracted for two candidates as another similarity measure. Our extraction pipeline extracts most frequent terms from the table's page context, but we also compare terms extracted from page title and URL separately.
- **Tag Similarity:** Similarly to (Zhang and Chakrabarti, 2013), we use extractors for frequent attribute variations, such as years, units of measurement or currencies. These extractors attach tags to candidate Web tables, and we use Jaccard distances between the extracted tags of both candidates as a further measure of similarity.
- **Domain Similarity:** The intuition here is that tables from similar Web domains are more likely to be consistent with each other than those from different domains. Since Web domain similarity is not in the focus of the thesis, we employ a simple model in which two sources have a domain similarity of 1.0 if they stem from the exact same domain. If not, we compare domain keywords and categories as returned by the same Alexa Web information service we used in Section 2.4.2.

Again, we employ a weighted sum of these factors for the final similarity between two Web tables, which is then used as the similarity function $\mathrm{sim}(d_1, d_2)$ required by the top-k consistent

[6] http://aws.amazon.com/awis/

set covering algorithms introduced in Section 2.3.2.

Having introduced the REA Web table retrieval and matching system, we can now evaluate our top-k consistent set covering algorithms in the next section.

2.5 EVALUATION

We conducted an extensive experimental evaluation of the proposed techniques to study the following questions:

- Do the proposed algorithms reduce the weaknesses of the baseline algorithms regarding consistency and minimality?
- Are the top-k results diversified, while maintaining individually correct and consistent augmentations?
- How do the basic measures such as coverage and precision change over the top-k results?
- How do the proposed algorithms compare with respect to runtime performance?

2.5.1 Experimental Setup

This section introduces our experimental setup, specifically the dataset and queries we use, the baseline algorithms, as well as the indicators we measure.

Dataset All queries are run against the full Dresden Web Table Corpus, a corpus of 125M Web tables we extracted from a public Web crawl, as described in (Eberius et al., 2015a) and published on `http://wwwdb.inf.tu-dresden.de/misc/dwtc`.

Queries To enable comparable results, we used similar domains and queried attributes as in related work (Yakout et al., 2012; Zhang and Chakrabarti, 2013): companies from the Forbes 2000 list[7] with the attributes *revenue*, *employees* and *founded*, countries with the attributes *population*, *population growth* and *area*, and finally largest cities based on the Mondial database[8], with the attribute *population*. When not mentioned otherwise, the default query set used for the experiments has 4 distinct queries for each domain-attribute pair, with 20 entities ($|E| = 20$) each, and $k = 10$. For the company and city domain we also differentiate between top and random entities, i.e., four queries from the top 100 of the Forbes list, and four with randomly picked entities. In addition to the standard queries, we also vary $|E|$ and k for the performance experiments.

Precision Assessment To establish whether an augmentation value returned by any algorithm is correct, one possible approach is to manually compile a Gold Standard, i.e., the set of correct answers for the test queries. However, even seemingly simple facts such as population

[7] http://www.forbes.com/global2000/
[8] http://www.dbis.informatik.uni-goettingen.de/Mondial/

counts can refer to different years, or may use different ways of measurement, e.g., different ways of defining city borders for the city domain. Furthermore, there may be slight variations of the queried attribute, such as "under 25 years" for the attribute "population", which may still be of interest for an exploratory query. As discussed in Section 2.2, we aim at embracing this variety and return top-k results, instead of a single-truth value.

We therefore evaluate precision in terms of *relevance* of the returned results to the query, instead of a binary correctness decision. To establish relevance, we asked a group of human judges to classify datasets. Specifically, for each entity in each result set we asked the judges to evaluate whether the data source used to cover the entity was relevant to the query, and whether the entity was matched correctly. While we collected judgments to cover all the domains and attributes introduced above, to keep the amount of manual work manageable, the number of augmentations created was smaller than in the automatic evaluation (only up to $k = 5$).

Indicators Due to the nature of the cover problem, it is not enough to measure relevance and coverage of the query results to asses the quality of the proposed algorithms. We further need to determine measures such as consistency and minimality of the individual augmentations, as well as the diversity of the result list. We will now discuss the indicators measured in our evaluation in detail.

- **Coverage:** We measure the percentage of the queried entities that are augmented with a value. However, we do not primarily aim at measuring the quality of the Web tables corpus or the schema- and instance matching system used. Instead, our evaluation focuses on the quality of the covering algorithms, which is orthogonal to these issues. For this reason, if not stated otherwise, we only measure the indicators for entities for which our combination of table corpus and matching system returns at least one (potentially irrelevant) result value. We do, however, also provide absolute coverage percentages for reference and comparison with other entity augmentation systems in Section 2.5.3.
- **Precision:** We measure the percentage of entities for which a relevant value was retrieved, as described above in the discussion of Gold Standards. This considers augmented values for each entity individually, i.e., an augmentation with relevance 1.0 indicates that each entity was augmented with a value that was judged relevant with respect to the query keyword. This does not necessarily imply that the augmentation is consistent.
- **Relevance Score:** For this indicator, we measure the average relevance score each individual dataset was assigned by the rel function. This score, however, is not to be confused with the actual *relevance* as measured by human judges, defined above. As explained in detail in Section 2.4.2, this score measures the confidence of the schema and instance match between query and data source, as well as the quality of the source. Thus, a higher value is of course better, as higher scored datasets are more likely to contain relevant results. It is calculated exactly as the relevance cover quality score defined in Equation 2.8.
- **Minimality:** We measure the number of datasets used in a cover in relation to the number of entities it covers. This corresponds to the minimality cover quality score as defined in Equation 2.9. The measure is purposefully not related to the set of all entities E, as a low coverage would be rewarded with a high Minimality. As we established in Sec-

tion 2.1, a smaller number of sources, and thus a higher *Minimality* score, is better.

- **Consistency:** We measure consistency as the average similarity between the datasets that were used to create the respective cover. A higher value means that the respective augmentation was created from more similar datasets. This corresponds to the consistency cover quality score as defined in Equation 2.10. While the consistency score is calculated fully automatically based on the similarity function sim, we also perform an additional manual evaluation of consistency and diversity using manual tagging described in Section 2.5.5
- **Diversity:** This indicator is measured not for a single cover, but a set of covers. We use the commonly employed method of measuring the average distance between all pairs of items in the result list. In our case, this corresponds to the diversity cover quality score as defined in Equation 2.11. Note that this score is not measured for a cover, but for a set of covers.

Relaxed Coverage To study the trade-offs between coverage and the other dimensions, we introduce early break parameters θ_{Cov} and θ_{Cons} to all of our algorithms. Specifically, we allow our algorithms to stop iterating if at least θ_{Cov} percent of the entities are covered, and the cover consistency would drop below θ_{Cons} by continuing. For example, in a *relaxed* scenario with $\theta_{Cov} = 0.9$ and $\theta_{Cons} = 0.4$, the algorithms are allowed to leave up to 10% of the entities uncovered, if the consistency score would otherwise fall below 0.4. However, if not mentioned differently θ_{Cov} is set to 1.0. In other words, we aim for full coverage of the queried entities regardless of consistency as the default setting.

Other Parameters If not mentioned differently, all parameters are kept constant throughout all the experiments. This is especially important for the underlying Web table retrieval and matching system that comes with a set of typical parameters of its own, e.g., threshold for string distances used in the matching step, or number of raw candidates retrieved from the Web table index. As this system is not the focus of the evaluation, we choose reasonable defaults and keep them constant. For the consistent set covering algorithms themselves we study the influence of $|E|$ and k, i.e., the number of entities in the query and the number of covers to create. The search space factor s, used in the Greedy* and Genetic approaches (Algorithms 2 and 3), determining the number of solutions to create or the number of generations respectively, is kept constant at 10 in the evaluation for space reasons.

2.5.2 Baseline Algorithms

We include two baseline algorithms into our evaluation: *ValueGrouping* and *TopRanked*. Both use exactly the same Web table retrieval and matching components as our proposed algorithms, so the set of candidate datasets and their scores are the same as well. Both of them do not follow a set covering approach, but augment values for each entity based on the scoring of the candidate sources. We will now describe both in detail:

- **ValueGrouping:** This baseline is modeled after the value prediction strategy used in (Yakout et al., 2012) and (Lehmberg et al., 2015). It collects all values found in all candidates into per-entity groups, and clusters these groups according to their values. It then

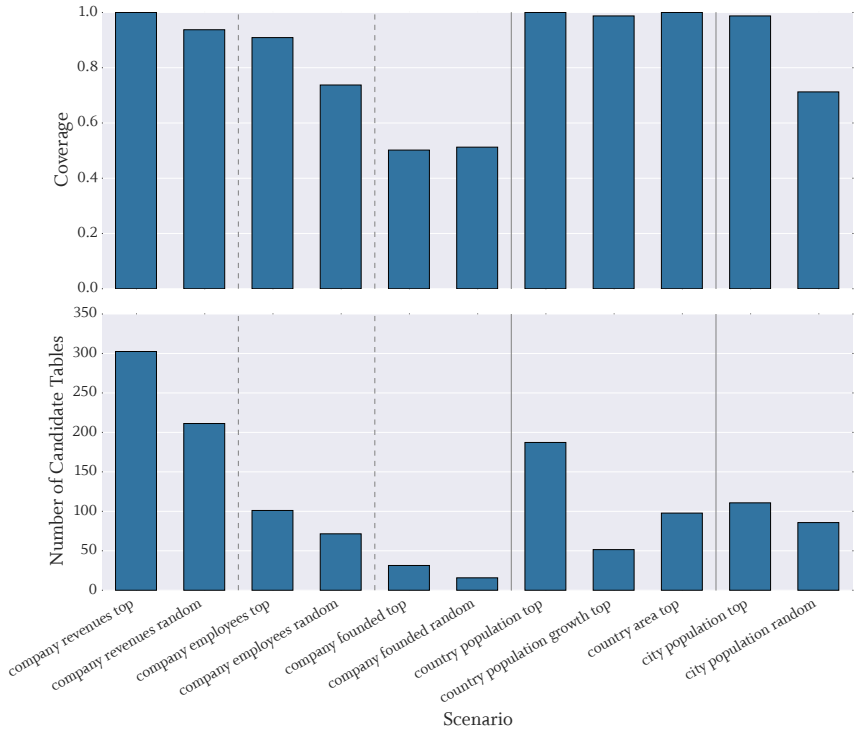


Figure 2.6: Coverable entities by domain

uses the highest scoring value from the cluster with the highest sum of scores, thus favoring high ranking data sources that are supported by many other sources with similar values. This baseline produces only one result value per entity, but often with a high-precision due to not simply considering the highest ranking data source, but considering evidence in the form of large clusters of similar values. It corresponds to the *fusion* paradigm shown in Figure 2.3.

- **TopRanked:** Here, the most relevant data source that can produce a value for an entity is chosen to provide the value for the first augmentation. The next augmentation is then constructed from the respective second most relevant value for each entity, until k results have been constructed. This approach does produce k different covers of the input entity set, but does not take consistency or diversity into consideration. It corresponds to the *TopRanked* paradigm shown in Figure 2.4a.

Having introduced data, queries, measures and baseline for our experiment, we will now discuss the results of our experiments, regarding coverage, precision, the cover quality indicators described above, and also runtime performance.

2.5.3 Evaluating Entity Coverage

As described above, absolute coverage, i.e., the percentage of queried entities for which a value could be retrieved from the Web table corpus, is independent of the top-k consistent set cover

Figure 2.7: Precision at Rank 1

algorithms that we study. Nevertheless, for reference and comparison with related work we give coverage statistics for the Dresden Web Table Corpus and our matching system REA in Figure 2.6.

The top part of the figure shows the coverage achieved, while the lower part shows $|D|$, the absolute number of candidate tables REA retrieved as input for the covering algorithms. While the absolute counts obviously depend on parameters and thresholds of the REA system, it still gives an impression of the frequency of tables about each domain in the DWTC. In this experiment, we also differentiate between *top* and *random* entity queries, as described in Section 2.5.1.

Generally, our results regarding coverage are in line with the results from (Zhang and Chakrabarti, 2013). In the city and country domain, coverage tends towards 1.0. For countries, the universe is sufficiently small so that most Web tables concerned with countries contain all, or a significant portion of them, making this domain the least difficult. For cities, the task could be made arbitrarily hard by querying very small cities, but as in (Zhang and Chakrabarti, 2013) we only consider sufficiently populous cities, so that the coverage is still high. The cities domain is still harder than the countries domain, as the total number of possible entities is not as limited. Finally, the company domain is the hardest of our test set. This is mainly due to the number of companies that exist and the many possible name clashes with very similar company names containing generic terms such as “East Japan Railway” or “National Grid”. Again, our numbers are in line with the related work: the top companies of the Forbes list can be easily found in the DWTC, while the coverage drops towards 50% for tail companies.

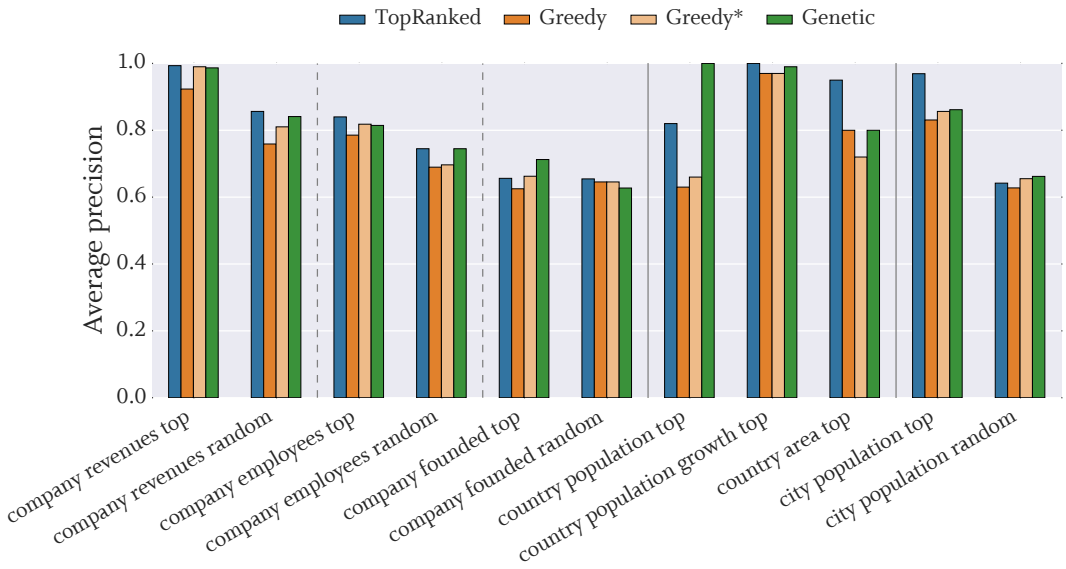


Figure 2.8: Precision up until Rank 5

	First Result is not the best Result	Average Improvement over first Result
Genetic	18%	15%
Greedy*	18%	20%
Greedy	27%	16%
TopRanked	32%	7%

Table 2.1: Queries with best solutions not on rank 1, and improvements over rank 1

2.5.4 Evaluating Single Entity Precision

In the case of our consistent set covering algorithms, we are not only interested in the correctness of the augmented values for each individual entity, but in the consistency of these values across a single cover. Still, we will first evaluate the relevance of the individual augmented values in this section, with no consideration of their composition to covers, and then study cover quality parameters as defined above in Section 2.5.1.

First, consider Figures 2.7 and 2.8, which depicts the average precision of each algorithm by queried domain/attribute combination. The first graph (2.7) shows the average precision of the respective first augmentation produced by each algorithm, while the second graph (2.8) shows the average precision over the top five results. Since one of the baseline algorithms, *ValueGrouping*, can not produce more than one augmentation, it is not shown in the second plot. For the *company* and *city* domain, we differentiate between top and random queries as described in Section 2.5.1.

Note that the differences between the algorithms in this experiment are minor. This is

because individual entity augmentation correctness is mainly dependent on the corpus and matching system used, and less so on the way the data sources are composed. At the first rank (Figure 2.7), when removing the outliers, the average precision of all algorithms is almost equal at about 88% and standard deviations between 10% (TopRanked) and 14% (Greedy*). When looking at single entity precision averaged until rank 5 using similar outlier correction, the greedy set cover-based methods drop to between 75% and 77%, while the Genetic and TopRanked approaches reach 82% over all domains. The differences in the general trend between the domains are similar to those in coverage (Section 2.5.3).

However, in section Section 2.1 we discussed that one of the reasons for introducing top-k entity augmentation is to increase *error tolerance*. In other words, we assumed that providing alternative solutions to the user would be advantageous in case the first solution is erroneous. To test this assumption, we measured the percentage of queries for which the best solution was not on rank 1, and how much the best solution improved over the first solution. We take the best solution to be the one with the highest individual precision, i.e., with the highest number of entities augmented with a relevant value. The results are given in Table 2.1. We can see that for the majority of queries, the first solution is the best solution, which is to be expected if the relevance scoring function has been constructed carefully. Still, for all methods, a considerable amount of queries, from 18% for Genetic and Greedy* approaches to 32% for the baseline, is better answered with one of the alternative augmentations. In these cases, the differences in precision are significant: The best solution's precision is, on average, 15% better than the one of the highest ranked solution. This supports our argument that top-k entity augmentation increases error tolerance by giving the user alternatives to choose from.

Note that this just considers single entity precision. Our findings concerning consistency and diversity presented in Section 2.5.5 will give further arguments for the top-k approach. There we will show that while the averaged single entity precision may be similar for all methods, the baseline strategies achieve their individual entity result quality by not considering the quality of the result as a whole.

2.5.5 Evaluating Cover Quality

In this section, we will demonstrate where our novel entity augmentation methods differentiate themselves from the baseline approaches: consistency and diversity, as well as minimality of the number of sources used. First, consider Figure 2.9, which depicts the five indicators consistency, diversity, relevance score, minimality and coverage by rank, from 0, the first augmentation, to 9, the last one in the $k = 10$ case. The plotted values are averages over the whole set of test queries. Since the *ValueGrouping* baseline algorithm can only produce a single result using majority voting between all possible values for an entity, it is only plotted as a larger triangle instead of a line.

In Figure 2.9, the coverage threshold θ_{Cov} (see Section 2.5.1) is set to 1.0, demanding full covers and representing the default case. We can see that the baseline algorithms, while producing results with very high average relevance scores, do so at the expense of using more different sources per cover, which is visible in the Minimality and the Consistency indicators. Here, the benefit of viewing the augmentation problem as a form of the set cover problem becomes apparent: the set cover-based algorithms can create solutions that are much smaller,

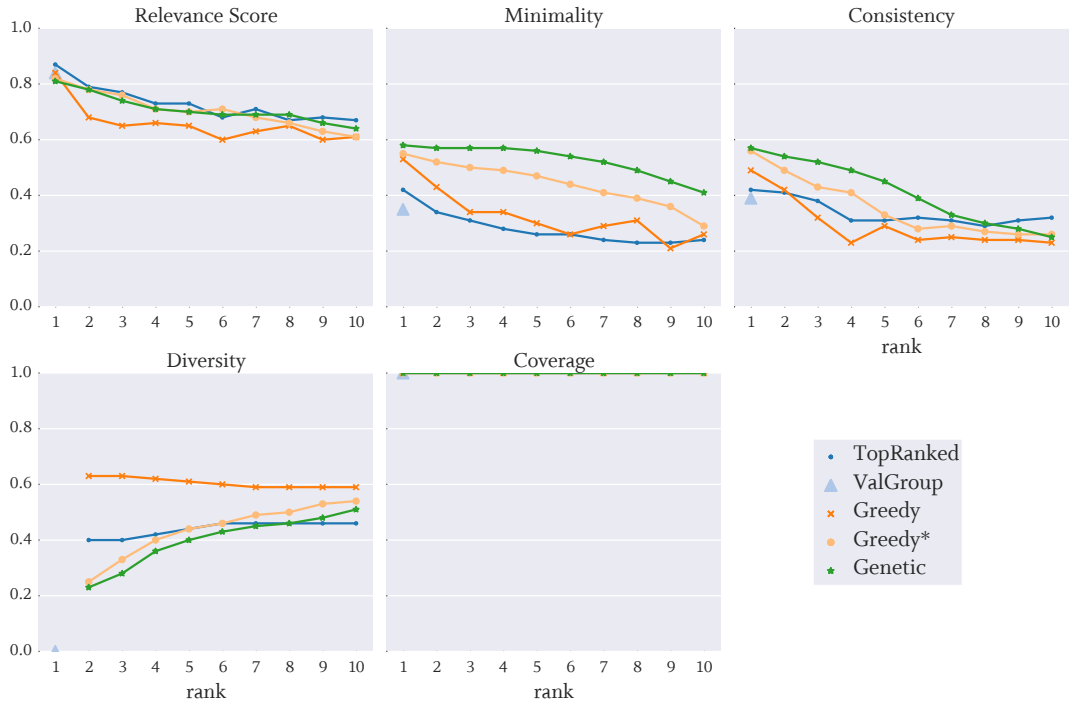


Figure 2.9: Cover quality indicators by rank, Full covers scenario: θ_{Cov}=1.0 and θ_{Cons}=0.0

without loosing noticeably in relevance. It is also quite noticeable that the basic Greedy approach, while constructing smaller solutions, is not able to outperform the baseline in terms of consistency or score. The Greedy* algorithm, by exploring more possible solutions is able to improve on the baseline, while the Genetic approach finally beats the baselines clearly with respect to minimality and consistency: In comparison to TopRanked, it produces results that are 21% more consistent and achieve a 87% better minimality (0.28 vs. 0.52) on average, while maintaining the average relevance score of the data sources used. Note that these are averages over all ranks, and the gains on the first ranks are even higher. Comparing the first cover created to the *ValueGrouping* baseline yields a 65% improvement in minimality (0.58 to 0.81) and 46% in consistency (0.39 to 0.57).

Now consider the diversity plots, which show at each rank the diversity for the result set from the first one up to and including the respective rank. The plots show that the algorithms behave differently with respect to using similar datasets for further solutions after the initial ones have been generated. The basic greedy approach moves through the similarity space of the candidate sources as it fills its usage matrix U (see Algorithm 1), penalizing used candidates and those similar to it. Since it creates at most k solutions, it is likely to pick new, unpenalized candidates in each iteration, which explains the strong diversity of the augmentations generated. The Greedy* approach on the other hand creates a multiple of the necessary solutions and will often "wrap around" the similarity space of candidates, reconsidering used sources in new combinations. The same is even more true for the Genetic approach, that will combine existing good solutions to new ones, promoting the survival of useful data sources.

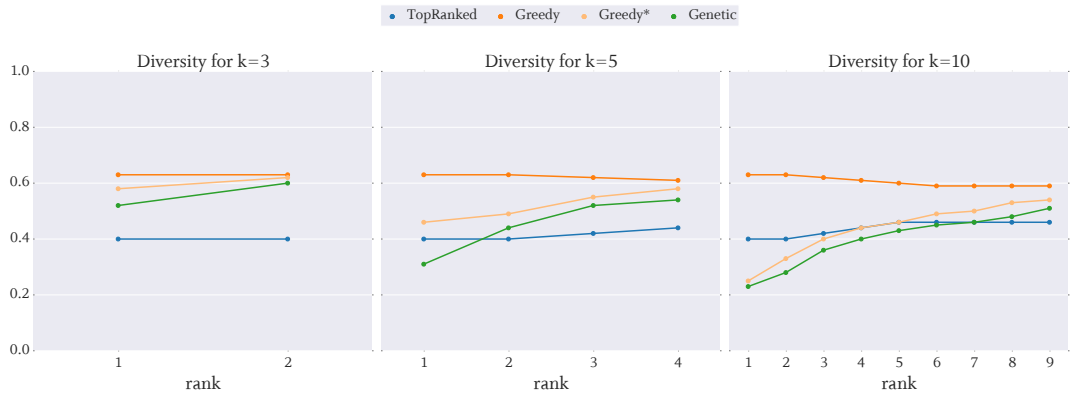


Figure 2.10: Diversity by Rank and by factor k

While the benefit of reusing good partial solutions is obvious from the other measures, this leads to more uniform results across the first results of the top-k list. Note however, that all algorithms converge to a similar diversity level as more solutions are added to the top-k list. At $k = 10$, the Genetic approach that dominates the baselines in all other aspects reaches similar levels of diversity when looking at the whole result list, even though it produces more similar solutions on the first few ranks. Measured over the whole result list, however, it produces covers that are still 10% more diverse than the TopRanked baseline. To summarize this experiment, the Greedy set covering approach produces the most diverse result list of all approaches, but is no improvement on the base in the other dimensions. The Genetic algorithm, however, greatly improves consistency and minimality, without sacrificing relevance.

Still, in this experiment it appears as if it only reaches high levels of diversity with larger number of requested solutions. However, this is in fact an artifact of our result selection strategy, which balances the various dimensions when picking and ranking results. Consider Figure 2.10, which shows only the diversity scores, but for various levels of k. We observe that even if k is set to a lower value, the genetic approach does still reaches the same level of diversity when calculated over the whole result set. This implies that the Genetic approach does have a tendency to produce more similar results on the first few ranks, but compensates for this in later ranks.

Next, consider Figures 2.11, depicting a *relaxed* scenario, and 2.12, depicting a *best-effort-coverage* scenario. In these figures we demonstrate the useful trade-off between coverage and cover quality that is possible with our algorithms: if we lower θ_{Cov}, i.e., we allow a certain amount of uncovered entities if necessary to keep consistency over θ_{Cons}. The baseline algorithms do not have early braking included and their results from the first experiment are included as a constant frame of reference. In the *relaxed* scenario (Figure 2.11), we allow 10% uncovered entities, if necessary to keep consistency above 0.4, and in the *best-effort-coverage* scenario (Figures 2.12) we allow the algorithms to leave as much uncovered as necessary to keep consistency above 0.4. Note that θ_{Cons} and θ_{Cov} are guidelines for the covering process, not hard limits. Allowing 10% uncovered entities can dramatically improve the results in all areas. The Minimality score of the TopRanked baseline is doubled by the Genetic approach, while also improving consistency by 49%. Interestingly, the results are even more diverse in

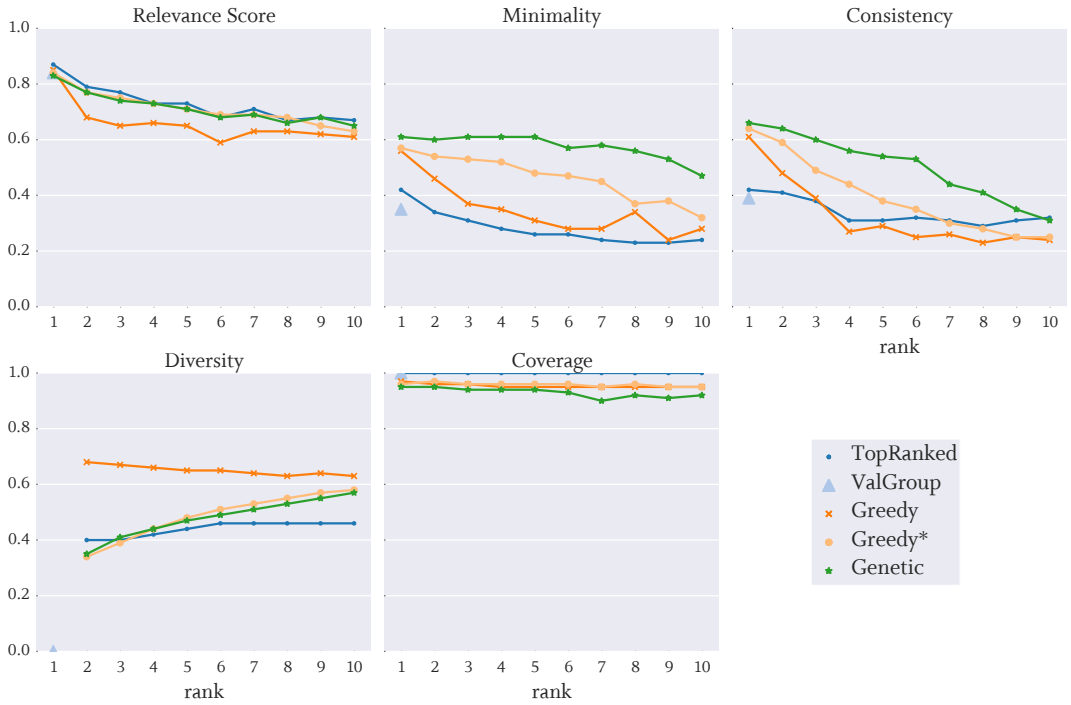


Figure 2.11: Cover quality indicators by rank, Relaxed scenario: θ_{Cov}=0.9 and θ_{Cons}=0.4

this case, improving 12% over the default case for the genetic algorithm. These two observations can be explained by the fact that certain entities can only be covered by data sources that have no fitting "partner" sources, which reduces consistency if they have to be covered in one cover, while other entities are coverable by only a small number of sources, which then have to be included in almost every cover, reducing diversity. These "unfavorable" data sources are skipped by our algorithms, if the coverage requirement is relaxed.

In the third covering experiment shown in 2.12, we allow aggressive optimization of the result regarding consistency by sacrificing coverage completely. We can see that perfectly consistent and diverse results can mostly only be reached by not using more than one source: both Greedy* and Genetic return mostly single datasets as single augmentation on the first few ranks. Note how in this configuration the algorithms all select almost completely different datasets for every augmentation as the diversity stays above 0.8 throughout the higher ranks, but to do so have to choose smaller and smaller datasets, which is visible in the coverage quickly falling to unusable levels.

To summarize, not only does the set covering-based approach to top-k entity augmentation improve the quality of the results in all measured dimensions, using the parameters θ_{Cov} θ_{Cons} also allows the algorithms to be tuned to varying use cases.

2.5.6 Manual Cover Quality Verification

Until now we have verified the consistency and diversity aspect of the evaluation using the same similarity function sim that is used by the algorithms to create the solutions. Therefore,

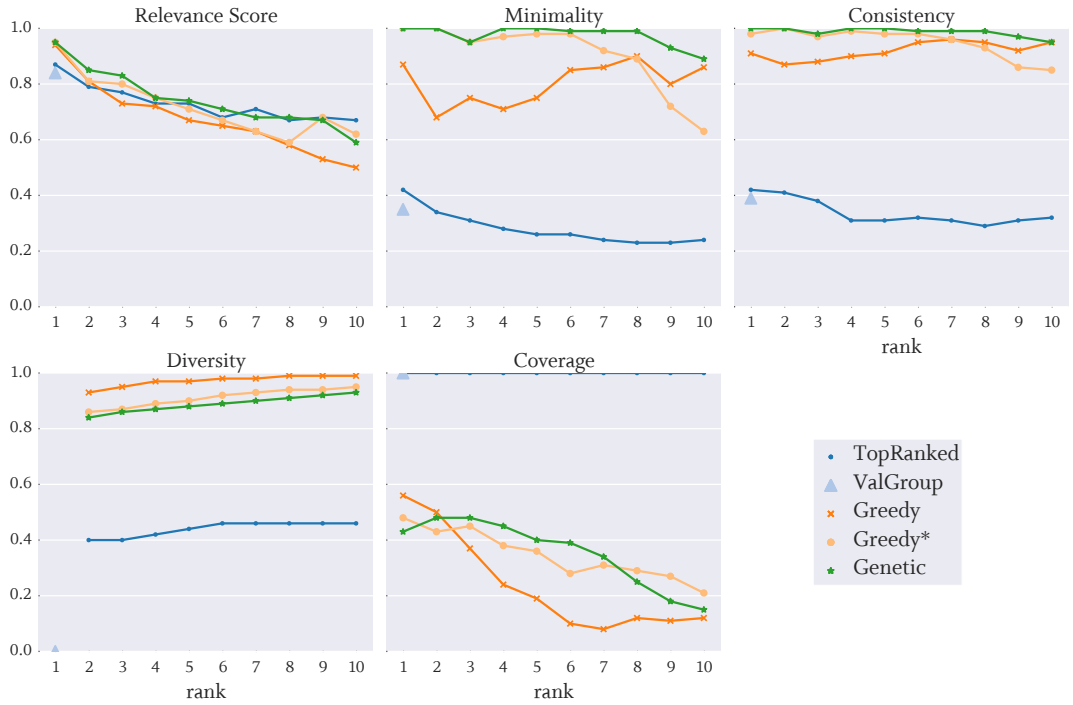


Figure 2.12: Cover quality indicators by rank, Best-effort-coverage scenario: θ_{Cov}=0.0 and θ_{Cons}=0.4

we added an additional test to verify our findings with respect to consistency and diversity, which are based on this function. Specifically, we used a manual tagging approach to supplement the purely automatic calculation of similarities. The same human judges that decided dataset relevance were also asked to attach tags to the datasets they judged that characterize them. Typical tags were units of measurement, years, orders of magnitude, but also attribute variations, such as "under 25" for the queried attribute "population". Figure 2.13 show the results of counting the number of distinct tags in the covers per rank, as well the cumulative number of distinct tags collected up until the rank. If a cover has many different tags, this implies that it was created from very different datasets, i.e., its consistency is low. If, on the other hand, a cover has only few distinct tags attached, this hints at a more consistent choice of datasets. Concerning the cumulative count, a high count at higher ranks means that many different datasets were used throughout the results list, and can be seen as evidence of a more diverse result set.

The results are shown in Figure 2.13. They confirm the findings of our main experiment: the results produced by the baseline algorithms collect more tags starting from the very first rank. The *ValueGrouping* baseline, which majority votes on the correct value for each entity individually, performs even worse, as is to be expected. It produces only one result, which carries an average 8.5 tags, 25% more than the *Genetic* approach at rank 1, with 6.7 tags in average. The *TopRanked* baseline is more competitive with our results, though strictly worse regarding consistency to the *Genetic* approach, using more tags at each rank. With respect to diversity,

Figure 2.13: Average number of distinct tags in covers on each rank and cumulative distinct until rank, Full Coverage Scenario (θ_{Cov}=1.0 and θ_{Cons}=0.0)

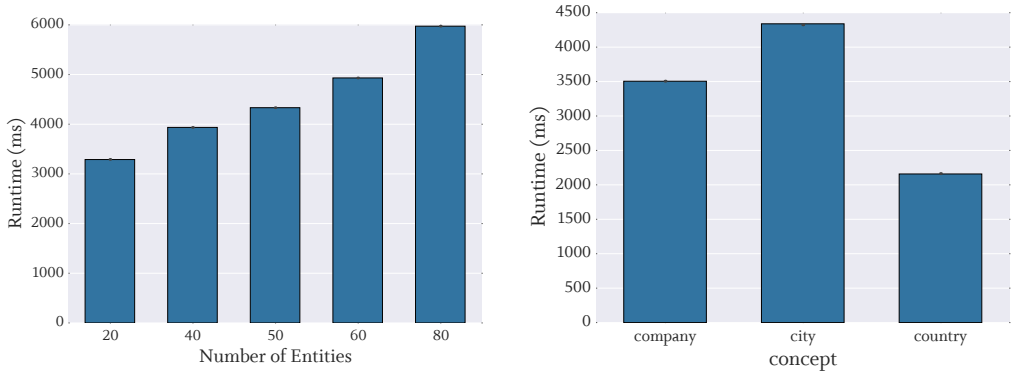
measured through cumulative distinct tags, it beats the *Genetic* approach for very small result lists, but for more than three results, the diversity drops below all of the approaches proposed in this chapter. Comparing our approaches, the results line up well with those measured using only automatic similarity functions, as given in Figure 2.9.

In general, there is a trade-off between consistency and diversity. For example, over the course of the ranks, the basic *Greedy* approach picks up very very high numbers of tags individually, more than the *TopRanked* baseline starting with the second results. However, it also produces much more diverse result lists, as shown by the high number of cumulative distinct tags. The same holds for the *Greedy** and *Genetic* approaches: they produce covers with less distinct tags, especially over the lower ranks, where the gains over the baseline are even more visible. The trade-off, as predicted, is that fewer distinct tags are picked up over the result list as a whole, i.e., the diversity of the list is smaller than with the *Greedy* approach. Note, that we performed this manual tagging experiment only up to rank five. The main experiment (Figure 2.9) suggest that the diversity our methods would converge on the high level of the *Greedy* approach when more results are generated, while the *TopRanked* approach should not.

2.5.7 Evaluating Runtime Performance

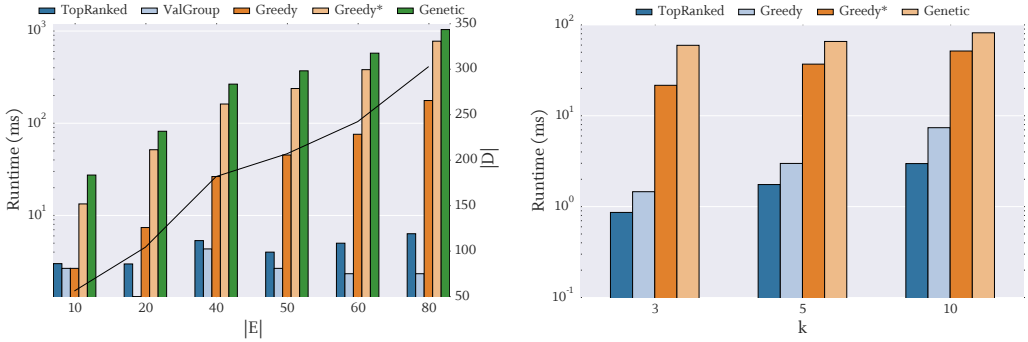
We evaluate the runtime performance of the proposed methods on a Ubuntu 12.04 Virtual Machine running on 2.4GHz Intel Xeon CPUs. The algorithms are implemented in Scala, using version 2.11, and are executed on a version 1.8.0 Oracle JVM. The set of queries is the same as in the main experiment reported on in Section 2.5.5.

First let us consider the runtime of the Web table retrieval and the schema and instance matching step. It is a constant overhead for all methods, whether baseline or set covering-based, and is therefore measured separately. It does not include the similarity calculations, as they are part of our methods only and not of the baseline algorithms. The results are shown in figure 2.14, once as a function of $|E|$, the number of queried entities, and once by queried domain concept. In Figure 2.14a, we can observe that the runtime grows roughly linearly with the number of queried entities. This is due to the REA system having to retrieve and match



(a) By number of entities $|E|$ (b) By query concept, fixed $|E| = 20$

Figure 2.14: Retrieval and matching runtimes

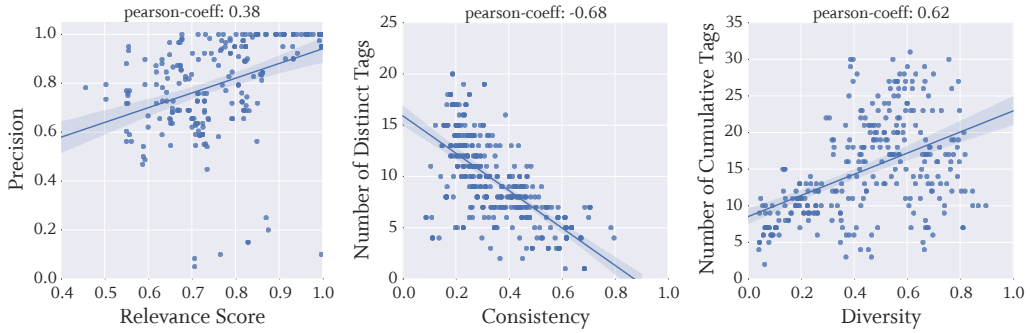


(a) By number of entities $|E|$, fixed $k = 10$ (b) By numer of covers k, fixed $|E| = 20$

Figure 2.15: Cover-generation runtimes

more candidate tables, as covering more entities requires more sources in most cases. From Figure 2.14b however, we can see that this effect varies by domain. For example, in the case of countries, the runtime is on average only 60% of the runtime for companies. This effect is due the size of the country domain. Since Web tables about countries often contain information about all possible entities of the domain, less tables have to be retrieved and matched to identify an adequate candidate set.

Next, we will consider the runtime of the cover generation step, i.e., the runtime of our set covering algorithms and the baseline approaches. Note, that these results do include the generation of the similarity values between the candidate sources and all other overhead that is associated with our novel methods. We give the runtimes once as a function of $|E|$, the number of entities queried in Figure 2.15a, and once as a function of k, the number of covers to be created in Figure 2.15b. General set cover algorithms' performance depends both on the size of the universe, or in our case $|E|$, and on the number of candidate subsets, or of candidate data sources $|D|$ in our case. In our system however, the number of candidates $|D|$ directly correlates with $|E|$, as our retrieval and matching system returns more candidate Web tables when more entities are queried. This is shown on the secondary axis in Figure 2.15a. We thus



(a) Relevance score vs. Precision

(b) Consistency Score vs. Distinct Tags assigned

(c) Diversity Score vs. Cumulative Tags assigned

Figure 2.16: Quality of rel and sim measured by correlation with human judgments

measure the influence of both dimensions of the problem size in one experiment.

From the results it is clear that the baseline techniques are superior in their runtime performance due to their simplicity. Both consist of grouping possible values by entity, and ordering by score in the case of *TopRanked*, or by fuzzy majority vote on the values in the case of *ValueGrouping*. The set covering approaches are more complex, moving them into a different order of magnitude concerning runtime. Still, even our prototypical implementation was able to answer all our test queries in less than one second.

Comparing the two sets of measurements, we can observe that the runtime for the retrieval and matching clearly dominates the overall runtime. This implies that, considering the improvement in result quality, the additional overhead incurred by using our set covering-based methods is justifiable in context of the whole augmentation operation.

2.5.8 Quality of Relevance and Similarity Functions

The functions $\text{sim} : D \times D \rightarrow [0, 1]$ and $\text{rel} : D \rightarrow [0, 1]$ are of crucial importance to the performance of our method. While their specific implementation is orthogonal to the set covering-based augmentation algorithms that we study in this chapter, we still evaluate their performance in this section. This allows us to assess the Web table scoring and similarity features proposed for the REA system, and put our main results into perspective.

The relevance and similarity features used in REA were presented in Sections 2.4.2 and 2.4.3. To evaluate them, we use the human judgments we created for the precision, consistency and diversity evaluations. In Figure 2.16, we show the results of three correlation analyses. We evaluate the correlation between the *relevance* estimation score $rel(d)$ and the manual data source relevance assessments used in Section 2.5.4 in Figure 2.16a. Next, we use the manual tagging results introduced in Section 2.5.6 and evaluate the correlation between the distinct tags per solution and *consistency* in Figure 2.16b, as well as the cumulative number of tags in a result set and *diversity* in Figure 2.16c.

The results show strong correlations between the tagging results and the two measures derived from $sim(d_i, d_j)$, diversity and consistency, and a weaker correlation of the relevance

score with the relevance judgments. This strengthens our results for the consistency and diversity improvements from Section 2.5.5, and increases confidence that the gains measured based on the sim-function translate to real solution quality improvements experiencable by users. While the relevance score is less well correlated with actual precision, it also is of lesser importance to the claims we made in the evaluation, since our methods are not aimed at improving by-entity precision.

2.6 RELATED WORK

In this section, we will survey related work and put it in relation to our method for top-k entity augmentation. We will first give an overview of the usage of Web tables as data sources in general, and then detail approaches to Web table-based *entity augmentation*. Then we survey *set cover* algorithms and applications, and compare multi-source Web data integration approaches from *data fusion* research. We finally discuss the field of query result *diversification*.

2.6.1 Web Tables

Notable first work on using the wealth of tables on the Web was done in (Cafarella et al., 2008). The authors were the first to extract a large scale corpus of tables from the Web. They found that a Google's internal general-purpose Web crawl at that time contained about 14.1 billion HTML tables, of which about 154 million contained useful relational data. They introduced a method for differentiating those data-carrying tables from *layout tables*, that make up the majority of actual HTML tables. Finally, they designed several applications for the extracted corpus, including table search, attribute synonym discovery and schema auto-completion. These initial approaches sparked the development of a large body of work that uses Web table corpora for specific query types. For example, in (Pimplikar and Sarawagi, 2012) whole tables are materialized automatically just from keyword queries without any known entities, while (Yin et al., 2011) uses Web tables to answer single fact keyword queries. Other examples of Web table application include data imputation, i.e., the retrieval of missing values in an existing dataset (Li et al., 2014b,c), or entity expansion (Dalvi et al., 2012a; Wang et al., 2015). Furthermore, there is more work about similarity between tables such as (Das Sarma et al., 2012), where the authors propose methods to identify tables related to a given query table in a large corpus.

Another class of work focuses on understanding the content of Web tables by trying to match table columns to an existing knowledge base such as (Limaye et al., 2010; Venetis et al., 2011). For example, (Venetis et al., 2011) uses two knowledge bases, a *isA* and a relations database, both extracted from the Web, to create labels for Web tables. Basically, the instances in the table are compared to instances in the knowledge bases, and if sufficient overlap is found, the class label of the knowledge base is used as the column label. Similar work has been done in (Limaye et al., 2010), but there a graphical model was used to label entities, attributes and relationships between them simultaneously in a holistic way, instead of labeling them independently and loosing context information. These approaches are extended in (Fan et al., 2014) to include crowd sourced human judgments into the concept matching process. The main contribution of the paper is a model for choosing which columns to forward to the crowd for human

inspection, as well as a hybrid concept inference model that incorporates the crowd sourced answers into the inference. Finally, there are studies on formalizing and classifying Web tables, as well as into quantifying their occurrence on the Web, such as (Crestan and Pantel, 2011) and (Lautert et al., 2013).

2.6.2 Entity Augmentation

An early publication on Web table-based entity augmentation is (Cafarella et al., 2009), which is concerned with automating the search for relevant Web tables. The paper does not aim at fully automated table extension, but proposes a set of operators that are to be used in a semi-automatic process, enabling the user to search for tables, extract their context, but also to extend a found table with information from a related table. This last operator, called *extend*, corresponds to our notion of entity augmentation. The paper proposes an algorithm called *MultiJoin*, that attempts to find matching Web tables for each queried entity independently, and then clusters the tables found to return the cluster with the largest coverage. However, it does not try to construct consistent solutions, but returns the set of possible values for each entity.

A strongly related work is the InfoGather system (Yakout et al., 2012), and its extension InfoGather+ (Zhang and Chakrabarti, 2013). The first system introduces Web table-based entity augmentation, as well as related operations such as attribute name-based table queries. InfoGather improved the state of the art especially by identifying more candidate tables than a naïve matching approach, while eliminating many spurious matches at the same time. This is achieved by introducing Web table similarity measures, and identifying tables indirectly matching the query through them. They also introduce methods for efficiently computing the similarity graph between all indexed tables offline, which could be applied to our approach as well, as we currently perform similarity calculations online. This is due to the main difference between the two approaches: InfoGather uses the similarity graph to find more candidate tables, but the actual choice of the augmented value is done per entity and based on fuzzy grouping of the values. In contrast, we use the similarity measures as evidence for selecting a minimal number of Web tables for our answer, and also produce alternative solutions, which would not be straightforward based on the fuzzy grouping value prediction. InfoGather+ improves the system by tackling similar consistency issues as our work: it assigns labels for time and units of measurements to tables, and propagates these labels along the similarity graph described above to other tables where such labels can not be found directly. While InfoGather+ tackles the problem of producing more consistent results from various possible Web sources, it does not produce top-k results, or minimize the number of sources used.

The basic problem that there will be more than one correct answer for many augmentation queries, e.g., multiple *revenue* values for single company because of different years of validity, is also explored in (Sarawagi and Chakrabarti, 2014). Specifically, the work targets quantity queries, i.e., queries for a numeric attribute of a certain entity. The earlier InfoGather+ already allows the user to specify a unit of measurement and a year-of-validity, and will only try to retrieve a single attribute value with this specific constraints. The QEWT system presented in (Sarawagi and Chakrabarti, 2014), on the other hand, solves this problem by modeling the query answer as a probability distribution over the retrieved values, and then returning

a ranked list of intervals as the final query answer. They contribute a table unit annotator that returns probabilistic units, instead of fixing one unit at extraction time, which allows to decide the unit of a table at query time as part of a collective inference model over multiple answer tables. This work is similar in spirit to ours, in that it does not try to simplify complex real-world attributes into single values, but deal with the uncertainty of data explicitly. While we return top-k ranked covers of a set of entities, which still contain the original values extracted from the Web, they fuse values into distributions and intervals, which are then ranked and presented to the user. We maintain that for many applications, such an abstracted answer will not be ideal, as the user will want to know exactly where the answers come from, and will therefore want explicit augmentation values tied to specific sources.

In (Lehmberg et al., 2015), a table augmentation system called *Mannheim SearchJoin Engine* is proposed that, in addition to entity augmentation given a specific attribute, also supports *unconstrained* queries, i.e., queries in which, given only a set of entities, all possible augmentation attributes are to be retrieved. Their method of dealing with multiple, possibly conflicting sources, by merging values using clustering and majority voting, is similar to (Yakout et al., 2012). We argue that this way of dealing with multiple values from different sources will not lead to usable results in many cases. In this merging and clustering step, attributes variations such as "GDP (€)", "GDP ($)" or "GDP PPP" will be merged, their different semantics will be lost, and the user will not be able to choose the most fitting variant for the information need at hand. These unconstrained queries, however, are useful in some exploratory search scenarios, where a user is interested in a set of entities, but has not yet focused on a particular direction of investigation.

In (Morcos et al., 2015) a comprehensive system for so-called *transformation queries*, that largely correspond to entity augmentation queries, is envisioned. The papers main contribution is that it proposes a system, named *DataXFormer*, that includes multiple transformation subsystems, based on Web tables, wrapped Web forms, as well as crowdsourcing, although it does not give specific methods of combining the subsystems. Furthermore, it also relies on returning a single value for each queried attribute.

In Table 2.2, we summarize our analysis of directly comparable entity augmentation systems with respect to the challenges we identified in Section 2.1. It shows that no existing system offers a comprehensive solution to all of these challenges. Infogather, while offering a more sophisticated Web table retrieval and matching method than REA, deals with none of the novel challenges that we identified. Infogather+ deals with some attribute variations by allowing exact specifications of the queried time frame and unit of measurement. The QEWT system partly deals with unclear user intent by introducing top-k intervals as the query result, instead of returning a single answer, which is similar to REA. The SearchJoin Engine introduces an unfocused form of exploratory querying through their unconstrained queries. Finally, the DataXFormer system improves error tolerance, by passing difficult queries to human experts via crowd sourcing, although they introduce no specific method for doing so. In all existing systems, the query result is centered on the augmentation value, and not on the sources used to provide the values, which means that trust and lineage are out of scope for all of these systems. In this section, we only discussed augmentation techniques that use Web tables as data sources. However, there are also approaches based on other types of sources. For example, (Löser et al., 2011) iteratively generate Web search keyword queries for retrieving text documents relevant

	Attribute Variations	Unclear User Intent	Trust & Lineage	Exploratory Search	Error Tolerance
InfoGather					
InfoGather+	✓				
QEWT	✓	✓			
SearchJoin				(✓)	
DataXFormer					(✓)
REA	✓	✓	✓	✓	✓

Table 2.2: Comparison of REA with related work.

to a table augmentation task, and then employ information extraction methods to obtain the desired attributes from free text. We view the specific choice of data source as an orthogonal problem to ours. Since the facts extracted from a Web page are transformed into a tabular form for integration, they could be used as candidate data sources for our set covering-based augmentation methods just as well as Web tables.

2.6.3 Mashups Tools

One of the central properties of entity augmentation systems is their declarative nature: users specify their information need simply by an attribute name, i.e, a keyword query. The EAS then automatically retrieves sources and creates mappings to existing user data. While this approach is sufficient in many use cases, more complex Web data integration problems may require more sophisticated, programmatic specifications. However, general purpose programming languages are not ideal for Web integration as well, as they overburden non-expert users, while most of their complexity is not even needed for all integration scenarios.

One class of approaches to this problem, called *Mashups tools*, aims at allowing users with only basic programming knowledge to easily compose various data sources to higher level services (Di Lorenzo et al., 2009). Many of these tools have been proposed, which differ in their level of abstraction, community features, user interfaces, or support for discovery of data sources and mashups (Grammel and Storey, 2010). Similarly to our work, most research on mashups uses public Web data sources because of their general availability. However, mashups have also been recognized as a lightweight form of data integration for the enterprise context, which is exemplified by systems such as IBM's Damia (Simmen et al., 2008).

The most well-known early Web mashup tool was Yahoo Pipes[9]. It allowed users to compose Web data sources such as RSS feeds or Web services in a graph-based drag and drop user interface. It did, however, not include sophisticated data integration features, such as entity resolution. In (Thor et al., 2007), a framework for mashup construction with strong focus on data integration is proposed. It adds more complex data transformation operators, such as a *fuse* operator for automatic object matching, and an aggregation operator to reduce multiple matches that result from a fuse operation to a concise representation. Furthermore, it

[9] `https://en.wikipedia.org/wiki/Yahoo!_Pipes`

introduced entity search strategies that minimize the number of queries that need to be issued to data sources such as entity search engines or Deep Web databases. These strategies where explored in more detail in (Endrullis et al., 2012a), where different query generators are combined with query ranking and selection strategies to form an adaptive querying process, which also utilizes initial query results to optimize the choice of further queries. In (Endrullis et al., 2012b), this feature set is further extended to include efficient, pipelined execution of such query processes, enabling stream-based processing and quick presentation of initial integration results. It further includes a declarative and extendable workflow definition DSL, that forms the basis for automatic GUI generation.

There are many other approaches aimed at supporting end-users in the creation of mashups. One example, called *mashup autocompletion*, was proposed in (Greenshpan et al., 2009). This work aims at recommending components and connection patterns for partial user mashups. The core idea is that many mashups have common characteristics, which can be exploited to provide completions based on previous user mashups.

By combining approaches such as the ones outlined here, mashups tools can become a powerful alternative to traditional data integration processes. While they are more expressive than entity augmentation systems, they are only applicable in situations where the sources to be integrated are known, and a user with programming knowledge is available to define the mashup and necessary source adapters.

2.6.4 Set Covering

The set covering problem, on which we model our problem of minimal consistent entity augmentation in Section 2.3.2, has been extensively studied in *operations research*, a field at the intersection of math, computer science and economics. It is concerned with the application of mathematical models and optimization strategies to problems of decision support (Winston and Goldberg, 2004). The field has a strong focus on practical use cases, and as such, set covering methods developed in the field have been applied to diverse problems such as crew scheduling, facilities location, and network design (Vemuganti, 1999).

In addition to having many practical applications, it is, in fact, one of Karp's original 21 NP complete problems (Karp, 1972), implying the need for heuristic solutions. These two properties have led to many methods based on various optimization techniques being developed. While exact algorithms have been proposed, e.g., based on the Branch-and-Bound paradigm, Lagrangian and LP relaxations (see (Caprara et al., 2000) for a survey), most algorithms give approximate results. The most common algorithm, which also forms the core of many more sophisticated techniques, is the greedy approach as proposed in (Chvatal, 1979). Since it is also the basis of our approach, we already gave a short overview in Section 2.3.2. While being rather straightforward to implement, (Chvatal, 1979) has shown that the approximation ratio of the greedy approach is $H(s') = \sum_{k=1}^{n} \frac{1}{k}$, i.e, the *harmonic number* of s', the largest set in S. Later results (Feige, 1998; Dinur and Steurer, 2014) even showed that approximating the set cover problem withing a factor of $(1 - o(1))ln(n)$ is NP-hard, which implies that the original greedy heuristic is already almost as good as any approximate polynomial-time algorithm for the problem can be. This, together with its simple implementation, makes the Greedy approach the most popular approach to set covering. For these same reasons, it was chosen as the core of our

top-k consistent set covering framework (Section 2.3.2).

Our methods for generating top-k covers are inspired by multi-start optimization methods (Martí et al., 2013), such as GRASP (Feo and Resende, 1995) and Meta-RaPS (DePuy et al., 2005), which have been applied to the set cover problem among others in (Bautista and Pereira, 2007) and (Lan et al., 2007). On a high level, these methods combine multiple iterations of a randomized *construction* phase and a local *improvement* phase. In the first phase, a solution is created using some heuristic, e.g., the Greedy approach, but adding in some form of randomization. The randomization allows the algorithm to create slightly different results in multiple runs. This is achieved for example by randomly making non-optimal decisions in the individual steps of the respective algorithm. Then, in the *improvement* phase, the generated results are improved using some form of local optimization. For example, in the set covering case, this local optimization is achieved by solving a smaller set cover problem derived from the uncertain parts of the initial solution. We adopted the idea of running a core algorithm several times and picking the best of several runs for our approach, and evaluated a GRASP algorithm for our problem, which however did not improve the results in our case, and was dropped in favor of our genetic approach.

A second inspiration for our top-k approach is *tabu search* (Glover, 1989), in which an initial solution is the starting point for multiple iterations of a neighborhood search aimed at improving the solution. The distinguishing feature is the *tabu list*, which stores already visited parts of the solutions space. It is used to prohibit the algorithm from returning to an already visited part of the solution space, unless a so-called aspiration criterion is met, or a certain amount of iterations has passed. In this way, the search is prevented from getting stuck in, or repeatedly revisiting, the same local optima. Our approach tracks which entities have already been covered by which datasets in previous solutions by using a *usage matrix* (see Algorithm 1 in Section 2.3.2). Similarly to tabu lists, this matrix is then used to discourage the algorithm from making those choices again, preventing future iterations from creating similar covers. This was partly inspired by the concept of tabu lists, except that our approach does not prohibit certain choices, but only adjust their weights. Also we do not create a single, best solutions as traditional tabu search does, but keep the usage matrix over multiple iterations, to explore diverse regions of the solution space.

So far we have only discussed work related to the basic, greedy core of our covering framework. Next, we want to highlight related work on applying evolutionary heuristics to the set cover problem. The most related work is (Beasley and Chu, 1996), as our genetic set covering approach is inspired by this work. It first treats the problem of how to represent covers as a genome using traditional binary feature vectors, and then proceeds to discusses the main problem with the application of genetic algorithms to set covering: Genetic operators such as *mutate* and *cross*, do not necessarily produce feasible solutions, i.e., complete covers, when applied to an existing feasible solution. In other words, applying a traditional *cross* to the genomes of two set covers does not necessarily produce a new cover. In (Beasley and Chu, 1996), this problem is solved using an additional heuristic feasibility operator, which can be seen as a local search on the near-feasible solutions produced by the cross operator. In addition, the paper also introduced a mechanism for adapting the mutation rate, i.e, the probability for random changes in each generation, according to the convergence of the algorithm. As described in Section 2.3.5, we choose the genetic approach as it naturally models a pool of solutions from which a top-k

can be drawn, and allows easy inclusion of our consistency and diversity constraints through suitable choice of fitness function, parent selection and replacement strategy.

Apart from basic research on the abstract set cover problem, there have also been database and data integration specific applications of the set cover problem. For example, (Munagala et al., 2005) formulated the query optimization problem of finding the optimal order of a set of possibly correlated selections as new variant of the set covering problem, called *pipelined set covering*. In this variant of the problem, the sets are associated with a per-element cost, and the aim is to find a sequence (not just a set) of these subsets that minimize the total cost to cover all elements. In (Cormode et al., 2010), the greedy algorithm for set covering is studied with respect to large, disk-resident databases. The authors observed that using the traditional greedy algorithm leads to many random disk seeks, which impedes performance. They therefore present a relaxed greedy algorithm that minimizes random access by considering blocks of sets grouped by size instead of all sets in each iteration. They show that the result quality is almost identical to that of the greedy results, while offering improved scalability.

Since we study a variation of the set cover problem with new objectives and constraints, we will finish our discussion by studying other variations of the problem that have been studied in related work. The first one we will examine is the *Set MultiCover* problem (Hua et al., 2010), in which the objective is to cover each element in the universe not only once, but multiple times. While this is superficially similar to our problems, a multi-cover is not the same as our top-k covers. A multi-cover is one object, i.e., one set of subsets that covers each element in the universe multiple times. This corresponds to a flat list of multiple augmentation values per entity, when mapped into our problem domain. As motivated in Section 2.2, we aim at k internally consistent results, not only k values per entity. Furthermore, consistency or diversity are not considered in this version of the problem.

Another variation of the problem, called the *Size-Constrained Weighted Set Cover* problem was recently studied in (Golab et al., 2015). It aims at finding a cover of maximum weight, just as in weighted set cover, but with the additional constraint of using at most k subsets, as in the *Maximum Coverage* problem (Hochbaum, 1997), which makes it a generalization of the two problems. While it is quite different from our top-k covers, the goal of limiting the number of subsets that are used for covering is similar to our aim of *minimality* as motivated in Section 2.1.

The problem we studied in this chapter is also variation of the original problem, introducing two new challenges, *cover consistency* and *diversified top-k result computation*. Both were modeled by defining a similarity function between the datasets. In terms of the set cover problem, we included additional weights that are not based on the sets or the elements, but on combinations of sets, i.e., we added a weighting function $f : S \times S \rightarrow [0, 1]$. This leads to choices of sets influencing each other. Furthermore we extended the original problem with the diversified top-k goal. To the best of our knowledge this variant of the problem is novel and has not been studied before in other contexts.

2.6.5 Data Fusion

There is a large body of related work on general as well as Web data fusion and truth discovery from multiple conflicting or correlating sources, see (Bleiholder and Naumann, 2009; Li et al., 2013a) for surveys. This class of work is concerned with fusing many sources for a fact to single

a truth, while we aim at providing "alternative truths" for a set of facts using a minimal number of consistent sources. In (Dong et al., 2013), a similar argument regarding minimizing the number of sources to consider is made, though their aim is to reduce the search space for data fusion techniques before applying them, while we aim at creating minimal covers of an entity set to make the result easier to understand and verify. They model existing fusion methods and study their monotonicity, i.e., whether adding more sources to be integrated will always yield better result with respect to cost and gain, measured by integration accuracy. They introduce a new fusion model that is monotonic to solve the marginalism problem, i.e., finding the largest number of sources to integrate before cost outweighs gain.

Furthermore, there are works in data fusion that are concerned with efficiently covering a domain using a large number of overlapping or *dependent* sources. For example, in (Sarma et al., 2011), three optimization problems that arise from this scenario are discussed: first, a cost-minimization problem, i.e., cover the whole domain while incurring minimal costs, then second, a maximum-coverage problem, i.e., given a fixed budget, maximize the coverage, and thirst, a source-ordering problem, i.e., in which order to query the sources to minimize latency. The problem studied is similar to ours, in that it is essentially set cover with an additional graph overlayed on the sources that influences decisions. In (Sarma et al., 2011), the graph is acyclic and models copying-dependencies between sources, and is utilized to estimate the coverage of subsets of sources, and to minimize the number of sources that have to be queried. In our scenario, we use a full similarity graph between all sources, and use it for our extended objective that includes consistency and top-k diversity. The objective of minimizing the sources is common with our approach, but other factors of answer quality are not considered.

While (Sarma et al., 2011) assumes that the exact nature of the copying dependencies are known a-priori and a static choice of sources can thus be computed, (Salloum et al., 2013) tackles a similar problem in an online setting, i.e., it can update the order of sources that it queries on-the-fly, given statistics collected from already queried sources. Our approach assumes full knowledge of the sources as well, as it works on a candidate set of sources that can be fully inspected at run-time. However, if our approach should be extended to sources that are queried online, the insights from (Salloum et al., 2013) could be applicable.

(Gupta and Sarawagi, 2009) introduced a system for answering *table augmentation queries* from unstructured lists extracted from the Web. This query type aims at automatically completing a table that is given with its schema, and a small set of example tuples. The paper introduces a statistical extraction method for segmenting unstructured lists into tuples with defined attributes. The paper also introduces methods for merging multiple tables created this way into a consistent single table. The method first creates a target schema for the query table, and then trains per-column cell resolvers, which score an arbitrary value with respect to a column. These cell resolvers are then combined to row-resolvers that are resistant to missing values and erroneous extraction, by means of Bayesian networks. These row-resolvers are used to merge equivalent tuples from different input tables. Finally, to rank the rows in the consolidated table by relevance, they combine frequency of rows in the set of input tables, extraction confidence, and a column-specific importance score. The technique was later extended to use Web tables as input, and require only a keyword query describing the target table, with no examples being necessary, in (Pimplikar and Sarawagi, 2012). This reduction in requirements is achieved by using the more expressive Web tables as data sources instead of lists. For example,

using tables enables the usage of column overlap between various candidate tables to improve matching accuracy. The paper details a graphical model for collectively mapping all columns of all candidate tables to the columns of the target table. While these two works address a different type of query, they are related to our problem in their ambition to create consistent results from a large number of candidate tables. Since their consolidation method merges tuples only duplicate tuples, it may even preserve attribute variants, as we require for our method. However, this effect is not planned for, and alternatives are not made explicit, but may be contained in the single, fused table as artifacts. We, on the other hand, aim at providing alternative, but intrinsically consistent top-k augmentations for a given set of entities.

2.6.6 Result Diversification

Diversity in search results is often studied in Web search and recommender systems research. A good overview is given in (Drosou and Pitoura, 2010). On a high level, they classify result diversification methods can be grouped into methods based on result *content*, i.e., item similarity, or based on *novelty*, i.e, by preferring content not seen before, and finally methods based on *coverage*, which in their context means preferring results that cover many categories of a predefined taxonomy. They also proposed two general categories of diversification algorithms, namely those based on greedy heuristics and those based on interchange or swap heuristics. The greedy strategies build up the result set iteratively by using a *item-set distance* to pick datasets with maximum distance to the already picked ones. Contrarily, the interchange strategies start with an initial solution only based on relevance, and then iteratively swap the items that contribute the least to the diversity of the set with the most relevant item that was not in the initial solution. For our approach, we designed and evaluated both types of heuristics for the selection phase of our MultiGreedy and Genetic approaches.

A general framework of axioms for result diversification functions has been proposed in (Gollapudi and Sharma, 2009). The authors give several problem formulations, i.e., general diversification functions based on abstract distance and weight functions, that allow a trade-off between novelty and relevance. In (Agrawal et al., 2009), the authors make the case for using diversification to tackle the problem of unclear user intent in search queries. They argue that keyword queries are naturally ambiguous, e.g., a query "apple" may to refer to the fruit, the computer company, or the record label. Therefore, presenting only the most relevant interpretation of the keywords may lead to unsatisfactory answers if the user intent was a different. They propose both a new algorithm based on existing document taxonomies, and also propose a new set of intent-aware metrics for use in evaluation. We make a very similar argument with regard to entity augmentation queries, which are specified by keywords as well, to motivate our diversity requirement in Section 2.1.

In (Drosou and Pitoura, 2012), the concept of DisC diversity for choosing diverse subsets from a set of structured query results is introduced. A subset is said to be DisC diverse, if each object in the query results is represented by a similar object in the subset, while those representatives are chosen to be as dissimilar as possible. Similarity between objects is measured using a radius around them. This is used to enable an intriguing operator called *zoom*, which allows to adapt the diversified result's granularity by varying this similarity radius. A lower radius leads to a larger number of more fine-granular representatives being chosen, while a

larger radius reduced the number of representatives for more of a bird's eye view of the query result. This notion of similarity and the operator would lend itself well for adaptation to our top-k set covering method. Covers created from similar datasets could be "hidden" behind their representatives to give the user a compact view of the possible directions of integrations, e.g., possible query interpretations. The zoom operator could be used locally to inspect specific representatives, and chose a final integration result based on closer inspection.

Recently, diversification has also been identified as a relevant problem for Web table search, i.e., keyword-based lookup of relevant tables (Tam et al., 2015). The authors claim that, because of copying effects on the Web (Li et al., 2013a; Dalvi et al., 2012b), returning only the most relevant tables for a given query will lead to many tables of equal structure and content in the top results, and thus to an unsatisfactory user experience. Similarly to our work, they employ Web table similarity measures and combine them with relevance scores, and use a greedy algorithm to create a top-k list of tables that is optimal with respect a combined measure of relevance and inverse similarity, i.e., diversity. We follow their argument regarding diversity and use inverse Web table similarity as a measure for diversity as well, though our solution has additional constraints such as diversity and minimality, due to combining data sources to covers instead of only presenting a flat list.

As mentioned, result diversification has many applications, e.g., it is crucial in recommender systems to counterbalance personalization (Ziegler et al., 2005), i.e, to create recommendations that align with the complete spectrum of a user's interests, not only the dominant ones of the user's profile. Furthermore, (McNee et al., 2006) make the case that users of recommendation systems are more satisfied with the recommendations if they also contain items that they would never have picked themselves, i.e., if the results enable *serendipity*. Similar arguments have been made for entity search (Bordino et al., 2013).

2.7 SUMMARY AND DISCUSSION

In this chapter, we analyzed the state of the art in processing entity augmentation queries, and identified several challenges so far unanswered in related work: *Attribute Variations*, *Unclear User Intent*, *Trust & Lineage*, *Exploratory Search* and *Error Tolerance*. We derived the two requirements: creating *consistent* solutions from a *minimal* number of sources, and answering augmentation queries with *diversified top-k* results. We presented an extended version of the Set Cover problem, called *Top-k Consistent Set Covering*, onto which we could map our requirements. We introduced a core framework for solving top-k consistent set covering, which includes a basic greedy algorithm modeled on the corresponding Set Cover algorithm. Then, we proposed two more advanced algorithms. The first is an extension of the Greedy algorithm that explores a larger part of the search space, and adds a selection phase to filter the larger number of solutions that are created. For the final approach, we mapped the problem into the domain of evolutionary algorithms, to reach a more sophisticated genetic algorithm. We then introduced REA, our system for Web table-based entity augmentation, which implements our novel, set cover-based algorithms. Finally, we presented an extensive evaluation of the proposed algorithms and the REA system, with respect to coverage, precision, and result quality measures such as minimality, consistency and diversity amongst others. Our experiments show

that our genetic set covering-based approach improves both consistency and minimality of the results significantly, without loss of precision or coverage, and while producing a diverse set of results for the user to choose from.

Having summarized the chapter, we will now review the challenges identified initially, and how our approach contributes to their solution. First, we discussed the problem of ubiquitous *attribute variations*, even for seemingly trivial attributes, such the "gdp" of a country. However, the real-world concept is complex, with many variants such as nominal GDP or GDP based on purchase power parity, different years of validity and different sources. While most related work assumes a single-truth answer is sufficient, our approach returns multiple, diversified solutions. Through the use of consistent top-k set covering, our approach ensures that individual solutions are created from similar sources, i.e., sources expressing the same attribute variation. The second challenge, *unclear user intent*, is tackled through result set diversification. Our system ensures, through fuzziness in candidate source retrieval, and algorithms that incorporate diversification strategies, that not only one interpretation of the keyword query is covered in the list of results. The third challenge is *trust and lineage*, which arises because professional users will have to understand and manually verify the augmentation provided, which is impeded by existing by-entity fusion models. It is tackled in our approach through optimizing for minimality of the created covers. By mapping our entity augmentation to the Set Cover problem and suitably constructing our algorithms, we were able to generate solutions using far fewer distinct sources, without sacrificing full coverage or relevance, and while improving consistency at the same time. This means the user will have to verify fewer sources, that are more likely to be similar, and thus consistent with each other. The fourth challenge, *exploratory search*, is related to the problem of unclear user intent, and is also answered through diversified top-k solutions. Specifically, the fuzziness inherent in Web table matching and scoring, combined with the diversification objectives included with our algorithms, will lead to unexpected data sources being used in higher ranks of the result list. Finally, *error tolerance* is addressed through top-k result presentation. As shown in our evaluation, the first solution according the system's ranking scheme is not necessarily the best solution, for example because of errors in Web table schema matching. Our evaluation showed, that for a significant share of queries, the highest-ranked solution was not the best one found. This implies the user would have had to work with a suboptimal answer if a single result system had been used.

To conclude, our method of top-k consistent set covering, by promoting consistency, minimality and diversity, without sacrificing coverage or precision, successfully tackles the entity augmentation challenges we identified. Our minimality requirement creates solutions that are easier to understand and verify, while explicitly modeling source similarity gives us a way to not only create more consistent results, but also to create useful alternative solutions. This top-k paradigm alleviates many of the weaknesses stemming from the uncertainty of Web tables as data sources, ambiguous queries, and errors from fully automatic data search- and matching techniques. At the same time, it also enables new use cases, such as exploratory search.

OPEN-WORLD SQL QUERIES

In the previous chapter, we introduced top-k entity augmentation as a novel paradigm for ad-hoc Web data integration. We demonstrated how it improves existing augmentation methods by managing the uncertainty of Web tables as data sources, the ambiguity in user queries, and errors resulting from fully automatic data search- and matching techniques. However, so far we only studied this novel form of automated data search and integration in an isolated context. All scenarios we studied dealt with only one single table, or one set of entities, that was the sole focus of the user's information need. Likewise, top-k entity augmentation queries were considered as single table, single attribute queries $Q_{EA}(E, a_+, k)$, with no usage context or constraints. However, it is natural to assume that ad-hoc data integration will be most useful in analytical scenarios, in which the user works with complex databases, and the augmentation query is only one step in a chain of analytical operations. In such a scenario, the data already resides in a DBMS, most likely a traditional relational system, or one of its recent Hadoop-based counterparts. The user would also not be interested in the augmentation per-se, but would require the integration of additional attributes in the context of a larger analytical task. In the next section, we will therefore discuss problems that arise when entity augmentation is utilized in analytical scenarios involving traditional database management systems. From the identified challenges, we will derive the need for closer integration of top-k EA systems and RDBMS, and derive requirements for a hybrid system in Section 3.2. In Section 3.3, we will introduce the system architecture of our DrillBeyond system, and describe its core, the DrillBeyond plan operator. We will also discuss peculiarities of hybrid augmentation/relational query processing, and introduce Drillbeyond's ways of dealing with them. We will show that naïve processing of those hybrid queries leads to substantial performance overhead. Therefore, in Section 3.4, we will introduce a series of optimization techniques that minimize the performance impact of integrated entity augmentation queries in DrillBeyond. In Section 3.5, we will present an extensive evaluation of the DrillBeyond system using TPC-H queries extended to include augmented attributes. We will show that our system fully integrates the two query paradigms with minimal overhead, and frequently even improves runtime and precision compared to using separate systems. Finally, we will survey related work in Section 3.6 and then summarize and discuss our findings with respect to the requirements in Section 3.7.

Parts of the material presented in this chapter have already been published in (Eberius et al., 2012) and (Eberius et al., 2015c).

3.1 MOTIVATION AND CHALLENGES

In this section, we will study an exemplary data analytics scenario and identify challenges that arise with the integration of an entity augmentation system to answer ad-hoc information needs. Consider a user in a traditional warehousing environment, analyzing sales totals by country. Let the current analysis task be illustrated through the SQL query shown in Figure 3.1a, which is defined on a TPC-H[1] data warehouse.

As a next step, the user may want to add external context to gain insight into the aggregated sales numbers. For example, it may be beneficial to introduce some global economic context into the analysis of the sales data. More specifically, if the user wants to limit the

[1] http://www.tpc.org/tpch/

```
select
    n_name,
    avg(o_totalprice)
from nation, customer, orders
where
    n_nationkey=c_nationkey
    and c_custkey=o_custkey
group by n_name
```

(a) Initial query on local data

```
select
    nation.creditRating,
    avg(o_totalprice)
from nation, customer, orders
where
    n_nationkey=c_nationkey
    and c_custkey=o_custkey
    and nation.gdp > 10.0
group by nation.creditRating
```

(b) Extended query with ad-hoc attributes

Figure 3.1: Exemplary SQL queries

scope of the analysis to those countries with a particular property, e.g., a high gross domestic product (GDP), it may be necessary to search for and choose between multiple sources, such as the World Bank Data[2] or UN Data[3]. As discussed in Chapter 2, the user will also have to deal attribute variations, e.g., different methods of calculating the GDP, such as "GDP ppp" or "nominal GDP" versus "real GDP". In yet another step, the user may want to aggregate the sales with respect to a different dimension, for example according to the credit rating of the country they were made in. However, just as with "GDP", the attribute "credit rating" is not defined in the local warehouse schema, making another search and integration process necessary. This attribute can again be obtained from several sources, i.e., different rating agencies, whose ratings may vary considerably. The user may not necessarily know initially which source is optimal in the given situation, and may thus even need to perform the integration process multiple times.

Each of those integration steps may be quite costly in terms of user effort, as classic data warehouses offer little support for ad-hoc integration. On the contrary, usually there are intricate ETL processes and policies that have to adhered to, before new data can be integrated into the warehouse schema (Cohen et al., 2009). Following an intuition that comes up within an analysis process is therefore quite cumbersome.

In summary, to fulfill these kinds of ad-hoc information needs in classic warehouse settings, the analyst has to perform a multitude of complex steps. These include manually searching for datasets that contain the information, moving them through an ETL process, reformulating the query to contain the predicates, aggregations and additional joins with the imported tables, and finally rerunning the query. Additionally, if the user is not content with the results using a particular set of sources, it may be necessary to go through the search and integration process iteratively until suitable data sources have been found. Ideally however, the user would be able to tackle this ad-hoc integration scenario in the same way he would tackle the analysis of the purely-local data: by using a single, declarative query. Instead of performing going through an elaborate process, the analyst should be able to pose the a new query to the system, as if all potentially available data would already be integrated into the system. This is illustrated in the extended SQL query shown in Figure 3.1b. Note, that the highlighted attributes "GDP" and "creditRating" are not defined in the TPC-H schema, therefore the query is not immediately

[2] data.worldbank.org
[3] http://data.un.org/

processable in a traditional relational system with a closed schema. However, the difficulties of retrieving and choosing from the multitude of sources and attribute variations we discussed to in the example above correspond to the problems that we tackled in Chapter 2. Therefore, a possible way to approach the above query would be to export the "nation" relation, and feed it into a stand-alone augmentation system, such as the *REA* system introduced in the previous chapter. However, part of our solution to these challenges is to introduce the top-k entity augmentation paradigm, in which the system produces several alternative solutions to the augmentation query for the user to choose from. This process would therefore result in a top-k list of possible augmentations for the exported table. Since a standard DBMS can not process the query based on a multi-valued augmentation, the user would have to choose one of the augmentations while in the independent context of the augmentation system, and then re-import the selected augmentation into the DBMS. Again, iterations of this process may be necessary if the initial result is not satisfactory. For example, if the user would like to compare the query result based on the different credit ratings of the several rating agencies, it would be necessary to repeatedly execute the original query using several imported augmentations tables.

To summarize, we introduced how top-k entity augmentation can be applied in analytics scenarios to fulfill ad-hoc information needs. However, we also argued that in complex analytical scenarios, using a standalone entity augmentation system to answer ad-hoc information needs has several deficiencies. In the remainder of this section, we will discuss these problems individually, and then derive requirements for a solution in Section 3.2.

Context-Switching First, there is a cost associated with context-switching, both with respect to *user effort*, but also with respect to *data locality*. The user would be required to move the data that is to be augmented into the specialized data search and integration system, such as REA presented in Chapter 2, inspect and verify the solution in this context, and then move the data back into the actual analytics system. On the one hand, this introduces a considerable overhead into the analysts workflow, requiring additional effort that may even discourage from performing certain ad-hoc exploratory queries at all. On the other hand, it also introduces physical overhead of moving the data between systems. This overhead may negligible if a small dimension table, such as the "Nation" table in the example, is to be augmented. For larger tables, however, data transfer times can be become significant, and further impede an interactive analytics workflow.

Incompatible Query Model A second challenge when introducing top-k entity augmentation into a traditional analytics workflow is the mismatch in query models. Augmentation systems such as REA produce top-k results, while other parts of the environment, such as DBMS, work with exact, single results. In this respect, top-k augmentation systems are more similar to information retrieval systems that handle the uncertainty of their results by producing a list of possible results. A traditional DBMS, on the other hand, is not natively prepared to handle uncertainty or multi-variant data, although we will discuss exceptions in Section 3.6. This gap needs to be bridged in order to enable effective combination of the two system types.

Loss of Context Information Third, by using a separate system for augmentation queries, the broader context associated with the analytical task is not taken into account. A generic en-

tity augmentation system uses only the set of entities and the augmentation attribute as input to guide its data search and integration process. However, the query context may contain valuable hints that can improve the augmentation system's performance or precision, if the systems were able to exploit them. For example, in an SQL query, other tables that are joined with the augmented table may provide useful context for the data source retrieval and matching process in the augmentation system. Similarly, predicates used in the original query add semantics that can be used to improve the precision of downstream augmentation system. In an optimal combination of DBMS and entity augmentation system, such context information incurring in one system would be utilized in the other.

Unused Optimization Potential Finally, by using separate systems, query *optimization potential* is wasted. For example, DBMS use cardinalities and estimated selectivities to choose an optimal join order for a given query. If a manual entity augmentation has to be performed before the query can be executed, then the cardinality of the augmented relation, or the selectivity of predicates on the augmented attribute can not be exploited in this optimization process. Depending on the exact circumstances, it can be beneficial to intermingle the normal DBMS query processing and the augmentation query processing to achieve optimal performance.

Having introduced the challenges of integrating top-k entity augmentation with DBMS-based analytical workflows, we will derive requirements for a solutions to these problems in Section 3.2.

3.2 REQUIREMENTS

From the challenges we identified in Section 3.1, it is easily recognizable that a closer integration of the two system types DBMS and EAS (Entity Augmentation System) is necessary. In this section, we will introduce requirements for a hybrid system that is able to close this gap.

DBMS-integrated Entity Augmentation In the previous section, we discussed how the lack of integration between DBMS and EAS systems leads to an increased user effort for situational one-of analysis queries. This effort could be reduced if the DBMS would directly support looking up and integrating Web data sources as part of its query processing, and allow the specification of such queries declaratively in SQL. However, there are several differences to bridge. First, the two system types differ in the type of data they manage: DBMS deal with structured and cohesive databases, while EAS deal primarily with large heterogeneous corpora of Web data sources. Further, DBMS work with fully specified queries in a structured language, while EAS, lacking a defined schema, accept keyword queries.

We therefore require a design that blurs the line between these classes of systems in all three aspects mentioned above: type of data managed, query language used, and nature of the query result. The resulting hybrid system should be able process mixed SQL/EA queries, which we will call *Open World SQL queries*. In these queries, the user may reference arbitrary additional attributes not defined in the schema.

Figure 3.3 will serve as our running example for such a query. The system will associate values of these additional attributes to instances at query processing time, avoiding an explicit data

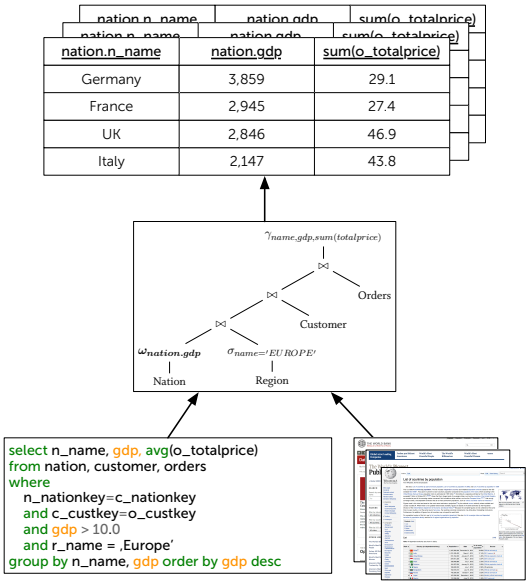


Figure 3.2: RDBMS-integrated top-k augmentations using Web tables

retrieval and integration step. This is achieved by executing top-k entity augmentation queries at runtime, which are integrated as a new type of subquery of regular relational queries. Since a top-k augmentation query will return multiple augmentations as described in Section 2.3.1, an Open World SQL query will, instead of returning a definitive query answer, return multiple alternative query results as well. Figure 3.2 gives an intuitive overview of our goal. It illustrates how an Open World query is processed by integrating entity augmentation into query processing, producing k alternative SQL query results.

In conclusion, the system should produce structured results of exactly the form specified by the user query, just as a regular DBMS would, but also presents several possible versions of the result, similarly to an information retrieval system. However, producing multiple alternative SQL results for a single query has performance implications, which we will discuss next.

Efficient Multi-result Query Processing A naïve approach to bridging the different query models of DBMS and top-k EA systems would be to process the EA query, and then process the SQL query multiple times. For example, if k possible augmentations are requested, the runtime of the SQL query is increased by this factor k. This is not acceptable, since in many Open World SQL queries, the majority of the processing time will still be spent processing local data, which does not change between runs of the query. For example, consider again the example Open World SQL depicted in Figure 3.3. Here, a large part of the work consists of local joins between the relations `Customer` and `Order`, and aggregation of the local attribute `o_totalprice`. When processing this query multiple times based on different augmentations, only the set of nations that pass the predicate on `gdp`, as well as the order of result tuples would change, but not the aggregates for the individual nations. Consequently, the hybrid system should process Open World SQL queries in a way that minimizes duplicate work between query variant executions.

```
select n_name, gdp, avg(o_totalprice)
from nation, customer, orders
where
    n_nationkey=c_nationkey
    andc_custkey=o_custkey
    and gdp > 10.0
group by n_name, gdp
order by gdp desc
```

Figure 3.3: Running Example: Open World SQL query with added attribute highlighted

Open World Query Planning The third requirement arises from the fact that properties of data sources used in augmentation are not fully known at plan-time. For instance, estimating the selectivity of a predicate over an augmentation attribute is not easy, as the set of data sources that will be used is not known at planning-time. The same holds for determining the open attributes' metadata, such as the data type, since we do not require the user to specify it in the query. Therefore, the system should be able to plan queries even if some attributes are only fully known at run-time.

With these requirements established, in the course of this chapter, we will make the following contributions:

- We introduce the *DrillBeyond* system and its entity augmentation operator, which implements top-k entity augmentation based on Web tables as part of regular relational query processing.
- We discuss issues regarding placement of this novel operator with respect to runtime and answer quality, and propose a cost model as well as plan- and run-time optimization rules.
- We detail how to efficiently process a query multiple times based on different entity augmentations.
- We explore how the entity augmentation process itself can be improved through information available in the context of an SQL query.
- We introduce a practical implementation of the operator and our optimizations in PostgreSQL, which allows us to meaningfully evaluate the operator on standard test databases and fully-featured SQL queries, both regarding runtime and influence of the techniques discussed above.

3.3 THE DRILLBEYOND SYSTEM

To solve the challenges identified in Section 3.1 and enable entity augmentation queries as part of relational query processing, we designed the *DrillBeyond* system. It is an RDBMS/EAS hybrid, that embeds entity augmentation sub-queries into standard RDBMS query processing.

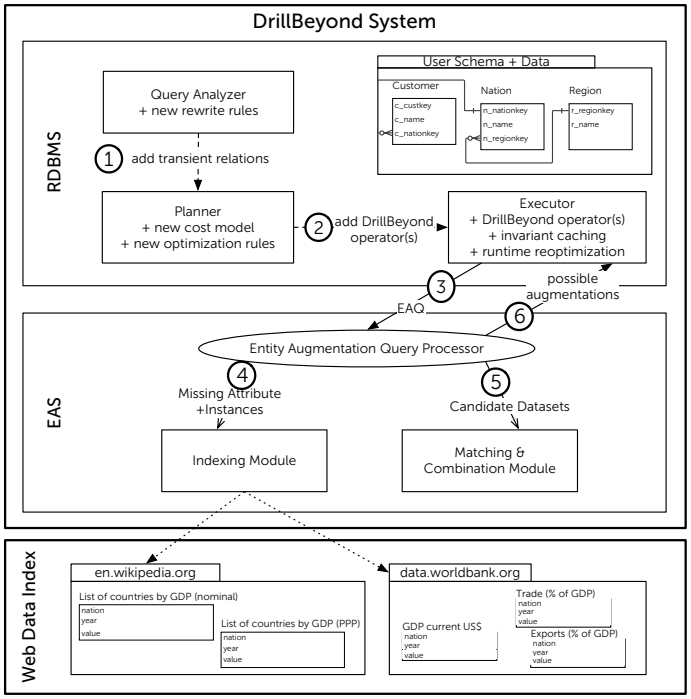


Figure 3.4: System architecture and high level control flow

The next sections will detail the required changes to the RDBMS architecture to realize this mixed query processing.

3.3.1 System Architecture

Processing open world SQL queries requires a top-k entity augmentation system, including a data source management system, as well as modifications to three core RDBMS components: the analyzer, the planner and the executor. Figure 3.4 gives an overview of the modified and the novel components, and further includes a high level description of the changes in control flow. The core augmentation functionality is introduced through the new *DrillBeyond plan operator*, which will be discuss in detail in Section 3.3.2. In the following, we will first give a general overview of all the novel or modified DBMS components in DrillBeyond.

Data Source Management System A standard RDBMS is tailored to manage a relatively small set of relations that form a coherent schema. An EAS, on the other hand, manages a large corpus of heterogeneous individual Web data sources. We aim at enabling one system to process both kinds of data. However, a typical Web data corpus, such as the DWTC (Eberius et al., 2015a), is independent of the application specific databases managed by an RDBMS. Since it is not domain-specific, it can be collected and managed independently, for example in a *data curation system*, at type of system we discuss in detail in Chapter 4.

Therefore, our design does not put the corpus under direct management of the DBMS. Rather, we assume that organizations utilizing such corpora will operate an additional data

curation system or data lake for managing heterogeneous data. An alternative would be to acquire access to such collections from external suppliers, e.g., from Open Data providers. The DrillBeyond system does not make assumptions regarding the nature of the data sources and system that they are managed by. A generic system that exposes an interface for keyword-based dataset search is sufficient. For example, when using augmentation methods based on Web tables, such as REA which we introduced in Chapter 2, an industry-standard document index server such as Solr[4] or ElasticSearch[5] is sufficient. The necessary source selection, matching and integration operations are performed in the integrated entity augmentation system, which we discuss next.

Entity Augmentation System This component implements the actual top-k entity augmentation processing inside the DBMS. It interfaces with the data source management system to retrieve Web data sources, and with the core RDBMS components to provide augmentation services to the executor and the planner. In Chapter 2, we gave the following definition for an entity augmentation query:

$$Q_{EA}(a_+, E, k) = [c_1,\ ...,\ c_k \mid c_i \subset D \wedge \mathrm{cov}(c_i) = E] \tag{3.1}$$

This definition closely corresponds to the internal interface of the augmentation subsystem, which is used by the DrillBeyond operator during relational query processing. However, DrillBeyond extends the generic augmentation query definition above using *query context hints* H, which are extracted from the respective outer SQL query. These hints are used to guide the search and the ranking of Web data sources. For example, if the outer SQL query includes a numeric predicate on the augmentation attribute, this fact can be used as a query hint by instructing the augmentation system to only retrieve data sources that provide numeric values. We discuss the exact nature and usage of these context hints in Section 3.3.5. The complete interface used by the executor is therefore $Q_{EA}(a_+, E, k, H)$. Having introduced the novel components necessary for entity augmentation in the DrillBeyond system, we will now give an overview of the modifications to existing RDBMS core components.

Query Analyzer The first step in DrillBeyond query processing is triggered by the query analyzer, which maps tokens in the SQL query string to objects in the database's metadata catalog. Unrecognized tokens, such as *gdp* in the running example (Figure 3.3), lead to an error in a typical RDBMS. In the DrillBeyond system, we take a minimally invasive approach: we introduce transient metadata for the duration of the query, so that the regular analysis can continue. The query is then rewritten to include an additional join with a transient relation, effectively introducing a source for the missing attribute into the query processing, and also paving the way for the DrillBeyond operator to be placed by the regular join order planning mechanisms of the DBMS.

The analyzer is also responsible for determining the type of all expressions in the query. In the case of augmentation attributes, which are not represented with a type in the database catalog, we include a type inference mechanism. It first tries to infer the data type syntactically, by considering filter and join predicates the attribute is used in, and comparing with the types

[4] http://lucene.apache.org/solr/
[5] https://www.elastic.co/

of regular attributes and constants in these expressions. If syntactic inference is not possible, DrillBeyond infers a type statistically. It uses the augmentation system to determine the most common data type occurring for attributes named a_+ in the dataset corpus using a fast probe query.

Having created the necessary query metadata, the analyzed statement can be passed on to the modified query planner.

Query Planner In DrillBeyond, the query planer has several new tasks to perform compared to a regular RDBMS. First, it needs to place the DrillBeyond operator in the query plan. This placement is crucial to the execution time of the query, but as we will show in Section 3.3.4, also influences the augmentation quality. Furthermore, while regular query plans are created to be optimal for a single execution, multi-solution processing requires plans that minimize the overhead of creating multiple result variants based on top-k augmentations. This is impeded by the lack of plan-time knowledge about the data sources, requiring plan adaption. We detail our approach to these problems in Section 3.4.

Executor The executor is modified to repeatedly execute the planned operator trees, creating the top-k query result. In each iteration, it orders the DrillBeyond operators to augment incoming tuples with values from a different augmentation. It further tags the SQL results produced with a distinct augmentation id, by adding a column carrying this id to each finished result. This allows external tools using the top-k SQL result to distinguish between the tuples belonging to alternative results. The majority of new functionality, however, is part of the DrillBeyond operator itself, which we will detail in the next section.

3.3.2 The DrillBeyond Operator

This section details the design of the DrillBeyond operator, and discusses special characteristics of hybrid relational/augmentation query processing that motivated the design. We will reference and explain all parts of this pseudo-code description shown in Algorithm 4 throughout the following sections.

In its basic form, the DrillBeyond plan operator, denoted ω, is designed to resemble a join operator, which facilitates integration with the existing system architecture. Specifically, it acts like an outer join: it adds new attributes to its input tuples based on join keys, but will not filter original tuples if no partner is found. Instead, it adds null values if the augmentation system can not produce a value. In this way, the part of the query operating on local data can still be processed. However, in contrast to a regular join, only one of the joined tables is known at plan-time, while the other table, as well as the join keys, are decided at query processing time. These run-time decisions are made by the entity augmentation system based on the input tuples of the operator. Specifically, the operator extracts distinct combinations of textual attributes from input tuples, as these are used to functionally determine the values of the augmented attribute. In the example query shown in Figure 3.3, the set of `Nation` tuples that reach the operator determine the input to the entity augmentation subquery, and therefore determine the Web data sources that are are retrieved.

We will now consider the implementation specifics of the operator. Algorithm 4 shows

Algorithm 4 DrillBeyond operator

```
function INIT
    state ← 'collecting'
    tuplestore ← ∅
    augMap ← HashMap()
    n ← 0                                   ▷ Current Iteration, runs from 0 to k − 1

function NEXT
    if state = 'collecting' then
        COLLECT()
        AUGMENT()
        state ← 'projecting'
    return PROJECT()

function COLLECT
    while true do                           ▷ Retrieve all tuples
        t ← NEXT(childPlan)
        if t = NULL then
            break
        tuplestore ← t
        augKey ← TEXTATTRS(t)
        if augKey ∉ augMap then
            augMap[augKey] ← ∅

function AUGMENT
    augReq ← (∀k ∈ augMap | augMap[k] = ∅)
    for all augKey, [augValues...] ∈ SEND(augReq) do
        augMap[augKey] ← [augValues...]

function PROJECT
    t ← NEXT(tuplestore)
    if t = NULL then return NULL
    augKey ← TEXTATTRS(t)
    t[a+] = augMap[augKey][n]
    return t

function RESCAN
    state ← 'collecting'
    tuplestore ← ∅

function NEXTVARIANT
    RESCAN(tuplestore)
    n ← n + 1
```

the specifics of the state kept in the operator and the implementations of its iterator interface and helper functions. DrillBeyond uses a traditional row-based iterating executor. The conventional interface functions *Init()*, *Next()* and *ReScan()*, as well as the novel *NextVariant()*

function, are called by the DBMS during regular query processing. The other functions shown in Algorithm 4 are used internally by the operator.

The *Init()* function, which is called by the RDBMS executor before processing the query the first time, initializes operator state. This includes a tuple store for materializing the lower operator's output, a hash table mapping local textual attribute values to augmented values called $augMap$, and two variables $state$ and n, determining the behavior of the operator when *Next()* is called.

The *Next()* function is called by the executor and produces augmented tuples. This is done in three phases: *Collect()*, *Augment()* and *Project()*. On the first call to *Next()*, since no augmented values are available, the first two phases are triggered. In the *Collect()* phase, the operator pulls and stores all tuples that the lower plan operator can produce, making DrillBeyond a blocking operator. The reasons for blocking are discussed in Section 3.3.3. In this phase, the operator also stores the textual attributes originating from the augmented relation and its context in a hash table, to obtain all distinct combinations of textual values in the input tuples. The concept of augmentation context is discussed in Section 3.3.4.

In the *Augment()* phase, all entries in the augmentation map that do not yet have values associated with them are passed to augmentation system as one augmentation context. After successfully retrieving values for all collected tuples, the operator is put into the *projecting* state, and produces the first output tuple. Output tuples are produced in the *Project()* function by replaying the stored tuples and filling the augmentation attribute by looking up values in the hash table.

The *ReScan()* function is called by the DBMS executor when subtrees have to be re-executed, e.g., in dependent subqueries or below a nested loop join. Here, the operator empties its tuplestore and changes state to collect new input, but keeps its augmentation hash table, to prevent expensive re-augmentation for values that have already been seen. Finally, *NextVariant()* is an interface extension not seen in typical RDBMS operators, which is necessary for producing the multi-variant query results as discussed in Section 3.2. When called, the operator's tuplestore is prepared for another iteration over the stored tuples using *ReScan()*, and the iteration counter n is incremented. This makes sure that in a new execution of the query plan, operators below the DrillBeyond operator are not called again. Instead their materialized output will be replayed, and augmented with the next augmentation variant in the *Project()* function.

The functions *Init()* and *Next()* are part of the traditional iterator interface used in RDBMS and are called by the executor in regular query processing. The *NextVariant()* function is different however, and requires novel functionality in the executor. Specifically, when the top operator of the plan returns null, the executor usually assumes that all data has been sent and stops the processing. DrillBeyond however, keeps a global iteration count n, running from 1 to k, to track the number of produced alternative results. In case the plan has finished, n is increased, *NextVariant()* is called on the DrillBeyond operator, and then the whole plan is restarted. In case there is more than one augmentation attribute, and thus more than one DrillBeyond operator, the system produces a crossproduct of alternative results. This is achieved by systematically calling *NextVariant()* only one operator in each iteration, to iteratively produce all combinations of possible results for all augmented attributes.

Having introduced the basic operator functionality, and the way it is called by the DBMS executor, we will now discuss more advanced questions in the next subsections: why is the

operator blocking, how does the placement in the query plan influence augmentation performance and quality, what is the operators runtime cost and how can it be optimized?

3.3.3 Augmentation Granularity

A naïve entity augmentation operator would work *tuple-at-a-time*, i.e., hand each tuple to the augmentation system as it receives it. This way, it would be most compatible to the iterator-based query processing used in most traditional RDBMS operators, and would not need to block. However, augmenting each tuple on its own implies looking up and matching Web data sources for each tuple individually. Consider the simplified augmentation example shown in Figure 3.5, where the table to be augmented is on top, and the available Web data sources at the bottom. With the *tuple-at-a-time* style, the augmentation system may choose ds_1, ds_2 and ds_4 for the USA, Russia and UK tuples respectively. These sources match each individual tuple best, and the individual tuples are what the augmentation system can process in this case.

Still, we can see that the augmentation system can not perform optimally with regard to the query as a whole, as it is not provided with the overall query context. While the chosen sources are the best fitting for each individual tuple, they do not form a consistent answer together, as the units of currency do not match. If the augmentation system is instead provided with the set the complete set of tuples as the input for one augmentation query, the more consistent solution comprised of ds_1 and ds_3 can be constructed, even though the individual entity matches are slightly worse. This is a similar argument to those we made Chapter 2 about result consistency. Extending single tuples individually would correspond to the *By-Entity Fusion* augmentation strategy as introduced in Section 2.3.1, and would lead results with similar deficiencies as discussed there. In Section 2.5.5, we showed that our approach to entity augmentation leads to higher quality results, but requires all entities to be queried in a group. Otherwise, the consistency of the source selection can not be ensured.

We conclude that for reasons of result quality, the DrillBeyond operator needs to be a blocking operator. This means it consumes tuples from underlying operators until they are exhausted, then hands them over to the augmentation system, and produces the first result tuple only when it returns. Referring back to Algorithm 4, this blocking behavior is realized in the state "collecting" in function *Next()*.

3.3.4 Context-dependent Results

Having established the DrillBeyond operator as blocking, we will now consider the question of where to place it in a query plan. We noted in Section 3.3.3 that the augmentation result is dependent on the set of input tuples to the augmentation system. However, it is not only affected by the augmentation granularity, but also the position of the operator in the graph. For example, consider again the query shown in Figure 3.3, and two possible query plans shown in Figure 3.6, where ω depicts the DrillBeyond operator. The set of `Nation`-tuples that reach the operator determines the input to the entity augmentation subquery, and therefore determines the Web data sources that are are retrieved. In variant (a), the DrillBeyond operator is executed directly after the scan of the `Nation` table, while in (b) it is executed after the local join and filter with the `Region` table has been performed. In the first case, the augmentation system

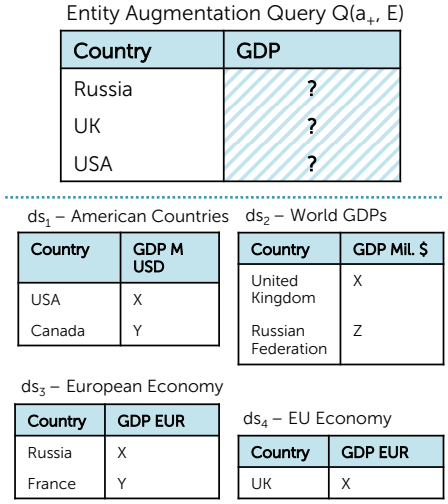


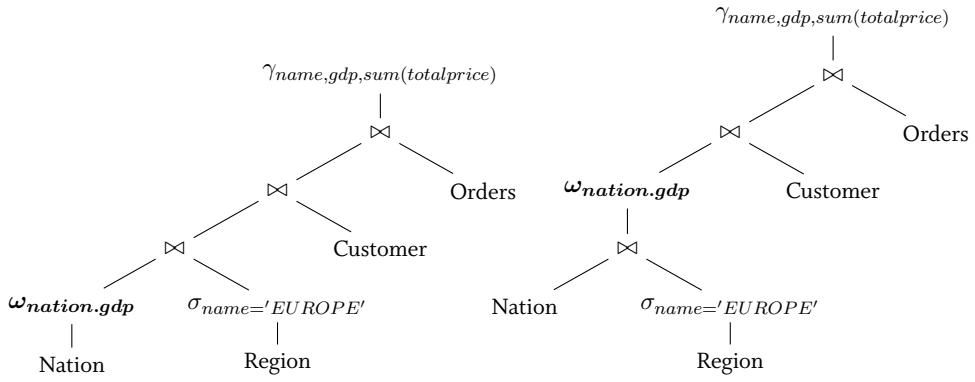
Figure 3.5: Example augmentation problem

will retrieve, match, rank and combine datasets for all countries in the local database. Likely, these will be datasets about the countries of the world. In the second case, the local join will remove tuples about non-European nations, so the results of the entity augmentation will be more likely be based on data sources specifically about Europe. Furthermore, completing the join with the `Region` table does not only limit the scope of the query, it also adds context to each tuple by adding `Region`'s attributes. So even if the join would not act as a filter limiting the number of tuples, adding information about the region name will improve the accuracy of the augmentation system. For example, while augmenting a set of *City* tuples may be hard and error-prone because of the ambiguity of common city names such as "Springfield", augmenting after a join with a *State* will be a more realistic task for the augmentation system. We have identified the following property of the ω operator:

Definition 3.1 (Selection Dependency)**.** *The DrillBeyond operator is not commutative with respect to selection in the general case. When augmenting relation R with attribute a_+ and selecting with predicate p on R, then $\omega_{R,a_+}(\sigma_p(R)) \neq \sigma_p(\omega_{R,a_+}(R))$.*

Note that the augmentation system uses only the textual attributes $textAttr(R)$ for matching with Web data sources. Therefore, in the special case that the set of distinct textual attribute values is invariant under predicate p, the augmentation results are also invariant under selection with p. For example, if a selection $\sigma_p(Nation)$ returns at least one tuple for each `Nation` in its result set, then $\omega_{Nation,a_+}(\sigma_p(Nation)) = \sigma_p(\omega_{Nation,a_+}(Nation))$. This however, is not the general case, and can not be relied upon.

A similar dependency property holds for another form of query context, namely for the set of attributes of the input tuples of an operator ω. As mentioned, the key for searching matching Web datasets for R are its textual attributes $textAttr(R)$. For example, not projecting the attribute `n_name` of the TPC-H `Nation` relation, possibly because it is not part of the desired query result, will make augmentation impossible. No Web data source can be found if the natural key of the relation, the nation's name, is not part of ω's input.



(a) DrillBeyond placed before the join (b) DrillBeyond applied to the join result

Figure 3.6: Possible drillbeyond operator placements

Definition 3.2 (Projection Dependency)**.** *The DrillBeyond operator is commutative with respect to projections only if it includes all textual attributes of R. Formally, when augmenting relation R with attribute a_+, and projecting to a set of attributes $\mathcal{A}$ with $\mathcal{A} \cap textAttr(R) \neq textAttr(R)$, then $\omega_{R,a_+}(\pi_{\mathcal{A}}(R)) \neq \pi_{\mathcal{A}}(\omega_{R,a_+}(R))$.*

Given that the result of ω_{R,a_+} changes under projection and selection, we define the following placement rule that defines bounds on the placement of the operator.

Definition 3.3 (Placement Bounds)**.** *The DrillBeyond operator augmenting a relation R can only be placed with respect to the following conditions:*

1. *after all operators filtering R such as joins and selections*
2. *before any projection removing textual attributes of R*

In other words, the DrillBeyond operator ω is always applied to the minimum number of distinct combinations of textual column values in the tuples of R, as these determine the matching process and its result.

3.3.5 Pushing SQL query context

As mentioned in Section 3.1, context from the outer SQL query can be used to improve the accuracy and runtime of the inner entity augmentation query, when compared to isolated augmentation. We already discussed filter effects through joins and predicates in Section 3.3.4. These filters work implicitly to improve the augmentation quality by narrowing the scope of the augmentation operation, and do not require any changes to the augmentation system or its API. However, we can further improve the augmentation by explicitly pushing additional query knowledge to augmentation system. Specifically, we push two types of information: *type information* and *predicates on augmented attributes*.

Type Information: Though the user can specify a data type such as *text* or *double* for open attributes using SQL syntax, we do not expect those annotations to be provided. However, in many cases it is possible to infer the type of the open attributes from the surrounding query by applying methods of type inference to SQL. As mentioned in Section 3.3.1, we integrated a type inference mechanism that determines an open attribute's data type both from the query expressions it takes part in, as well as from the Web data corpus used. We can pass type information to the augmentation system, which in turn uses this type information to restrict the set of candidate data sources to those matching the open attribute's type.

Consider again the exemplary query shown in Figure 3.3. From the constraint on the *gdp* attribute it can be inferred that it must be of a numeric type. This allows the augmentation system both to reduce its runtime and increase precision by pruning non-numeric candidate sources.

Predicates on augmented attributes: In addition to the data type of open attributes, we can also push-down the predicates on open attributes themselves. This allows a similar, but more sophisticated, candidate pruning. We assume that users expect some filtering on the database instance level to happen when specifying a predicate, i.e., we assume that some domain-knowledge is encoded in the predicate. With the query shown in Figure 3.3, the user clearly intends to filter low *GDP* countries, and has given a relatively large integer number as the specific condition. Though the user query is given only with the very general keyword *GDP*, by also considering the predicate, the augmentation system can improve its data source ranking. In the example, all candidate datasets that give the GDP as a percentage or rank can be ranked lower, because those datasets will not discriminate the entities with respect to the predicate. In other words, given the predicate above, using a dataset that provides *GDP* rank values will lead to all entities being filtered, which is clearly not the user-intent. To improve the augmentation systems ranking, we can therefore check how well the data source's values fit to the predicate that will be applied, i.e., how well the data source discriminates the entities with respect to the predicate. Using the SQL context, we can thus add the following sub-score to the scoring functions already in place in our augmentation system REA (see Section 2.4.2). Given a predicate p and a candidate data source D:

$$s_p(p, D) = \begin{cases} 0 & \text{if } \neg applicable(p, D) \\ \frac{1}{|D|} \sum_{v \in D} 1.0 - |\log_{10} p - \log_{10} v| & \text{if numeric(p)} \\ 1.0 - (2 * |0.5 - sel(p, D)|) & \text{otherwise} \end{cases} \tag{3.2}$$

This equation also includes a special case for numeric predicates: we check whether the predicate value and the average of the data source's values are in the same order or magnitude. Again, the intent is measuring whether the data source is fit to evaluate the given predicate.

We will show the improvements in precision and runtime of the augmentation system when applying this score, as well as the data type filtering described above, in Section 3.5.

3.3.6 Cost Model & Initial Placement Strategy

The observations from the previous three sections might suggest that the DrillBeyond operator should be placed as late in the query plan as possible, to maximize the context knowledge avail-

able in the intermediate result. However, in addition to quality considerations, there are also performance considerations to be made. As mentioned, the operator is modeled to resemble a join from the perspective of the DBMS. We can therefore reuse the existing join optimization machinery to place the DrillBeyond operator. This however depends on a model for the operator runtime and the operator's output cardinality.

Output Cardinality In the most basic case, the operator produces exactly as many tuples as its input relation, as it just adds a single attribute to each tuple. However, we also must consider selectivity of possible predicates on augmented attributes, and the value of k, i.e., the number of alternative augmentations that are to be processed. Since at plan-time we do not assume any knowledge about which Web data sources will be used to augment an attribute, correctly estimating selectivity is almost impossible. A well-known solution to processing queries with unknown selectivities is run-time plan adaptation (Ives et al., 1999). We therefore initially use the DBMS' default selectivities for different types of predicates, and employ run-time optimizations to compensate when more information is available. These optimizations will be discussed in Section 3.4.

The variable k on the other hand is known at plan-time, and could be part of the cost model. However, with our execution strategy, the operator does not produce k tuples for each input tuple, but just a different one in each of k query executions. We therefore do not consider k at this point, but create optimal plans for single execution, and then later optimize their re-execution, as we will discuss in Section 3.4.

Cost Model The operators runtime depends on three components: The first part is incoming tuple processing, the second entity augmentation, and the third is projecting tuples with the augmented attribute (see Algorithm 4). Since we designed the operator as blocking, the first part consists of reading all tuples from lower plan nodes, storing them, and computing all distinct combinations of all textual attributes, which are needed by the entity augmentation system for matching. In the second phase, the distinct combinations are then submitted to the augmentation system. The third phase then consists of iterating all stored tuples, and projecting the new attribute based on the combination of textual attributes found in each tuple.

We can estimate the cost of the operator using a similar model as for a hash join, as phases one and three, hashing the child relation, and then probing it against the augmented hash table correspond to the phases of a hash join. Additionally, there is the cost of phase two, the actual augmentation, which depends not on the number of tuples, but on the number of distinct entries in the augmentation hash table. The processing cost per entry depends on the augmentation algorithms, and is therefore not easy to model in the context of a generic DBMS cost model. However, based on our runtime results for Web data-based entity augmentation from Chapter 2, we can assume that the cost per entry is in a different order of magnitude than the per-tuple cost of the relatively primitive database operations such as comparison or hashing. For our cost model we therefore assume an additional large constant factor $\mathcal{C}$ for the phase two component which, for evaluation purposes, we learn from previous executions of our augmentation system.

$$Cost_{Drb} = 2 * |subPlan| + \mathcal{C} * |distinct(subPlan)| \tag{3.3}$$

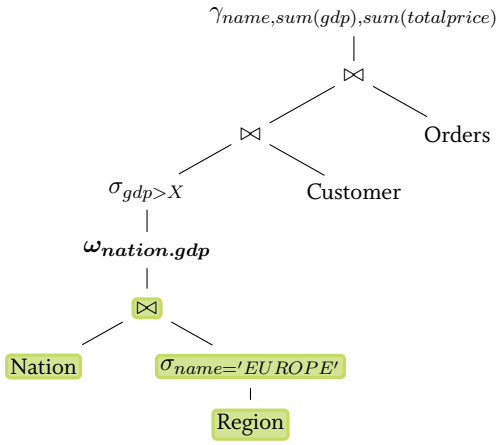


Figure 3.7: Plan Invariants

We use the cost model together with the bounds introduced in Section 3.3.4 to place the operator in the query plan. The bounds are enforced by pruning join orders in which the operator would be placed before all filters to the augmentation base relation are applied. Additionally, we ensure that all textual attributes necessary for augmentation are shielded from early projection by adding them as necessary join clauses to the operator during planing.

3.4 PROCESSING MULTI-RESULT QUERIES

So far, we have considered the DrillBeyond operator in a single query result setting. However, as discussed in Section 3.2, we aim at translating the top-k result returned by the augmentation system into a top-k SQL result.

The naïve method, given our operator, is to simply re-execute the query plan k times, and after each execution trigger the projection of a new augmentation result from the operator via the *NextVariant()* API depicted in Algorithm 4. This obviously leads to duplicated work , as only the output of the DrillBeyond operator changes between executions, while the other parts of the query plan operate the same. Consider the query plan in Figure 3.7. Here, only the values of the `gdp` attribute would change between query runs, while the other operations, notably the more expensive joins with the `Customer` and `Orders` relations, would not change. This means that simple re-execution would increase time to compute the query k-fold, with most of the effort being inefficient duplicate work. Our first approach to this problem is to identify and then maximize invariant parts of the multiple executions, preventing their re-execution by materializing intermediate results. As a first observation, note that all tuple flows below the DrillBeyond operator can actually never change between executions: in the example shown in Figure 3.7, these operators are highlighted. The cost of those operations can be minimized by materializing the input to the DrillBeyond operator. This elementary optimization is already included in the basic operator implementation shown in Algorithm 4, in the form of the tu-

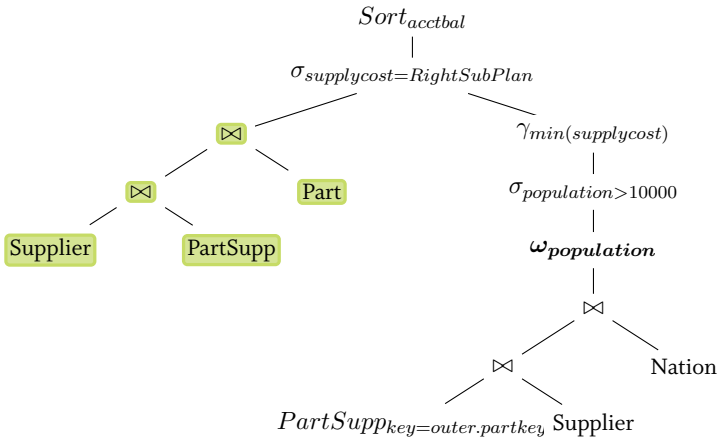


Figure 3.8: Invariant Subtrees

ple store created by the operator. However, the changing augmentation output may influence the result of other operators further up the query plan. In the example, the aggregation of the `gdp` attribute will change its output in each iteration based on the augmentation values provided by the DrillBeyond operator. In addition, even the aggregation of the regular attribute `totalprice` is influenced by the changing `gdp` values: Since the selectivity of the predicate on `gdp` may vary using different augmentations, the set of `Orders`-tuples that has to be aggregated may vary in different executions as well.

The next sections will introduce our optimization strategies for these problem strategies: invariant caching, augmentation operator splitting, selection pull-up, projection pull-up, partial selection, and finally, run-time reoptimization.

3.4.1 Invariant Caching

We already discussed the necessity for materializing intermediate results, and introduced the basic caching done by the DrillBeyond operator itself. However, this is not the only opportunity for caching. Consider the following query plan in Figure 3.8, modeled after TPC-H query 2, in which a dependent scalar subquery, on the right side, is executed for each tuple of the outer query. A DrillBeyond operator and a predicate on the augmented attribute was added in the dependent subquery. Here, the result of the main query before the selection is invariant between query results, as the augmented values only change the result of the filtering subquery.

In general, operators that have no DrillBeyond operator in their descendants are invariant. We exploit this using a planning pass, that inserts materialization nodes into the plan, which is depicted in Algorithm 5. The idea is to recursively descend into the query plan, until a DrillBeyond operator is encountered, which marks all nodes above it as varying. Furthermore, a node is marked if any of its child subtrees or any subqueries evaluated at the node are marked.

At each varying node, we check whether explicitly materializing any subtree would be ben-

Algorithm 5 Add MatNodes Planning Phase

```
function AddMatNodes(node)
    if type(node) == DrillBeyond then
        return true
    lvar ← AddMatNodes(lchild(node))
    rvar ← AddMatNodes(rchild(node))
    sqvar ← ∀sq ∈ subQueries(node) | AddMatNodes(sq)
    if (lvar ∨ rvar ∨ sqvar) then
        if ¬lvar ∧ ¬(param(lchild(node)) ∧ rvar) then
            Materialize(lchild(node))
        if ¬rvar ∧ ¬(param(rchild(node)) ∧ lvar) then
            Materialize(rchild(node))
    return (lvar ∨ rvar ∨ sqvar)
function Materialize(node)
    if MaterializesNatively(node) then
        return node
    return makeMatNode(node)
```

eficial. This is the case if the subtree is invariant, and the subtree's top node does not offer native materialization. A good example are intermediate join results, such as the whole left subtree underneath the top selection in Figure 3.8. Here, the addition of an explicit materialization node above this subtree would speedup further executions of the query plan considerably. If the top node of the invariant tree does materialization natively, e.g., sort nodes or inner sides of hash joins, we can just reuse this intermediate result and no extra work is necessary. Note, that even a subtree without DrillBeyond operators may be varying, if it contains parameters, e.g., the inner side of a nested loop join. In this case, the variability of a sibling subtree will propagate, as the parameters may change between executions if the other subtree is varying.

Having introduced our methods for invariant caching as a first step to optimize multi-result queries, the limits of this first method have also become clear: We can only cache invariant subtrees, and depending on the cost-based placement of the DrillBeyond operator, most of the query may be varying and thus not amendable by caching. We will therefore introduce further optimizations in next sections.

3.4.2 Separating Augmentation Input and Output

So far we have optimized by caching invariant parts of the query plan as it is produced by the planner. However, our planning process as described in Section 3.3.2 assures only that the overall order of joins and DrillBeyond operators is optimal with respect to the cost model, and correct with respect the placement bounds from Section 3.3.4. It does not however, try to maximize the invariant parts of the query. Consider the following example queries, shown in Figure 3.9. The first case (Figure 3.9a) is optimal with respect to a single query execution: the operator is applied to the smallest possible intermediate result, while the placement bounds are

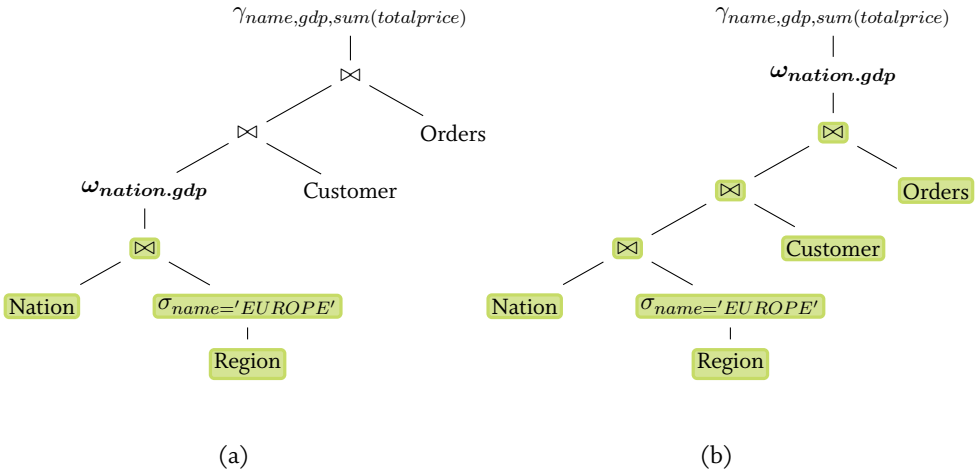


Figure 3.9: Influence of DrillBeyond operator placement

adhered to. Still, with respect to multiple execution and the caching mechanisms described above, the second plan (Figure 3.9b) is more efficient. By moving ω up the tree, this plan trades higher cost for a single execution of the DrillBeyond operator for a much larger invariant subtree. This pays off over more executions of the query plan. While it would be possible to use an extended cost model that incorporates the costs for multiple plan executions and thus choose the second plan, we can do even better.

The key observation is that to maximize the invariant parts of the query plan, the operator should be placed as late as possible, i.e., not earlier than the augmented values need to be accessed. This is the upper bound to the ω operator placement.

Definition 3.4 (Latest Possible Placement)**.** *The DrillBeyond operator augmenting yielding values for an attribute a_+ may not be placed later than the first access to attribute a_+.*

Though a seemingly obvious observation, it enables a useful optimization: we can separate the input part of the operator from the actual projection of augmented values.

We split the DrillBeyond operator in two parts, one performing the augmentation, symbolized with ω, and one doing the projection of values, depicted as Ω. The ω operator performs the input hashing and triggers the augmentation process, i.e., it works through the *collect()* and *augment()* phases of the basic operator implementation shown in Figure 3.3.2. However, instead of projecting the actual augmented values, it projects placeholders, which can be uniquely mapped to the actual value arrays returned by the augmentation system. Technically, these can be implemented as pointers into the *augMap* hash table that is populated in the *augment()* phase. These placeholders traverse the operator graph until the Ω operator, which dereferences to the correct value array, and projects a single augmented value depending on the current query iteration, i.e., the current n in Algorithm 4. We place Ω at exactly the latest possible placement defined above, and ω with respect to the placement bounds defined in Section 3.3.4. We therefore minimize augmentation operator cost, while maximizing the size of the invariant query subtrees.

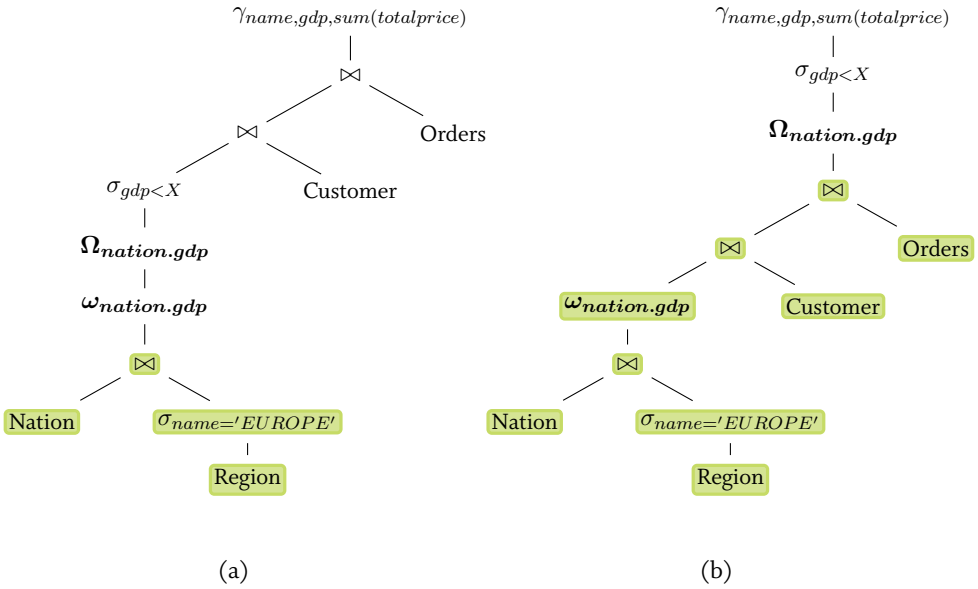


Figure 3.10: Selection Pull-up and influence on invariants

This optimization provides more efficiency gains, but is still limited by the first access to the augmented attribute in the query plan, i.e., its effect is based on the distance between ω and Ω. We will therefore explore more optimizations that delay the first access to an augmented attribute: *selection pull-up*, *projection pull-up* and *partial selection*.

3.4.3 Selection Pull-Up

The strategy of projecting augmented attributes as late as possible is in conflict with one of the more fundamental classic optimizations, namely selection push-down. For example, if a predicate on the augmented attribute is given, it would make sense to evaluate this predicate as soon as possible, i.e., directly after ω and before any other joins, to limit the size of intermediate results, as done in traditional query optimization, i.e., $(\ldots \bowtie \sigma_{a_+}(\omega_{a_+}(R))$. This however means that our ω / Ω split cannot be applied anymore, since augmentation and first access to the values coincidence, i.e., we would have $(\ldots \bowtie \sigma_{a_+}(\Omega_{a_+}(\omega_{a_+}(R)))$, eliminating all benefit of the split. We therefore have to decide between two alternatives, depicted in Figure 3.10: traditional selection push-down at the cost of having more varying nodes in the plan, or the opposite approach: *selection pull-up*. With this method, we place Ω as described in Section 3.4.2 while ignoring predicates on the augmented attribute, i.e., before the first other access. The trade-off in this case is between smaller intermediate join results with the plan in Figure 3.10a and larger but invariant joins in Figure 3.10b. Which plan is to be favored depends on the selectivity of the predicate on the augmented attribute. We thus employ a cost-based approach to decide on whether to use selection pull-up. Two questions are to be answered: what cost model to base this decision on, and how to estimate the selectivity of the predicate. Concerning

```
select
    n_name,
    sum(price * (1 - discount)) as revenue,
    (sum(price * (1 - discount))) / nation.population
from
    customer, orders, lineitem,
    supplier, nation, region
group by
    n_name;
```

Figure 3.11: Example of a query amendable by projection pull-up

the cost-model: since we have a binary decision we only need to compare the results of two suitable models and pick the cheaper plan. We use the following two models for the cost of the subtree above the ω:

$$C_k(\omega, \top) = \begin{cases} C_1(\omega, \Omega) + \sum_{i=1}^{k} s_i \cdot C_1(\Omega, \top) & \textit{(Pull-Up)} \\ \sum_{i=1}^{k} s_i \cdot C_1(\omega, \top) & \textit{(no Pull-Up)} \end{cases}$$

Here, C_k is the cost of k plan executions, s_i the selectivity of the i-th data source for the attribute in question, and $C(x, y)$ denoting the cost of the subplan from operator a until to operator b. With selection pull-up, we have to pay the price for the subplan between ω and Ω only one time, but with selectivity 1.0, as no predicate is applied. The rest of the plan, from Ω to the result, has to be executed k-times, but with the predicate applied. Without pull-up, the selection is applied early, but the whole plan is executed k-times.

The cost of the partial queries can be calculated from the normal query optimizer output, the k is user-defined and fixed, but what about the selectivities s_i?

As discussed in Section 3.2, we have no knowledge at plan-time about which data sources will be used, and thus can not easily estimate their selectivities. Instead of deciding the selection placement at plan-time when selectivities are hard to estimate, we therefore make the decision at run-time, specifically after the augmentation system has returned, i.e., in the *augment()* phase shown in Algorithm 4.

With the selectivities known, the ω operator decides at run-time, using the cost-models defined above, between two alternatives: It can either project real values and perform the selection, in which case Ω operator becomes a dummy, and further executions must recalculate all operators in the (ω, Ω) range. Or it can pull-up the selection, and emits only placeholders as described in Section 3.4.2, in which case the selection is performed after Ω.

This dynamic selection-pull up allows DrillBeyond to exploit knowledge of the augmentation attribute's selectivity available at runtime to optimize overall query performance.

3.4.4 Projection Pull-up

So far, we have introduced the methods for materializing query invariants, and methods for increasing the portions of the query that are invariant on the level of operators. Recall that the goal of the selection pull-up optimization was to restructure the plan so that the projection of augmented values can be delayed. A similar optimization can be applied to any projection

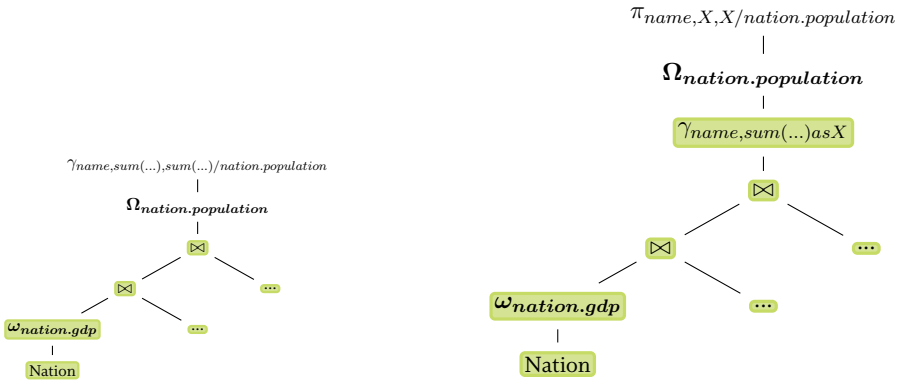


Figure 3.12: Projection Pull-up and influence on invariants

expression. Consider the modification of TPC-H query 5 shown in Figure 3.11. With this query, the user wants to compute not only revenues by nation, but also wants to relate the revenue to the respective nation's population. This leads to the whole aggregation being a varying node, necessitating the Ω to be placed before the aggregation, and thus the k-times execution of the aggregation, as shown in Figure 3.12a. The idea of the projection pull-up optimization is to split expressions that contain augmented attributes into varying and invariant parts and separate the computation of these sub-expressions into different plan nodes. The aim is to move the varying part of the expressions up the query tree, adding explicit projection nodes if necessary. Consider Figure 3.12b: instead of computing the expression $\frac{\sum(extendedprice*(1-discount))}{population}$ at the grouping node, we split the expression into $(\sum(extendedprice * (1 - discount)))$ as X and $\frac{x}{population}$, and compute only the first expression at the group by node, while moving the second and final expression into an explicit projection node above it. This pull-up enables us to pull the Ω operator with it, since the first access to the augmented attribute is moved. In the example, this allows the grouping node to be treated as invariant.

We perform the expression splitting at a node by recursively descending into the expression tree until its atomic components: attribute references or aggregate functions. If this split yields both varying and invariant attributes, then there is potential for a projection pull-up. In this case, the node's output specification is rewritten to the atomic components, and the original expressions composed from these components are moved to a dedicated projection node above it. This optimization allows DrillBeyond to decouple calculations on augmented data from calculations on local data, which further increases invariant parts of the query plan.

3.4.5 Partial Selection

In the previous section, we discussed how the position of selection operators leads to trade-offs between intermediate result cardinalities and plan invariability. In this discussion, pulling or pushing the selection was a binary decision: either execute the selection at ω, or at the

corresponding Ω operator. This assumed that the Web data sources used for the different augmentations are completely independent, i.e., it assumed that each augmentation will lead to a unique intermediate result. However, for many augmentation queries, the sources may actually agree on whether a predicate holds for a certain tuple. We can therefore introduce a *partial selection* operation, which selects all tuples for which the selection predicate holds in all of the available augmentations.

$$partialSel_P(R) = \{t \in R \,|\, \exists i \in (1..k)\ P(t[a_{+_i}]))\}$$

Here, a_{+_i} denotes the augmented attribute in its i-th possible variant. The selectivity of this operation depends entirely on the correlation between the sources with respect to the predicate. It may range from being equal to the selectivity of the individual sources if they completely correlate, to being 1.0 if the predicate holds for every tuple in at least one augmentation.

Even though the effect varies, it can lead to better query performance because it can be applied at the ω operator, while still returning an invariant result. It can therefore be combined with selection pull-up described in Section 3.4.3, by returning placeholder values for tuples that pass the partial selection, and performing the data source-dependent final selection at the Ω operator.

Partial selection can therefore, depending on the agreement of the sources, minimize intermediate results lower in the plan, while still keeping the result of the selection invariant throughout multiple executions.

3.4.6 Runtime Reoptimization

The optimizations proposed so far where concerned with optimizing the plan regarding invariants and selectivities. Dynamic selection pull-up even allows to adapt the plan at run-time, depending on the selectivity of predicates on augmented attributes. However, if the actual selectivity is very different than the one used for initial planning, there may be more optimization potential than just the placement of the selection operation. For example, the original choice of access paths for base relations, e.g., index vs. table scan, may be suboptimal with respect to the actual selectivity. We therefore use a form of adaptive query processing: When executing a query, DrillBeyond always starts execution at the subtrees below ω operators, to be certain about selectivities as early as possible. DrillBeyond then invokes the planner to create a new plan using the better selectivity estimate. This is conditioned on the difference between the estimated selectivity used to create the original plan, and the real selectivity known after partial plan execution.

At this point, either the average selectivity of the k augmentations or the selectivity of the partial selection over all augmentations (Section 3.4.5) can be used as an estimate. This decision depends on whether selection pull-up will be used with the new plan, and can be made using the selection pull-up cost model (Section 3.4.3). The already executed subtrees underneath the ω nodes are then merged into the new plan, to make sure that materialized intermediate results and especially the results of the augmentation operation are reused with the new plan. This run-time reoptimization can lead to improved query performance at the cost of one additional execution of the query planner.

3.5 EVALUATION

In this section we present the results of our experimental study on efficiency of open world SQL query processing in the DrillBeyond system. We study the performance of the DrillBeyond relational operator in general, and the influence of our optimizations in particular: intermediate result caching, the ω/Ω split, selection pull-up and partial selection and run-time reoptimization. We also study how the use of SQL query context, especially predicate push-down to the augmentation system, can improve both augmentation quality and runtime. This section is organized as follows: we will first introduce our experimental setup, i.e., the systems and Web data sets, as well as test databases and SQL queries used. We then move on to the query performance experiments, and finally study augmentation quality improvements.

3.5.1 Experimental Setup

In Section 3.3 we introduced our system architecture consisting of three components: the modified RDBMS, an augmentation system and a Web data source index. In this section we will detail all of the implementations used for in evaluation.

Implementation We implemented the operator described in Section 3.3 and the optimizations described in Section 3.4 into the PostgreSQL[6] RDBMS and made our modified version available open source[7]. The optimizations discussed in Section 3.4 can be turned on and off individually on a per-session basis, which allows to study their individual performance impacts. The augmentation system we employ is the REA system presented in Chapter 2, also available open source[8]. For the experiments in this chapter, it was extended to accept predicates as query hints. They are used to guide the search for data sources, as described in Section 3.3.5. To provide a list of candidate sources for the augmentation, we use an index over our *Dresden Web Table Corpus*[9] (Eberius et al., 2015a).

Test Database Throughout the chapter we used examples based on the TPC-H benchmark. However, TPC-H only has two relations with natural identifiers for which Web data will be available: `Nation` and `Region`. These contain geographic names, i.e, country and continent names respectively. All other relations in TPC-H, such as `Supplier` and `Part` carry only artificial identifiers, i.e., "supplier_1" or "part_1", and can therefore not be extended with external data due to their lack of real-world identity. We tackled this issue in two ways: First, for the pure performance experiments that test our query processing optimizations, we use an entity augmentation system that generates artificial data instead of looking it up on the Web. This also has the advantage that we can exactly specify properties of the returned augmentations, such as selectivities and correlation between augmentations, to test various aspects of query processing.

[6] http://www.postgresql.org/
[7] http://github.com/JulianEberius/DrillBeyond
[8] http://github.com/JulianEberius/REA
[9] http://wwwdb.inf.tu-dresden.de/misc/dwtc

```
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)),
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'HOUSEHOLD'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '1995-03-29'
    and l_shipdate > date '1995-03-29'
    and customer.employees > X
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
limit 10;
```

Figure 3.13: Exemplary modified TPC-H query 3, extension highlighted

Second, for the quality related experiments, we created a variation of TPC-H, which replaces generic identifiers with real-world entities. For example, the names in the `Supplier` relation are replaced by real company names crawled from the Web. For all experiments, we used TPC-H with scale factor 1.0.

Test queries We focus on a subset of the TPC-H queries in which dimension tables are used, as fact tables such as `Lineitem` or `Orders` do not contain information about real world entities that could be augmented. We added arbitrary attributes to one of the dimensions, by adding a `where`-clause of the form "relation.A > X" to each query. An example for such a modified query is shown in Figure 3.13. For the performance experiments, A and X are arbitrary, since we use generated augmentation data with customizable selectivities for these experiments. For the quality experiments we will give specific (relation, attribute, predicate)-tuples in the respective sections.

Parameters The main parameters are k, the number of augmentations and thus the number of SQL results to be produced for a query, and s, the selectivity of the predicate on the augmented attribute. If not stated differently, we perform our experiment for each $k \in (1, 3, 5, 10)$ and ten different levels of selectivity s, ranging between 0.01 and 0.99.

Configurations With the experimental setup and parameters as described above, we measured five different configurations of DrillBeyond to test the optimizations introduced in Section 3.4. The configurations are:

- **No Optimization**: The query is planned using just the cost model introduced in Section 3.3.6 and default selectivity estimates. This means the query is planned for one execution, and not optimized for producing multiple results.
- **Invariant Caching**: In this configuration we enable the caching of invariant plan subtrees, as described in Section 3.4.1.
- **Static ω/Ω**: This configuration introduces the split between the augmentation operator ω using the cost model and the projection operator Ω placed at the first attribute usage,

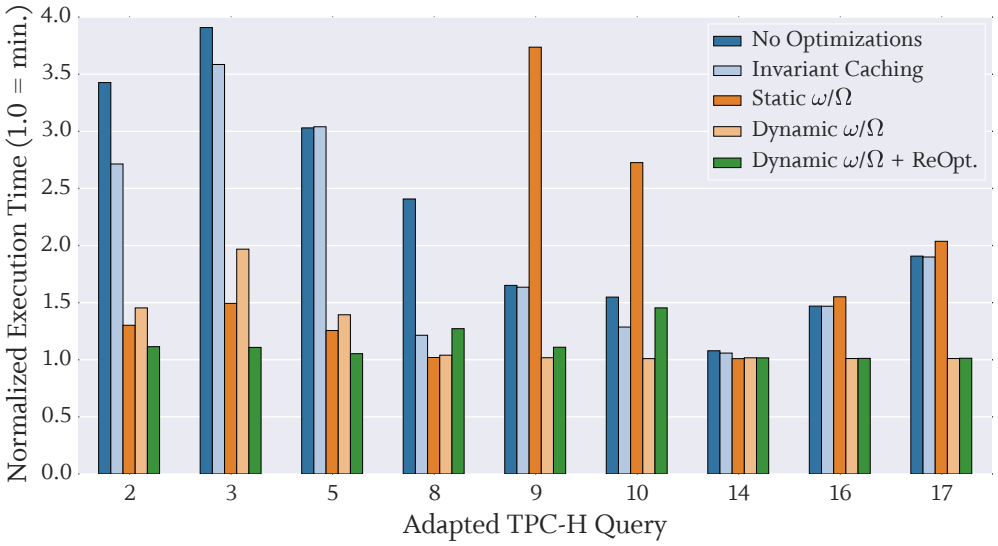


Figure 3.14: Overview of normalized execution times by query

as introduced in Section 3.4.2. In this context, "static" means that the selection is always performed at the Ω operator, maximizing invariant plan nodes, but at the cost of never utilizing the selectivity of predicates.

- **Dynamic ω/Ω**: This configuration introduces dynamic, run-time placement of the selection at either ω or Ω, as described in Section 3.4.3.
- **Dynamic ω/Ω + ReOpt.**: In this configuration, we enable the reoptimization of the remainder of the query plan after ω is executed, as introduced in Section 3.4.6. The time needed for the reoptimization itself is included in all measurements.

3.5.2 Performance

In this section, we will discuss our experimental results regarding the performance of the DrillBeyond system, especially with respect to the effects of the optimization techniques introduced in Section 3.4. We will show how our goal of enabling hybrid relational/augmentation queries can be reached with minimal runtime overhead compared to pure relational queries. If not mentioned otherwise, the figures in the following sections show normalized execution runtime, i.e., all times are given as multiples of the fasted configuration in each individual measurement. This measures was chosen to be able to compare relative performance improvements across the different TPC-H queries with their highly varying absolute run times.

By Query A first overview of our results is given in Figure 3.14, where the data is shown aggregated by query, i.e., aggregated over all of settings of k and s. The first major observation is that the various optimization techniques respond quite differently to different queries. For example, invariant caching alone is almost good enough for Q8, as most of its runtime is spent in a large join between filtered `Order` and `Part` relations, which can be cached as invariant.

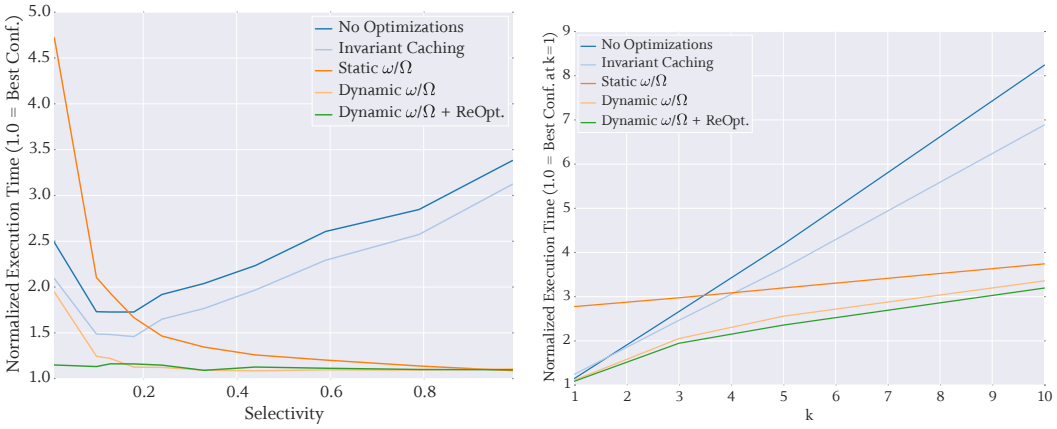
Scenario	Normalized Execution Time	Standard Deviation
No Optimizations	2.27	1.82
Invariant Caching	1.99	1.59
Static ω/Ω	1.79	2.35
Dynamic ω/Ω	1.21	0.67
Dynamic ω/Ω + ReOpt.	1.13	0.28

Table 3.1: Normalized execution time, average and standard deviations

The other techniques do not improve the performance significantly, runtime reoptimization even introduces a small overhead in this case. However, in all other cases, invariant caching alone is not enough. Introducing the Ω operator with static selection improves most queries considerably, by making most of the plan invariant. This static optimization already works well for queries 2, 3, 5 and 8, which already include highly selective predicates on regular attributes. However, some queries, such as Q9 and Q10, can profit strongly from a further selective predicate, so that a static placement of the selection at Ω looses much optimization potential. In these cases, even the basic invariant caching approach performs better, despite re-executing more plan nodes, because it can execute the predicate directly at ω. Keep in mind, that Figure 3.14 shows results for a mix of high and low selectivity predicates on the augmented attribute. We will individually analyze the influence of selectivity later in this section.

Continuing with the dynamic configurations, we can see that these improve even more queries, as they can dynamically place the predicate execution depending on k and runtime knowledge of s. However, we can also see that introducing a new cost-based dynamic decision into query processing also introduces some new uncertainty. For example, while the dynamic selection approach without plan reoptimization produces the best plan in Q10, it is even worse than static selection in queries Q2, Q3 and Q5. Here, the runtime switching of the selection leads to the use of sub-optimal access methods for regular relation scans, as those are chosen at plan-time. This is alleviated via run-time reoptimization, which should lead to the best query plans in theory. However, it is not optimal for all queries either, as seen in Q10. In Table 3.1, we can see that while the reoptimization method does not gain much on the dynamic approach in average, its result show a considerably lower standard deviation, i.e., it produces more conservative plans.

Selectivity Analysis Figure 3.15a shows the same data, but aggregated by selectivity of the predicate on the augmented attribute. First, we can observe that invariant caching improves runtime by a constant offset, independently of selectivity. This is because caching applies only to subtrees below DrillBeyond operators, while the selectivities that we vary in this experiment apply above these operators. Second, this view of the data also confirms our explanations from the previous section: The ω/Ω-split drastically improves runtime if the selection is not an important factor for the overall runtime. However, it is an inefficient choice if the predicates on the augmented attributes are highly selective. Enabling or disabling selection pull-up at run-time using our cost model leads to improved results across the whole selectivity spectrum.



(a) By Selectivity

(b) By k

Figure 3.15: Normalized Runtimes

Finally, our run-time reoptimization further improves performance on the lowest selectivity levels, where it usually replaces sequence scans with index scans based on the run-time selectivity knowledge.

Analysis of Factor k In Figure 3.15b, the results of our main experiment are aggregated by the factor k, i.e., the number of alternative augmentations considered which corresponds in turn to the number of SQL results generated. To discern scaling behavior, the values in this figure are normalized to the value of the best configuration at $k = 1$.

First, note that the unoptimized configuration scales almost linearly, as to be expected. Producing $k = 10$ results leads to a runtime that is about 8.2 times higher. Next, we can observe that the effect of invariant caching scales with k, i.e., the more result variants are to be generated, the more work is saved through caching of intermediate results. For $k = 1$, the minimal overhead of materialization is visible. In this case, the invariant caching configuration is slower than the baseline. We can again see that static selection pull-up leads to inefficient plans for low k, as the larger intermediate results it produces outweigh the gains through better caching opportunities. Our dynamic approach again uses the best variant on average. Finally, the improvements achieved by reoptimization are independent of k as expected. In conclusion, we can see that our optimizations allow to produce multiple query results based on alternative augmentations with a comparatively low overhead: Over all queries and selectivity levels, we can produce $k = 5$ results with an overhead of 2.3 and $k = 10$ results with an overhead of 3.2.

Individual Query Analysis To give an insight of how the observed aggregated effects arise, Figure 3.16 shows absolute execution times for a single query, Q9, plotted by selectivity of the predicate, once for each $k \in (1, 3, 5, 10)$. For this individual query, absolute runtimes are shown. We can clearly observe the difference between pushing or pulling the selection on the augmented attribute. In the invariant caching configuration, selection is performed early at ω, which is faster for low selectivities and low k. However, the higher variability in the plan leads to higher re-execution costs for repeated query executions, making this strategy infeasible, e.g.,

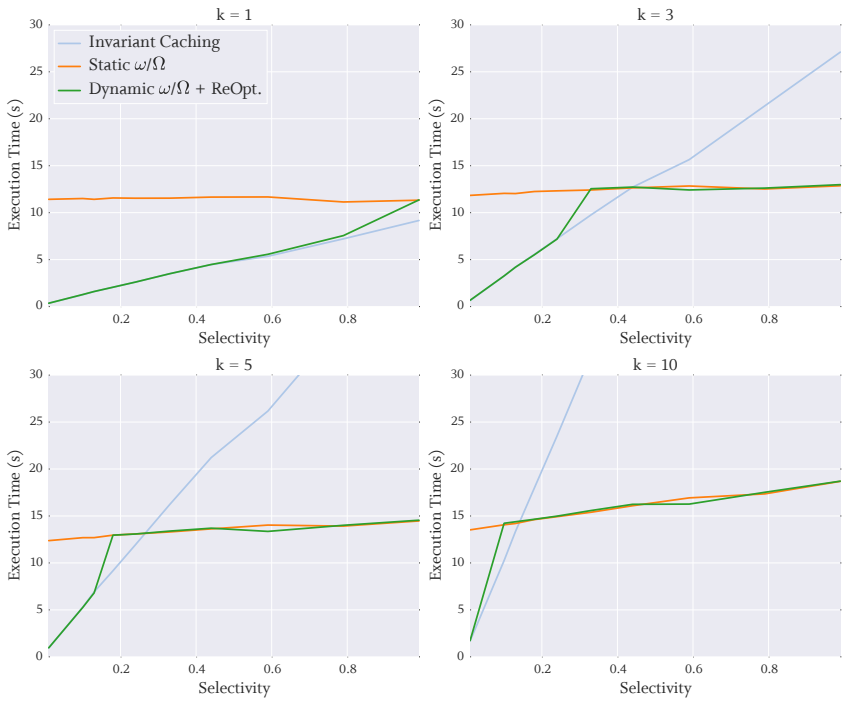


Figure 3.16: Execution time for query 9, by selectivity

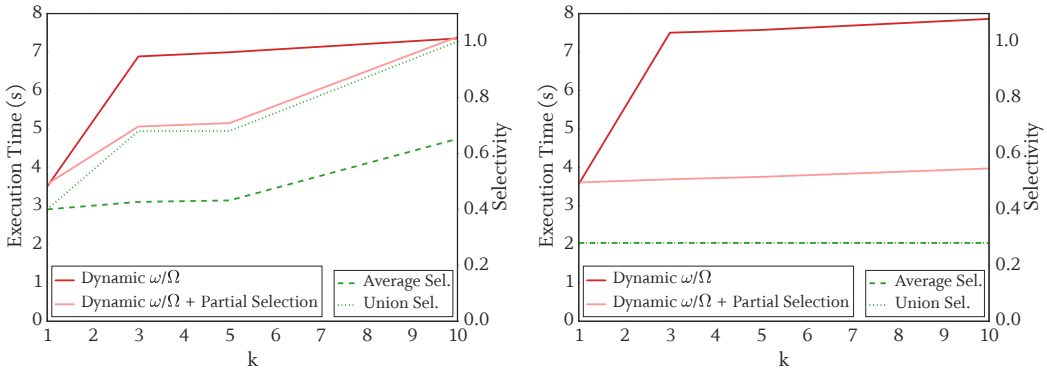
for $k = 10$. In contrast, in the case of static selection at Ω, most of the plan becomes invariant, and thus repeated executions are not very costly. However, this strategy can not benefit from low selectivities. The dynamic switching combines the benefits of both variants, although the cost model is not perfect: selection pull-up is chosen slightly to early for this query.

Partial Selection Next, we will study the effects of partial selection, as introduced in Section 3.4.5. As mentioned in Section 3.5.1, our main performance experiments have been carried out with generated augmentations, so various levels of selectivity could be easily observed. Since partial selection depends mostly on the correlation between the data sources that the augmentations are based on, it is more interesting to study with augmentations using real Web data. So for the next experiment, we employed our Web table retrieval and matching system REA, as described in Chapter 2, and used the SQL query template shown in Figure 3.17, which was modeled after TPC-H query 5. We evaluated this query with two different (attribute, predicate)-pairs: (gdp, >60) and (population, >1E8). The results are shown in Figure 3.18. In this figure, the x-axis shows k, the number of augmentations requests, while the first y-axis shows the execution time with and without partial selection. Finally, the second y-axis shows the average selectivity over all augmentations, as well as the union selectivity, i.e., the selectivity of the partial selection over all augmentations as defined in Section 3.4.5. The union selectivity can be understood as the ratio of tuples that pass the predicate in at least one of the augmentations.

In the first case, we can see the effect of the number of augmentations on union selectivity.

```
select n_name, <attr>, sum(...)
from nation, customer, orders, supplier, lineitem
where
    ...
    and <attr> <predicate>
group by n_name, <attr>
order by <attr> desc
```

Figure 3.17: Query template used for the partial selection evaluation



(a) gdp > 60.0

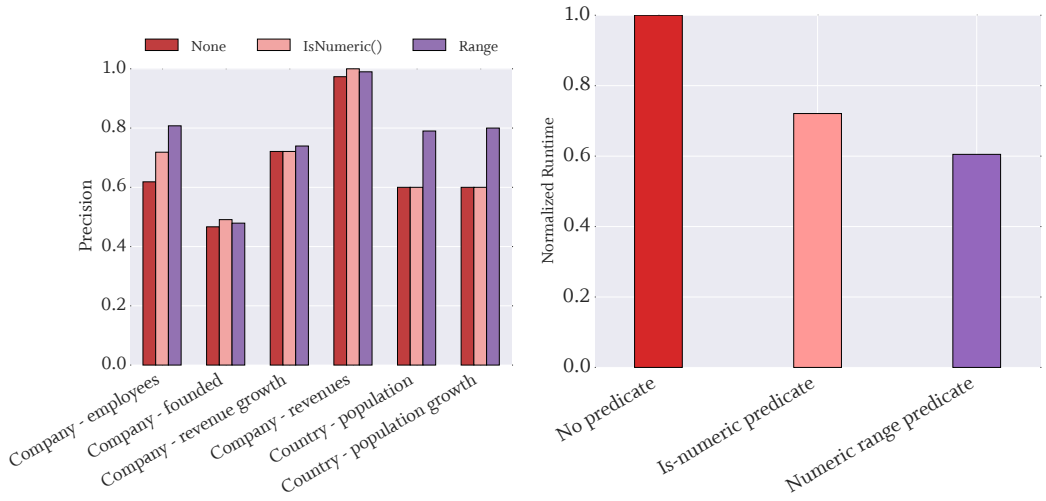
(b) population > 1E8

Figure 3.18: Effects of partial selection

Although the average selectivity rises slowly until 0.65, the union selectivity rises more quickly and reaches 1.0. This means that as k rises, even though in all augmentations only a subset of nations is selected, all nations are selected at least once. However, as shown in Figure 3.18, as long as the union selectivity is below 1.0, the runtime can be improved proportionately using partial selection. Whether this is the case depends entirely on the correlation between data sources used. Consider Figure 3.18b, which shows the much less ambiguous query for "population". In this case, all augmentations agree on which countries have more than 100 million inhabitants, which leads to equal average and union selectivity. This allows to save a corresponding amount of work in query processing, as large parts of the database are filtered by early partial selection.

3.5.3 Augmentation Quality

So far we have discussed only the effect of the performance optimizations proposed in this paper. However, in Section 3.3.5, we also claimed that the quality and runtime of the actual entity augmentation process can be improved by pushing context available in the outer SQL query to the entity augmentation system. So for the next experiments, we again used the REA system and extended it to handle predicates on augmentation attributes as guides for the Web tables search and scoring process. Following our discussion in Section 3.3.5, we first added type information as a query hint, by implementing an "*IsNumeric*" predicate in REA. This allows



(a) Changes in precision by type of predicate used (b) Normalized runtimes by type of predicate used

Figure 3.19: Effects of pushing predicates to the EA system

Concept	Attribute	Predicate
company	employees	>50000
company	founded	<1900
company	revenue	>100
company	revenue growth	>10.0
country	population	>1E8
country	population growth	>2.0

Table 3.2: Concepts, attributes and predicates used in precision evaluation

DrillBeyond to signal REA that the augmented attribute should be numeric, which in turn allows REA to filter non-numeric Web table columns from its search. In a second step, we allowed range predicates to be passed to REA, which uses the equation given in Section 3.3.5 to improve its data source ranking.

With this improved REA system, we ran a subset of the precision tests given in the REA evaluation in Chapter 2. The set of concepts and attributes used, as well as the predicates we added to test predicate hints are shown in Table 3.2. The results are shown in Figure 3.19a. We can see that adding type information and range predicates improves precision, although the changes are notable only in some domains. For example, in the "employees" domain, the effect was mostly due to filtering Web tables that gave information about local branches of the queried companies, giving either no numbers, or low numbers in relation to the query predicate. In the "country" domain, the original REA system sometimes mistakenly uses Web tables about capital cities to answer "country"-queries. In this case, the added range predicate allowed for a more precise augmentation. Lastly, Figure 3.19b shows the change in augmentation runtime. The gains shown are proportional to the share of data sources that can be pruned early due to mismatching data type in the case of using the "*IsNumeric*" predicate, or due to being non-discriminative with respect to the given predicate.

In conclusion, we have shown that DrillBeyond is capable of processing Open World SQL queries, i.e., hybrid relational/augmentation queries based on Web tables. We have demonstrated the effectiveness of various optimizations in minimizing the runtime overhead of producing multiple SQL query results based on alternative augmentations. We further showed that pushing SQL query context into the EA system can improve quality and performance of the EA query processing compared to standalone processing.

3.6 RELATED WORK

In this section, we will survey related work and put it in relation with our proposed DrillBeyond system. In general, to the best of our knowledge, there is no previous work that allows the user to augment a relational database with external data at query-time by simply adding arbitrary new attributes to an SQL query. However, there are considerable amounts of works that can be related to the various individual aspects of DrillBeyond. First, since DrillBeyond aims at supporting users in analytical workflows with ad-hoc integration capabilities, we discuss similar works on ad-hoc self-service BI. We then briefly touch the classic problems of open versus closed world databases, as well as null value semantics, to put the Open World SQL queries processed by DrillBeyond into the right context. In general, DrillBeyond is a hybrid system, integrating entity augmentation and relational query processing. Therefore, the most directly related areas of research are entity augmentation itself, and integration of external information systems with RDBMS. However, since we already discussed general entity augmentation research, we refer the reader to Section 2.6.2, and focus this section on the system integration aspect. For example, we will discuss related work on hybrid systems between information retrieval and database management, as well as other forms of in-DBMS analytics and integration functionality. Next, we will discuss systems facilitating query formulation, such as keyword search interfaces for RDBMS. These system aim at enabling a user to query a database without knowing its exact schema, which is similar to DrillBeyond's intent to allow a user to search and integrate data without prior knowledge of the external sources. Moreover, DrillBeyond is concerned with tackling uncertainty in its results, by producing top-k result variants. We will therefore explore other techniques for dealing with uncertainty in DBMS. Lastly, we proposed efficiency optimizations for queries producing multiple result variants, and for planning queries with incomplete knowledge of the data sources that will be used. We will therefore have a detailed look at methods for efficient processing of multi-result queries, and discuss related work from the areas of multi-query optimization and adaptive query processing.

3.6.1 Self-service BI using Web Data

A work closely related in spirit to ours is (Alberto Abelló, 2012). In this paper, the authors discuss their vision for self-service BI, in which an existing data cube can be semi-automatically extended with so-called *situational data* in the course of an interactive OLAP session. Such situational data is distinguished from regular warehouse data by its focus on a narrow domain, short life-span, lower reusability, or smaller group of concerned users. However, such data, although not modeled in the warehouse at hand, can still be critical to a specific decision process.

Therefore, they envision an OLAP operator which, based on keyword queries and an interactive question-answer process, performs the discovery and selection of data sources, as well as the data acquisition and integration for a given ad-hoc information need. However, in contrast to the work at hand, it is a vision paper, and does therefore not present a concrete system design or experimental results. Further, the envisioned technical paradigm or the cube extension is based on RDF, while ours is relational with a top-k extension. Further, the proposed process is semi-automatic, incorporating user feedback at every stage of the process through a suitable interface. DrillBeyond on the other hand operators fully automatic based on a SQL query specification, deferring user interaction until the choice of result. Still, the work at hand was partially inspired by (Alberto Abelló, 2012), which also coined the term "drill-beyond" for its envisioned OLAP operator.

In (Löser et al., 2009), a vision for ad-hoc BI based on data extracted from the textual Web content is presented. They envision using cloud-computing and parallelized information extraction methods to answer OLAP queries over the general Web, e.g., by enabling ad-hoc extracting of product data, reviews and customer sentiments from the general Web.

Further, there is a considerable amount of work aimed at bridging traditional OLAP and semantic Web technologies to enable novel, agile forms of business intelligence. (Abello et al., 2015) gives a comprehensive survey of these efforts to utilize the semantic Web to acquire and integrate relevant external data with internal warehouse data. For example, (Etcheverry and Vaisman, 2012) propose a RDFS vocabulary for expressing multidimensional data cubes, so-called *Web Cubes* over semantic Web data, and show how traditional OLAP operations can be performed over combined internal and Web cubes.

3.6.2 Open/Closed World Assumption and Null Value Semantics

In some respect, our work is in line with classic database works on the dichotomy of closed vs. open world assumption (Reiter, 1978), as well as works on the different possible semantics of null values (Roth et al., 1988). The closed world assumption states that all statements not explicitly modeled in a database is assumed to be false, while the open world assumption states that facts not existing in a database are simply unknown, and may have no, any, or even multiple values. Traditional RDBMS usually use the closed world assumption, i.e., they can not provide any results based on facts that are not modeled in them. DrillBeyond is more closely related to the open world assumption as it can process queries based on attributes not defined in the database it directly manages. Concerning null values, typical interpretations are that null values are either *not existing* or simply *unknown*, though others have been studied. For example, (Gottlob and Zicari, 1988) introduced the idea of another type of null value, called *open*, that symbolizes a value that may or may not exist, or may have one or more values. Using these *open* values, the database can be selectively convert to an open world database, by introducing *open* for individual values, complete columns or even rows. In the last case, i.e., by adding a tuple $t_{open} :< open, open, \ldots, open >$, the database is transformed into a complete open world database. They propose formal semantics of relations with *open* values, and extend the standard relational and logical operators to cope with these values. Their work is motivated mainly by the higher expressive power it provides. They are not concerned with how these open values could be evaluated to regular values, but aim at processing queries and providing

results with these *open* values as first-class citizens. DrillBeyond can be seen as a special case of this work, as it allows only columns to be "opened". It also limits the usage of open columns to query-time, and does not allow them as modeled elements in the database. However, it provides a mechanism for resolving these open columns to actual values using external sources, and provides specialized top-k query processing to tackle the resulting uncertainty.

3.6.3 Hybrid DB/IR Systems

We already established that there are no other works on embedding entity augmentation into RDBMS in the way DrillBeyond does. However, augmentation systems are strongly related to information retrieval systems, and closer integration between the worlds of database and IR research have long been a goal of both communities (Weikum, 2007; Chaudhuri et al., 2005; Amer-Yahia et al., 2005). There are several classes of DB/IR integration, and multiple system architectures for achieving them. First, there are works that aim at including IR capabilities, such as full text search into database engines, e.g., to support keyword queries on textual attributes of a database. Such systems have already been available in commercial RDBMS for some time (Dixon, 2001; Hamilton and Nayak, 2001; Maier and Simmen, 2001).

Of course, the other direction of system integration is also possible: extending information retrieval system with more database-like features. For example, in (Pound et al., 2010) a method for processing semi-structured keyword queries over large, Web-extracted knowledge bases is presented. One novel aspect is the query language, which only adds a minimal amount of structure compared to pure keyword queries, but still allows users to formulate precise information needs without having to understand the large, heterogeneous schema of the underlying knowledge base. The other aspect is the disambiguation of the keywords in the query with respect to the concepts and relations in the knowledge base. To this end, they model the possible query interpretations as a so-called disambiguation graph, consisting of candidate concepts and relations from the underlying knowledge base. They use a rank-join algorithm to select maximum scoring connected components in this disambiguation graph as interpretations of the query. Similarly to DrillBeyond, with this system the user can pose queries on structured data without knowing the exact schema and contents of the sources. The approach also produces several possible query interpretations and thus multiple results to deal with ambiguity in the user query. It may be worthwhile to apply their method for graph-based query disambiguation to DrillBeyond. If the underlying augmentation system would be based on a knowledge base, then the context knowledge from the user's SQL query and the schema of the local database could be used to form a disambiguation graph as well, allowing mire precise interpretation of the user intent.

Similar work has been done in the area of XML databases, where a large class of work is aimed at enabling ranked retrieval over collections of XML documents. These works search collections where the individual documents are (semi-)structured, but the collection as a whole is assumed to have no common schema. This makes standard structured query processing approaches, for example using XPath or XQuery, inapplicable. An recent overview over this extensive field is given in (Tahraoui et al., 2013). The problems faced in this research are related to those faced by DrillBeyond in that an underspecified, but structured query has to be mapped to a large heterogeneous collection of datasets.

In (Bast and Weber, 2007), a hybrid DB/IR query type called context sensitive prefix search is introduced. Given a set of documents forming a *context*, and a word prefix, this query type aims retrieving documents containing words with the given prefix, conditioned on the words also occurring in the context document set. The authors show how, by introducing further structure through special markup keywords, this abstract operator can be applied to implement many database and expressive information retrieval operations, including joins and aggregation.

Other works aim at processing structured queries based on Web information extraction, e.g., (Cafarella et al., 2007), or even at decomposing structured queries to keyword queries that can be posed to an information retrieval system (Liu et al., 2006). The idea in this case is to enrich the structured query result with relevant documents.

In general, the large amount of related work in DB/IR hybrid technologies shows the necessity of fusing the two paradigms for many applications. By integrating entity augmentation directly into relational query processing, DrillBeyond also aims at supporting scenarios that require a combination of structured and semi-structured data.

3.6.4 Facilitating SQL Query Formulation

By allowing arbitrary keywords in place of formal attribute references, DrillBeyond relaxes the formality of SQL as a language. It enables the user to pose queries that would hard to formulate in SQL in a simple manner. In this aspect, it is related to other works on facilitating the formulation of SQL queries, with the intent of minimizing user effort in analytical scenarios. The largest class of such works, which was inspired by the efforts to consolidate DB and IR systems discussed above, are keyword-based query interfaces for RDBMS. These methods are aimed at transforming free-form keyword queries to valid SQL queries for a given database. Very generally speaking, they are based on mapping keywords to database identifiers, i.e., tables and attributes, and then using the constraints defined in the database to derive possible join paths connecting those identifiers. These approaches are generally divided into schema- and tuple-based approaches. The schema-based approaches model the relations and constraints of the database as a graph, and try to find *candidate networks*, which are connected groups of relations that together contain all keywords of the query. Classic examples for such systems include (Hristidis and Papakonstantinou, 2002) and (Agrawal et al., 2002). The tuple-based approaches, on the other hand, construct a data-graph, where nodes are individual tuples, and then identify minimal connected subtrees in this graph. Notable first works in this category are (Bhalotia et al., 2002) and (He et al., 2007).

In addition to those general approaches, there are also more specialized techniques for OLAP queries. For example, (Wu et al., 2007) tackle the construction of aggregation queries in multi-dimensional databases from key word queries. In (Blunschi et al., 2012), the authors propose a system that uses query patterns and an additional metadata model of the underlying warehouse to improve precision of those methods. While this method requires the definition of an additional conceptual schema describing the business domain, as well as external ontologies such as DBpedia, it can provide much more targeted support for the business users the queried warehouse.

These are just a few selected examples from this area of research. A current overview and

survey can be found in (Bergamaschi et al., 2014), and a comparative evaluation of some of the most well-known techniques was presented (Coffman and Weaver, 2010).

In addition to keyword search approaches, most of which liberate the user completely from any structural requirements in their queries, there are alternative approaches that still process structured SQL queries, but relax the necessary formality. For example, the *Schema-free SQL* approach by (Li et al., 2014a) aims at relaxing the need for exact query specification. Specifically, it enables the users to use imprecise schema terms, i.e., relation and attribute names that are only similar to those actually used in the schema. Additionally, in this system, tedious specification of exact join paths becomes optional. The system is even able to gradually degrade to completely free-form keyword queries, while still preserving many advantages of structured query languages. For example, even without specifying join paths and exact relation names, the user can still specify exact range queries, which would be impossible in a pure keyword system. Their user study shows that users were able to formulate complex queries without exact knowledge of the schema, with lower effort than even with the use of a graphical SQL designer. This is similar in intent to our DrillBeyond system. We also preserve SQL to enable users to exactly specify the form of the desired query result, while freeing them from the effort of exactly specifying data sources used to process the query.

3.6.5 Multi-result Query Processing

So far, we have only discussed works that are related to DrillBeyond with respect to its functionality or intent. However, a large part of our contribution also consists of methods for planning, processing and optimizing the specific type of SQL query processed by DrillBeyond.

One of the characteristics of DrillBeyond is that it produces top-k query results. Superficially, one could therefore assume that the large body of work on processing top-k queries in databases, as surveyed for example in (Ilyas et al., 2008), would be related. However, there is a conceptual distinction. All approaches that are usually grouped under the term *top-k query processing* are concerned with efficiently identifying the k most relevant result tuples to a given query, where relevance is defined by user-defined criteria or scoring functions. These techniques are therefore mainly aimed at increasing efficiency and making results easier to handle by limiting their size, since in many applications a user is interested only in the most relevant answers. Our approach however, produces k variants of a single, complete query result, which differ only in the set of external data sources used to resolve the augmentation attributes. In other words, our approach does not produce a subset of the available data, but a superset. Therefore, existing techniques for "top-k" query processing do not apply.

However, that does not mean that there is not related work with respect to our form of top-k processing. The most related system in this respect is the MCDB system (Jampani et al., 2008), which deals with Monte Carlo simulations in an RDBMS context. MCDB processes queries over databases that contain a mixture of regular single-valued attributes, and probabilistic attributes whose values are drawn from some distribution. More specifically, MCBD allows arbitrary so-called *VG functions*, or variable generating functions, to be stored in the database together with their parametrizations. These functions are used to pseudo-randomly generate concrete values for probabilistic attributes at query processing time. This enables MCDB to generate multiple independent but identically distributed instances of the database,

i.e., possible worlds, process the query of interest over all of them, and then summarize the effects of the uncertainty modeled in the VG functions. This method allows to bring complex statistical models into the database, and enables approximative evaluation of arbitrary queries over them, yielding a highly expressive system.

However, it has similar performance implications as the DrillBeyond model: the query, including all of its deterministic parts, are processed k times, leading to unacceptable performance with a naïve implementation. MCDB uses a different solution than DrillBeyond: It employs a technique called "tuple bundling", in which the variants of a tuple can be bundled into one tuple when the common attributes need to be accessed, and unbundled into its variants when varying attributes to be processed. Efficient bundling is enabled by the pseudo-random nature of the possible worlds embodied in the tuples variants: Tuples variants can be *bundled* by reducing them to the deterministic tuple they were generated from, representing probabilistic attributes by storing only the seed that was used to generate their variants. In addition, a special *isPresent* attribute stores whether a tuple is actually included in each possible world. This allows to represent the fact that regular relational operations, such a selection, may remove tuple instances in some worlds, that then need not to be regenerated on unbundling. From just these two pieces of information, an *instantiate* operator can project the required tuples instances from a tuple bundle when the probabilistic attributes need to be accessed individually.

In some respects, this method is similar to our idea of the ω/Ω splitting of the DrillBeyond operator. In our solution, a set of tuple instances is represented by single tuple for as long as possible as well, while the augmentation attributes' possible values are represented by placeholder values. Those placeholders refer to a set of concrete augmentation values, not unlike the seed values represent a set of possible worlds in MCDB. However, there are some crucial differences between the systems. First, MCDB aims at coping with much higher number of tuple instances for Monte Carlo simulation, for example $N = 1000$, while DrillBeyond aims at human inspectable result lists, for example of length $k = 10$. To cope with this much larger overhead, MCDB requires modified relational operators, such as selection of join, to work directly on bundled tuples. While this enables the higher number of tuple instances, it also introduced a much higher implementation overhead, and diminishes the generality of the resulting RDBMS. A production RDBMS includes many variants of operators, e.g., join implementations, different aggregation operators, but also user-defined functions or extensions. None of these can be reused unchanged, in fact, some can not even be efficiently implemented on top of tuple bundles. Furthermore, other components of the RDBMS, like the optimizer, can not be easily adapted to this completely new query processing approach, but need to be written largely from scratch. In fact, an early version of DrillBeyond was also built on a tuple bundling approach with customized operators. However, we quickly discovered that even after reimplementing many RDBMS operators for tuple bundles, edge cases in query processing remained, e.g., the inability to efficiently sort tuple bundles on bundled attributes.

We therefore used the minimally invasive approach as described in Section 3.3 and 3.4, which allows unchanged reuse of all operators and requires only minimal changes to RDBMS components. It yielded a system that does not loose any of the generality of the original DBMS system we extended, while still yielding acceptable performance for the number of results variants that we aim at in our use case.

3.6.6 Uncertainty and Lineage in Databases

The MCDB approach discussed above is part of a large field of research on modeling uncertainty explicitly in database management system. Many approaches were developed, giving various answers to the central questions of how to represent uncertainty in databases, how to answer queries based on the chosen representation, and how to present the results (Dalvi et al., 2009). While the *possible worlds* model (Dalvi and Suciu, 2007) has been widely accepted as the standard semantics of probabilistic databases, many different representation models have been developed. They range from simple models, where each tuple is associated with a probability of belonging to a randomly chosen world independently of all other tuples, to more complex models with various formalism for describing correlations between tuples (Antova et al., 2009). The trade-off with these methods is usually between expressiveness or theoretical completeness vs. practicability with respect to query evaluation and modeling (Sarma et al., 2006). Which model to choose depends on the application. For example, the uncertainty model used by DrillBeyond corresponds to the comparatively simple model of *or-sets* proposed by (Imielinski et al., 1991). In this model, values are associated with a finite, fixed set of possible values without confidence values. For the DrillBeyond use case, which, for example, does not include any probabilistic inference but simply presents possible worlds, this model is actually sufficient.

Another aspect often tackled together with uncertainty is *lineage* (Benjelloun et al., 2006), which captures the way a result tuple was derived from uncertain base tuples. This aspect is only rudimentary implemented in DrillBeyond through the result set identifiers appended to each query result variant. However, if results produced by DrillBeyond are later materialized in the DB, or used as the basis for further Open World queries with yet more external sources, more sophisticated tracking of lineage may be become a necessity for DrillBeyond.

Furthermore, approaches to uncertainty in databases can be distinguished based on their integration with the DBMS. While many approaches are implemented as database middleware, e.g., (Widom, 2004), there are also approaches that integrate operations on uncertain data tightly with an existing RDBMS core (Huang et al., 2009), similarly to the way DrillBeyond does.

Finally, in addition to representation systems, there has been a wide range of works on efficient evaluation of queries on probabilistic databases. In Section 3.2 we discussed the strongly related MCDB system in detail. For an overview of the many alternative techniques we refer the reader to recent surveys (Li et al., 2013b; Aggarwal and Yu, 2009).

To conclude, we can say that DrillBeyond shares many characteristics with probabilistic databases. However, in DrillBeyond the uncertainty has a much more limited scope. It is never part of the database itself, but is only introduced at query time. Also, it does not consider confidences of the different instances at query processing time, and does not include inference mechanisms. Instead, it aims at efficient processing of an analytical queries based on a small, finite set of possible worlds. Still, as we mentioned, methods for efficient execution, DBMS integration, or tracking or lineage are highly related to DrillBeyond.

3.6.7 Adaptive- And Multi-Query Optimization

One of the challenges of processing Open World SQL queries is the lack of plan-time knowledge about the data sources that will be used to answer the query. Therefore, we utilized ideas from *adaptive-query processing* research, specifically the run-time decision for or against selection pull-up as described in Section 3.4.3, and the run-time reoptimization of the physical plan operators as described in Section 3.4.6. The first technique is a special case of *dynamic query evaluation plans* as proposed in (Graefe and Ward, 1989). In this work, a *choose-plan* operator is introduced, which can dynamically choose different prepared plan fragments based on run-time information such as query variable bindings or intermediate result cardinalities. The ω operator in DrillBeyond offers similar functionality as part of its general augmentation processing. Based on the knowledge about the actual selectivities of predicates on augmentation attributes as returned by the augmentation system, it either directly executes those predicates, or defers their execution until the Ω operator. In addition, based on the difference between the selectivities estimates used for planning and the actual selectivities encountered at runtime, it will also trigger a reoptimization of the unexecuted remainder of the plan. This style of reoptimization is similar for example to work by (Ives et al., 1999), where the query plan is subdivided into *fragments*, or partial plans. During execution, they use event-condition-action rules generated by the optimizer to trigger reoptimizations or toggle certain operators at the end of each query fragment.

While these methods of adaptive query optimization were sufficient for DrillBeyond, there are works proposing much more dynamic systems. The most famous example for a highly adaptive query processing method is the *Eddy* mechanism (Avnur and Hellerstein, 2000), which gives up tree-structured query plans for completely dynamic run-time routing of individual tuples between operators. For a general overview of the area of adaptive query processing, we refer the reader to (Deshpande et al., 2007).

Another major component of the DrillBeyond query processing is the caching of invariant intermediate results to increase the efficiency of repeated query executions. Similar approaches can be found in the area of *multi-query optimization*. The goal of multi-query optimization is to globally optimize the execution several queries, instead of producing locally optimal plans for each individual query as traditional optimizers do. This goal can be reached on several levels: The first is to identify common intermediate results between the individual query plans and use caching to avoid recomputation of those results. The second level is to create a set of plans that is globally optimal, i.e., a set that may contain plans that are locally sub-optimal, but offer better caching opportunities globally. Early works such as (Finkelstein, 1982) and (Sellis, 1988) present exhaustive algorithms to the problem or consider it as a post-processing phase to standard query processing. In (Roy et al., 2000), heuristic algorithms for the problem are integrated with a standard optimizer, improving the practicability of the method. More recent work, such as (Zhou et al., 2007), tries to generalize the problem of identifying and caching common subexpressions, which is used not only in multi-query optimization, but also in materialized view selection and optimization of nested subqueries.

In DrillBeyond, the multiple executions of a query plan can be interpreted as a multi-query optimization problem. In fact, DrillBeyond makes similar trade-offs, as it also creates query plans that are less efficient in a single execution, but offer more caching potential.

3.7 SUMMARY AND DISCUSSION

In this chapter, we presented the *DrillBeyond* system, a hybrid RDBMS / augmentation system, that processes SQL queries in which users may use arbitrary attributes in queries, that need not to be defined in the database. DrillBeyond processes these queries by tightly integrating a top-k entity augmentation system, that searches a large corpus of Web data sources for possible ways to augment the local relations with the missing attributes. To solve the uncertainty and ambiguity of such an automated integration process, DrillBeyond answers these queries with multiple alternative query results, each based on different external data sources.

We discussed integration challenges and design choices when integrating the two types of systems, and proposed various optimization techniques to minimize the overhead of processing multi-result SQL queries. The system was implemented in PostgreSQL and evaluated it on modified TPC-H queries. We demonstrated the effectiveness of various optimizations in minimizing the runtime overhead of producing multiple SQL query results based on alternative augmentations. Finally, we showed that pushing SQL query context into the EA system can improve quality and performance of the EA processing compared to standalone processing.

To conclude this chapter, we will review the challenges we identified in Section 3.1. First, by integrating augmentation capabilities directly with the DBMS, we eliminated the problem of *Context Switching*. Analyzing data as well as searching for and integrating external data can now be performed in one system. This is especially valuable, as it enables typical BI tools or other DBMS front ends to easily integrate entity augmentation with their features. Second, we bridged the *Incompatible Query Models* of the two system types by integrating entity augmentation directly with the declarative query specification in RDBMS, and by enabling the efficient processing of top-k results in the relational context. Also, the *Loss of Context Information* we discussed is alleviated by DrillBeyond's explicit consideration of query context. The system optimizes the positioning of the augmentation operators with respect to maximizing the available query context. It then pushes this context to the augmentation system, implicitly through the set of tuples reaching the operator, and explicitly through sending query hints such as data types and predicates of the augmented attributes. Finally, the various optimization strategies we proposed, such as selection pull-up or partial selection, make use of *Unused Optimization Potential*. By using run-time knowledge of the augmented attributes' selectivities or the correlation between data sources, query plans can be optimized in ways that are impossible with separate systems.

To summarize, our DrillBeyond system, with its tight integration of augmentation and relational query processing and its various optimizations, enables the use of ad-hoc data search and integration in new contexts, and greatly increase the practicality of the augmentation methods we introduced in Chapter 2.

PUBLISH-TIME DATA INTEGRATION

In the previous two chapters, we discussed novel methods for answering entity augmentation queries based on large corpora of heterogeneous sources. So far, we made no further assumptions about the nature of these corpora, or about the systems that manage them. However, the quality of the augmentation results depend strongly on the quality of the data sources used. Therefore, it is worthwhile to investigate the types of systems used to manage such large-scale collections of disparate data sets.

In fact, schema-less, collaborative and multi-domain dataset repositories are becoming more prevalent in many areas. One example for platforms of this kind are Open Data Platforms, such as `data.gov` or `data.gov.uk`, where government agencies publish datasets of public interest, allowing enterprises and citizens to reuse the data. The published datasets stem from a large number of domains, various authors, and come in a variety of formats and levels of quality. In addition to those public data repositories, commercial data marketplaces with similar properties are also emerging (Schomm et al., 2013). A second example would be scientific data management, where it is common to publish flat files containing raw measurement data, at best accompanied by textual metadata, for sharing with other institutions and researchers (Gray et al., 2005; Ailamaki et al., 2010). A final example are the central Hadoop clusters used in a growing number of enterprises which are commonly used as *data lakes*, i.e., repositories in which data from all kinds of not-yet integrated sources in the enterprise is collected for potential future reuse (Mohanty et al., 2015; O'Leary, 2014).

These repositories are characterized by a lack of centrally enforced schemata or fixed input and output pipelines, and often contain heterogeneous, mostly unconnected datasets from disparate sources and different domains. The essential idea is to provide a centralized hub that offers only basic services such as redundant distributed storage for all managed data, but which in turn is open to all types of data that accumulate in an organization. The goal of such a platform is to enable data publishing and reuse with flexible, but scalable effort, e.g., from pure dumping of server log files to publishing single datasets with their own defined schema and metadata, but not necessarily any connection to other datasets. This makes them a contrasting, but complementary concept to data warehouses, where rigid standards and processes have to be complied with. Since organizations now typically are confronted with a higher variety and velocity of datasets than they can manage and integrate immediately, such open repositories become more attractive alternative to either discarding data or integrating everything upfront. Furthermore, with more and more data, the potential for future usage is not always clear initially, so investment in a particular dataset is hard to justify. Finally, with the increasing variety of data, keeping a centralized schema up-to-date becomes infeasible. More generally, curating large collections of datasets, i.e., facilitating their retrieval, reuse and integration can not be achieved by manual human effort alone, but must be supported by automated support systems (Stonebraker et al., 2013).

Another important property of these platforms is their collaborative character. Many parties typically independently publish to the same system, leading to further difficulties in managing the data. For example, in the case of an internal data lake, many departments and many individual analysts may use the same platform, while in the governmental case, many agencies and officials may have their own conceptions of how to publish data. This trend is not limited to public organizations. Even enterprises have started to turn to community-driven forms of data management, that utilize crowd-sourcing and semi-automatic tools (Curry et al., 2010).

The free-for-all nature of these data publishing platforms leads to a strong heterogeneity in the corpora managed by these platforms, which contradicts the primary goals: data reusability. All in all, although useful for their respective purposes, such free-for-all data management systems also create new challenges for their users.

In this chapter, we will study these platforms, which we will call data curation systems, as opposed to, for instance, *databases* or *data warehouses*. We identify their common properties and current weaknesses, and propose an improved repository architecture with services designed to mitigate them, centered around our proposed concept of *Publish-time Data Integration* in Section 4.1. After introducing the general concept, we will discuss two specific instantiations, *attribute name recommendation* in Section 4.3 and *extraction of relational data from partially structured documents* in Section 4.4. In Section 4.5, we will survey other works related to data curation systems in general, and to our two specific methods in particular. We will discuss and summarize our results in Section 4.6.

But at first, to create a more detailed picture of these challenges, we will study existing Open Data platforms in detail in the next section. As mentioned these platforms are a good, publicly available example of a data curation system, and allow us to study their properties and usage in the real world. The findings will be used to motivate and design our concept of a *Publish-time Data Integration* platform in Section 4.2.1. We will both quantify the potential usefulness of these platforms, but also their current drawbacks, hindering quick and efficient reuse of the published data.

4.1 A STUDY OF EXISTING OPEN DATA PLATFORMS

Following the *Open Data* trend, governments and public agencies have started making their data available to the public using Web portals, Web services or REST interfaces. Ideally, making this data available on the Web would lead to more transparency, participation and innovation throughout society. However, just publishing the data on the Web is not enough. To truly enable efficient reuse of the data by third parties, the publication platforms need to fulfill certain administrative as well as technical requirements. To study to which degree current platforms facilitate data reuse, we studied over fifty Open Data repositories operated by national, regional and municipal governments, as well as international organizations. Features such as standardization, discoverability and machine-readability of data were taken into account. Furthermore, a subset of five repositories was examined in more detail, additionally analyzing data and metadata quality. We introduce a number of aspects of *reusability* in order to classify the surveyed repositories and assess the state of Open Data on the Web.

Data published on existing Open Data platforms covers a wide range of domains, from environmental data over employment statistics to the budgets of municipalities. Publishers can be individual government agencies or providers of larger repositories that collect public datasets and make them available in a centralized and possibly standardized way. Though set up with the same goal in mind, these repositories show very different characteristics. They vary, for example, in size, domains, comprehensiveness, or the application of technical standards. But most importantly, we argue that they vary in their usefulness and suitability to their task.

We present our survey of existing Open Data repositories in Sections 4.1.1 to 4.1.1. We

then classify the studied repositories and illustrate these classes with our study results in Section 4.1.2, and discuss our survery result's implications for the study of data curation systems in the rest of the chapter in Section 4.1.3.

Parts of the material presented in this section have already been published in (Braunschweig et al., 2012).

4.1.1 Structure and Methodology

We surveyed fifty Open Data repositories operated by national, regional and municipal governments, as well as international organizations. We based our list of repositories on a catalog published by the Open Knowledge Foundation[1]. Then, we expanded and refined this list using the following steps:

1. We diversified the list by adding more repositories from all parts of the world, including repositories from all five continents.
2. We took care to add regional and municipal repositories in addition to national initiatives.
3. We included both official, i.e., state- or agency sponsored platforms, as well as community-driven efforts.
4. We removed very small or inactive repositories, except for countries for which we could not find other repositories.

The full list can be found in the appendix and on the survey website[2]. It forms the basis of our global assessment of open data repositories, which is presented in Section 4.1.1. In addition to that, a subset of five repositories was examined in more detail, which means for those we include further statistics that require more manual effort to collect. The results of this detailed analysis are presented in Section 4.1.1.

Global View

For the first part of the survey, we studied features of open data platforms that can be measured by just browsing the page or by writing a simple crawler for platforms that do not provide statistics on their content. More complex features that require downloading and analyzing the available datasets were studied for a limited number of platforms in the second part of our survey (see Section 4.1.1).

The goal of the global survey is to assess each repository's suitability for automatic reuse of the data in general. The specific features that we surveyed will be described below. If not stated otherwise, the feature values were measured manually.

Number of published datasets Almost all platforms have the notion of a dataset: an independent unit of data, describing one special topic or aspect of the world. One of the basic questions for an Open Data repository is: does it provide any interesting datasets? Although

[1] `http://lod2.okfn.org/eu-data-catalogues/`
[2] `http://wwwdb.inf.tu-dresden.de/opendatasurvey/`

the absolute number of datasets can easily be skewed, a repository with more datasets is still more likely to useful data. Many sites provide the number of published datasets, and for all others, we measured by writing a simple Web crawlers to count them.

Standardized file formats Offering data in standardized file formats makes reuse much easier because it is immediately apparent from the metadata how to process the published files. In contrast, platforms allowing upload of any file format will always require manual processing to enable reuse of the data sets. We measured whether the platform regulates file formats.

Standardized metadata attributes Many platforms provide the option of adding metadata to datasets. However, for automatic processing of the datasets, they should have a set of standardized attributes, which can be looked up for any dataset. Special cases of metadata attributes that we survey separately are the existence of *domains*, i.e., some form of dataset grouping, which is help for retrieval and discovery of related datasets. Furthermore, we examined whether the platforms offer standardized temporal and spatial metadata attributes. These are especially useful, as the majority of datasets only describe specific geographic entities and/or time intervals. Reuse of these datasets is facilitated when the respective spatial or temporal information is provided as metadata.

API This should be an obvious feature of every Open Data platform. Without an API, the platform can only be accessed manually through its Web interface, preventing automated discovery and reuse. Furthermore, we differentiated between APIs that offers access to metadata, while keeping the data in downloadable files, and those providing direct access to the data through the API. A uniform data access saves users the effort of building parsers for individual datasets. This usually requires that the raw data is stored in a database management system.

Curation A curated platform is not necessarily better than a public editing platform. We included this feature in the study to differentiate between platforms run by a governmental agency, which are usually curated, and platforms driven by an interest group, which usually allow unlimited dataset upload.

For every platform, we also recorded the country of origin, the administrative level it represents (country, region, city, etc.), as well as policies regarding licensing, up-to-dateness and provenance.

Detailed Analysis

In addition to the global analysis, five repositories where surveyed in more detail. This detailed analysis was performed for the *data.gov* (US), *opendata.ke.gov* (Kenya) and *data.gov.uk* (UK) repositories as well as the global repositories *data.worldbank.org* of the Worldbank and *data.un.org* of the United Nations. For this analysis, we surveyed additional features that require more manual effort to gather.

Downloadable Datasets The fraction of the available datasets where we succeeded in downloading actual files. One of our first observations was that many of the offered datasets did

Country	Repository	Number Of Datasets	Last Visited Date	Down-loadable datasets	Machine-readable datasets	Avg. numbe of tags	Tagged datasets	Length of description	Datasets with description
United Kingdom	data.gov.uk	7.439	07.09.11	79.9%	42.20%	5.9	93.1%	349	92.4%
United States	data.gov	4.941	05.10.11	78.6%	77%	22.2	99.6%	381	99.8%
Kenya	opendata.go.ke	4.53	06.10.11	100.0%	100%	1.8	70.9%	67	80.8%
Worldbank	data.worldbank.org	5.500	04.11.11	100.0%	100%	0.0	0.0%	145	57.5%
UN	data.un.org	5.413	04.11.11	100.0%	100%	0.0	0.0%	0	0.0%

Table 4.1: Results of the detailed repository analysis

not include working downloads: many of the provided links were dead, returning HTTP error codes or timing out on request. Other links led not to the data, but to HTML documents containing links to the data, or sometimes just the home page of the organization distributing the data. In both cases, manual browsing is necessary to obtain the actual data. Of course, this prevents automatic retrieval and thus automatic search and analysis on the data. We measured this feature by obtaining all available download links from the respective platform's API and following them. If there was a valid HTTP response, and the content was not in HTML format, we counted the dataset as downloadable.

Machine-readable Datasets The fraction of the downloadable datasets where we succeeded in opening the downloaded files using standard, of-the-shelf parsers for the respective file formats. In our research on Open Data, we quickly noticed that a great share of the datasets are in obscure or proprietary formats, or not in the format given in the metadata. Obviously, this prevents automatic searching and processing of the data. We measured this feature by looking at the file type specified in the metadata and applying a standard parser for the respective format. If the file could not be parsed using this method, we counted it as not machine-readable.

Tags and Description Almost all surveyed platforms use tags to make datasets discoverable and searchable. However, tags are only a useful instrument of navigation if they are applied to all datasets, in an adequate number and without repeating generic tags for many datasets. We counted the average number of individual tags per dataset. Furthermore, all surveyed platforms offer some textual information describing the dataset. We counted the average number of characters in the description of a dataset.

An overview of the results of the detailed survey are shown in Table 4.1, while the complete results, global and detailed, can be found at the survey website[3]. The next section will interpret and discuss the results, and derive a classification of Open Data Platforms. We will discuss the properties of these classes and support them using our data.

4.1.2 Platform Classification and Survey Results

In the course of our study, we identified several dimensions in which Open Data platforms can be classified. Figure 4.1 illustrates the four dimensions and possible values for each one. Open Data platforms can be split according to the technical implementation of the platform,

[3] `https://wwwdb.inf.tu-dresden.de/opendatasurvey/`

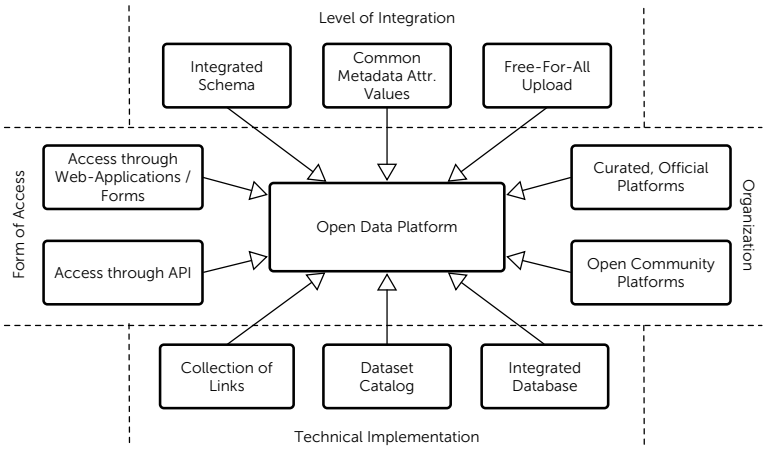


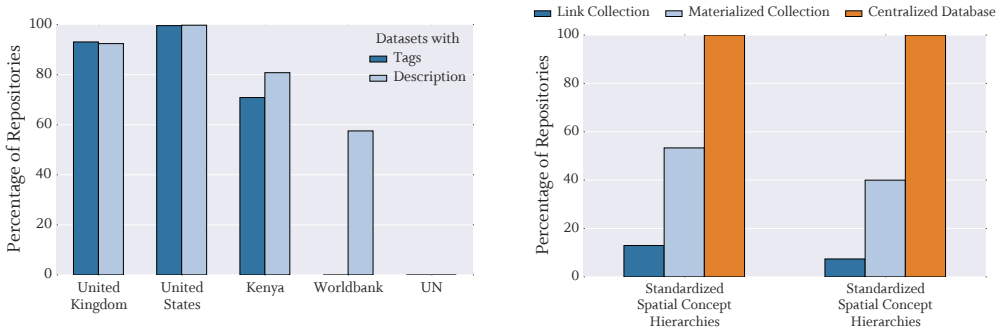
Figure 4.1: Classification of Open Data Platforms

the forms of access that are allowed, the level of metadata standardization, and the platform's organizational form. We will now classify the surveyed repositories according to these four dimensions and present the survey data in this context.

Technical Implementation Concerning the technical implementation, the most common type is *collection of links*, which accounts for 55% of the surveyed platforms. These platforms host only metadata, and store URLs as the only way of accessing data. This class is not only the largest, but also generally the least useful one. As our detailed analysis of `data.gov` and `data.gov.uk` showed (see Figure 4.1.1), large percentages of these links do not resolve to a working file download. Apart from the dead-link issue, our survey also shows that these platforms have a lower level of standardization and integration between datasets. For example, only 20% of the link collections feature standardized file formats, while the rate over all surveyed platforms is 42%. Generally speaking, leaving the actual data with a multitude of different providers leads to a lower data reusability. We argue that this is due to the distribution of the data over many different spheres of influence, each with its own standards, formats, and levels of dedication to Open Data.

The second largest class regarding technical implementations is *download catalogs* consisting of 29% of the surveyed platforms. In contrast to link collections, these platforms also host the files, instead of just the links. Interestingly, platforms implementing this simple concept have a much higher reusability, e.g., 80% of the download catalogs have standardized file formats. The other features related to standardization show similar results. A possible explanation is that these platforms might operate with greater personnel or financial resources, allowing more integration and higher technical sophistication.

With 8%, the *integrated databases* class is the smallest one within our classification. These platforms do not operate on the level of individual datasets or files, but offer integrated datasets using a database management system. This difference in storage manifests in the actual Web platform through the ability to filter and query datasets. Most of these platforms use relational data, but Linked Data (e.g. SPARQL endpoints) is used as well. In contrast to catalogs and es-



(a) Metadata Availability

(b) Spatial and Temporal Metadata, by type

Figure 4.2: Capabilities of Open Data Repositories

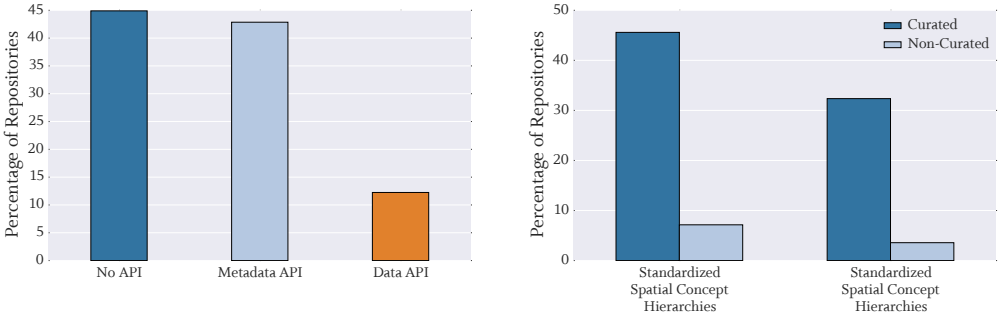
pecially link collections, these platforms achieve 100% downloadable datasets as well as 100% machine readable datasets. Even though these repositories offer the highest reusability of data, they also have the lowest diversity of available data sets. This is due to the fixed relational schemata they employ: *data.worldbank.org* and *data.un.org*, the two main representatives of this category, both offer only statistical values that are always relative to a certain combination of country and year.

One important distinction between these types of platform is the availability of metadata and description. In comparison to integrated databases, dataset catalogs and link collections less frequently offer spatial or temporal metadata, but offer more often feature individual dataset descriptions and tags (see Figure 4.2). A possible interpretation could be that in integrated databases, the semantics of the data is defined in the global schema, which makes spatial and temporal data available explicitly, while in collection- and catalog-type platforms, textual descriptions are used to make for the lack of schema and the higher diversity.

Form of Access All surveyed platforms offer a *Web interface* for finding and downloading datasets. While an API seems like a mandatory part of an Open Data Web site, 43% of them do not feature an API, but instead allow only manual download using HTML listings of datasets. Practically all these sites contain a search functionality, but this search it is often confined to search dataset titles or textual descriptions.

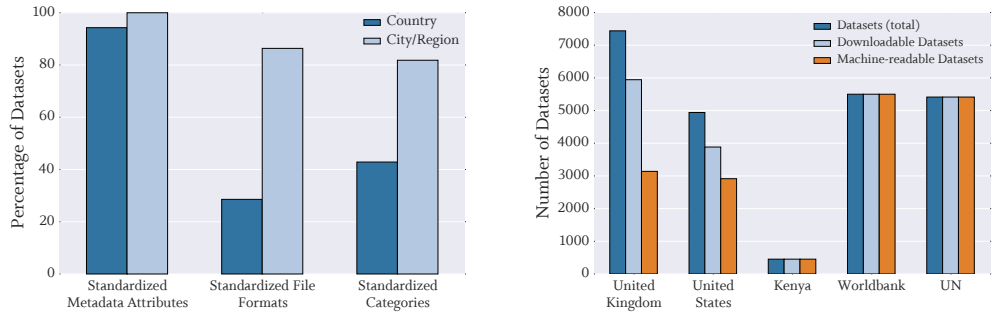
About 57% of surveyed platforms feature an *API*, although their capabilities vary, as Figure 4.3a shows. Generally, APIs can be classified in those that only allow to retrieve metadata such as dataset titles, categories, and download links, and those that allow to access the data directly. On first glance, it surprises that there are so many platforms that have an API, but no direct data access. This is mainly due to the prevalence of link collections over download catalogs and databases.

Organization We differentiate between curated and non-curated platforms. Starting with the *non-curated* ones, we observed that these platforms are most often not run by institutions or agencies, but by a community that collects sources of open data or raw datasets. In contrast to curated repositories, they offer facilities for uploading datasets that can be used by everyone. The *curated* platforms, however, do not allow uploads by users. They are moderated and usu-



(a) API capabilities, by repository type (b) Spatial and Temporal Metadata, by curation

Figure 4.3: Metadata Capabilities of Open Data Repositories



(a) Standardization by Level (b) Accessability of Datasets

Figure 4.4: API Capabilities and Platform Standardization

ally run by some public institution. The lack of a central authority in non-curated platforms usually leads to less structured repositories, as Figure 4.3b shows for the example of spatial and temporal metadata. Another difference is average size: while the average number of datasets for curated repositories is about 1.600, it is only about 450 for non-curated repositories.

Metadata Standardization The first category we observed are *Free-for-All* platforms. This category encompasses all repositories that have no regulations concerning datasets that are published. Repositories in this category do not have features such as standardized file formats, or spatial or temporal metadata dimensions. Of the surveyed platforms, 43% do not even have one of those features.

The next category contains repositories which apply restrictions to the datasets or metadata that can be attached to them, i.e., they feature *common metadata values*. Examples would be restricting the possible file formats, or requesting standardized date formats. When considering the standardization features of the survey, i.e., standardized file formats, spatial and temporal metadata, and standardized domains, only 10% of surveyed repositories have all four features, while another 20% have three of them.

The final class, platforms with *integrated schemata*, offers the highest degree of integration. The platforms that we categorized as centralized databases usually have an underlying rela-

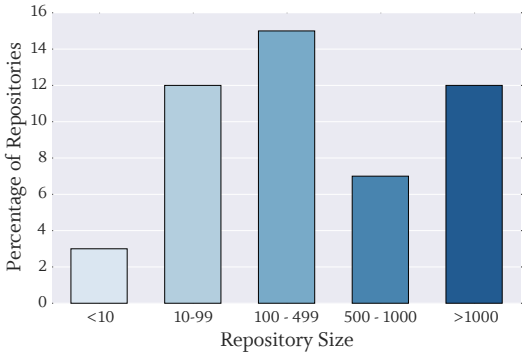


Figure 4.5: Size of Repositories

tional schema, even it is not explicitly stated. Platforms like data.worldbank.org and data.un.org use the same combination of attributes for a large number of datasets, usually mapping from a country and a year to an attribute value.

General Observations Finally, we want to discuss some general observations that cross-cut the repository classes discussed above. Firstly, *repository sizes* vary considerably in our study, as shown in Figure 4.5. The largest repositories are in general those of larger nations and international organizations, reaching up to 4,900 for data.gov and 7,400 for data.gov.uk, while many countries have only small community driven platforms with sometimes less than 100 datasets. Second, we discovered that *locality* matters: Local platforms, operated by municipalities or states, performed better regarding standardization than national repositories (see Figure 4.4a). It seems that platforms focusing on a more limited domain can more easily enforce constraints for datasets. The users of local platforms also seem to be more willing to enhance datasets by adding tags or descriptions. The larger platforms try to be universal, and end up being more generic. Lastly, two major problems are common to almost all platforms: dead links and a plethora of different file formats. The data from the detailed analysis concerning data.gov and data.gov.uk illustrates these problems (Figure 4.4b). In both cases, only about 79% of the metadata entries on these two platforms could be used to download actual files. Some links were dead, while others led to HTML pages or Web applications where the data might be found. With regard to our goals of machine readability and reusability, those entries can not be seen as useful open datasets. But even from the downloaded files, only 77% in the case of data.gov and only 42% in the case of data.gov.uk could be opened with a set of standard parsers. The difference between these two platforms can be explained by the fact that data.gov uses a small set of standardized file formats, while the resources of data.gov.uk have no limitations regarding the file format. We counted 32 different formats in the data.gov.uk metadata, the overwhelming majority of datasets being marked as “no-format”.

4.1.3 Conclusion

Over the years, public and governmental institutions have generated and collected vast amounts of data which is locked within heterogeneous, proprietary and undocumented data files. Within

the Open Government initiatives that emerged recently, municipals and agencies have started making this data available to the public in order to achieve transparency, participation and innovation throughout society. Since these platforms are a good example of data curation systems, we surveyed the top 50 of them, and benchmarked these repositories in terms of reusability. Furthermore, we analyzed a subset of five repositories in terms of data and metadata quality. Our analysis showed that a majority of the platforms lack of proper standards and APIs, and have a lot of data published that is not machine-readable or in a proprietary format. This prevents effective reuse of the published data, at least without considerable human manual effort. On the other hand, we can also see that those platforms that impose rigid, pre-defined schemata such as `data.un.org`, while having high quality data with high reusability, also offer the most limited breadth of datasets. In the UN data example, all datasets are indicators about countries, grouped by year, with no other schemata allowed. The main finding of the study with respect to data curation systems in general is that there is a strong contradiction between two goals of such systems, namely easy *free-for-all* publication of all kinds of data, versus easy *reusability* of the managed data in new context.

Note, that while in this thesis we did not survey enterprise or scientific data curation systems, results of other research, such as (Stonebraker et al., 2013), align well with our results. Therefore, without loss of generality, we will use our findings on Open Data platforms to motivate our concept of a data curation system that enables efficient data reuse, while keeping the publication process simple and free-for-all.

4.2 PUBLISH-TIME DATA INTEGRATION

This section will first define requirements for a data curation system that facilitates data reuse in Section 4.2.1. We will then introduce our concept of *Publish-time Data Integration* in Section 4.2.2.

4.2.1 Improving Reusability in Data Curation Systems

As we have shown in the previous sections, the data curation systems we study have two conflicting goals. The first is that they should have as little limitations as possible on what can be published, allowing in many formats, from different users, domains and with different degree of schema, i.e., they should be *free-for-all*. This also means that users should be able to invest as much effort into their dataset as they want, including no effort at all besides pure storage of the raw data. On the other hand, they should be optimized for *data reusability*, i.e., make it as easy as possible for a user to retrieve datasets fitting a specific information need, and allow processing of the data with minimal effort. So, while following the first goal, a user would be encouraged to store, for example, a small dataset collected in an isolated experiment in the form of a spreadsheet file, as this is a format most users are likely to be familiar with. In consequence of the second goal, however, the data should be extracted from the spreadsheet into a format that separates data and metadata and has an explicit schema. In the best case, it should even include references to related datasets in the repository. It is immediately apparent that the

mentioned goals, high reusability and strong integration on the one side, versus large quantity of datasets and large number of contributors on the other side, are contradictory.

Since limiting publication or imposing too many rules on users can not be the solution in the context of data curation systems, new approaches to data integration will be necessary to cope with the problems of free-for-all data platforms. So, what are the main differences between traditional data integration and the kind of integration we aim for with data curation systems? Traditional data integration is goal oriented: integration tasks are specified for a defined set of data sources and defined application or analysis goal. In contrast, data on Open Data Platforms would have to be integrated for the sake of having integrated data, as the possible reuse scenarios are not known beforehand. Furthermore, in classical data integration usually considers one-to-one integration between two well-defined schemata that describe the same domain. A data curation system, however, contains a large number of mostly unrelated datasets that have very few corresponding attributes on average. Still, subsets of them describe the same domains and could be reused together if they were properly consolidated. So while there is reuse and recombination potential, making all the different datasets obey to a global schema is unfeasible. This view of data curation systems is similar to the concept of dataspaces (Franklin et al., 2005). The current philosophy in working with such dataspaces is the so-called pay-as-you-go approach (Madhavan et al., 2007), in which integration is postponed until it is clear how the data should be reused (see Section 4.5 for detailed discussion of related work. Still, we argue that some integration tasks can and should be tackled a priori, to improve the usefulness of data curation systems as a whole. Specifically, we identify the need for developing tools and services for enriching data curation systems, which should be aimed at fulfilling the following requirements.

Standardization without Schema In typical integration scenario, a global schema is defined, to which a set of databases can be mapped. In those scenarios, the databases to be integrated cover the same domain, are moderate in number, and usually under a central administration. For data curation systems, there can be no single central schema: they cover many domains, are used by many users, and contain a large number of datasets. There is usually no incentive or not enough resources, i.e., personnel and time, to create and maintain a global schema. The data curation system should therefore be able to foster commonalities between published datasets without an explicit target schema.

Integration before Reuse Traditionally, data integration techniques are applied at what we call *reuse-time*, i.e., when the datasets to be integrated and the context in which they are to be used are known. In contrast, we aim at improving a dataset's reusability before the context of reuse is known, but without a central schema to integrate with. This means the data curation system should aim at removing integration obstacles within single datasets that are general and independent of a specific reuse case. But it should also aim at keeping the usability of the corpus of published datasets as a whole high, e.g., by preventing the growth of heterogeneity in the corpus over time.

Minimal requirements to the user Data integration is typically the domain of experts, people who are skilled with various data producing and consuming systems, and the tools used

to create mappings and transformations between them. This is due to the quality standards that have to be adhered to in a traditional integration scenarios, such as warehouse architectures. In contrast, data curation systems aim at enabling data publishing and management for all kinds of users with different backgrounds, and different demands with regard to their data management tools.

Adaptation to Evolving Usage The corpora managed by data curation systems are evolving constructs. Since no usage pattern or domain is prescribed, the way the repository is used, and the kinds of data that are managed will vary over time, e.g., according to changing business needs in its environment. For example, the schema vocabulary used to on a given data curation system may change over time, because source systems are added, or whole new types of data are stored. A data curation system should be able to adapt its services to such changes automatically.

Scalable integration We already mentioned that the data curation system we envision should make minimal demands with respect to its users. However, that does not mean that it should by minimal with respect to features. In contrast, it should provide various integration features and tools that can be applied independently and also gradually over time, thus scaling with the effort a specific user is willing to spend.

In the next section, we will define a specific architecture that is able to cope with these requirements.

4.2.2 Principles of Publish-time Data Integration

To create a data curation system that fulfills the requirements we derived in the previous section, we will now introduce our proposed *Publish-time Data Integration* paradigm. The concept is driven by the realization that with data curation system, dataset publishers and dataset users are decoupled. For example, on an Open Data platform, a government agency publishes a dataset of demographical data concerning a certain area of their jurisdiction. The future usage of the data is unknown to them. At a later point in time, the data is to be reused in a new context, e.g., by a journalist using the data to support a story. Integration with other data is complicated because the publisher's knowledge of the finer semantics or intricacies of the dataset is lost to the reuser. For example, the vocabulary or abbreviations used may be very specific to the publishing agency, and may loose their meaning when viewed by a third party user. Another example would be implied context information, such as the way the data was gathered, or constraints of its validity. This information may be known to the original dataset creators, but is lost in the publication process. Even more, if the dataset was lacking proper metadata and description, it may be difficult or impossible to find at all. Our central insight is therefore, that to enable efficient and effective data reuse, it is necessary to capture, formalize and make available as much knowledge of the publisher at the time it is still available, i.e., at *publish-time*.

However, in many cases, forcing publishers to adhere to strict publication standard will not be possible or not even be beneficial. On the one hand, it may be impossible because

central guide are hard to define for a data curation system with many domains and participants. Further, capacity to enforce these standards may be limited. On the other hand, it may even stifle usage of the platform if publishers are forced to invest to much time and energy, or learn to many guidelines before being able to make a dataset available.

We therefore propose to combine a stack of fully automatic integration methods with the human judgment of the publisher itself to offer lightweight integration services for data curation systems. In general, our concept, called *Publish-time Data Integration*, is characterized by the following properties.

- **Automated Creation of Integration Recommendations**: Instead of forcing the user to apply methods of data integration or cleaning, the data curation system uses automated methods. Since data integration is known to be not fully automatable, we use these methods to create recommendations on how to improve the reusability of a dataset that is being published.

- **Usage of Platform Statistics**: While data integration with data curation systems brings new challenges, it also offers new possibilities. Specifically, the large amount of datasets available enables standardization even with no explicit global schema defined. To this end, similar datasets already managed by the repository can be used as guidelines or templates for newly published datasets. By aligning new datasets with existing ones, the growth of heterogeneity in the data curation system can be managed.

- **Simple User Decisions**: If the publisher is confronted with complex choices, it is likely that the automatically generated integration recommendations will not be considered. The recommendations generated by our are methods should therefore only require minimal user action. The most preferable form are simple yes/no decisions.

- **Top-K Recommendations**: Publish-time Data Integration should not only produce recommendations, but produce a ranked list of alternative recommendations. This has the advantage of involving users in the resolution of naturally occurring ambiguities in data integration, by letting them choose from a list of possible improvements.

- **Interactive Latency**: A similar argument as with decision simplicity can be made for latency of the proposal generation: if the publisher has to wait to long for recommendations to be made, they will likely not be considered to. So while traditional integration is an offline process where latency is usually a lesser concern, it is much more important for Publish-time integration.

Having established the guiding principles for Publish-time Data Integration, or *PTDI*, we will now discuss how to design a PTDI-enabled data curation system in more detail. Consider Figure 4.6, which gives an high-level overview of our proposed architecture. The repositories purpose is the management of a loosely coupled corpus of heterogeneous datasets, depicted in the lower left of Figure 4.6. Central to the platform is the Publish-time integration system, which implements *PTDI-operators*, i.e., techniques that, given a newly published dataset DS_+ and with respect to the context established by the existing datasets, will create one or

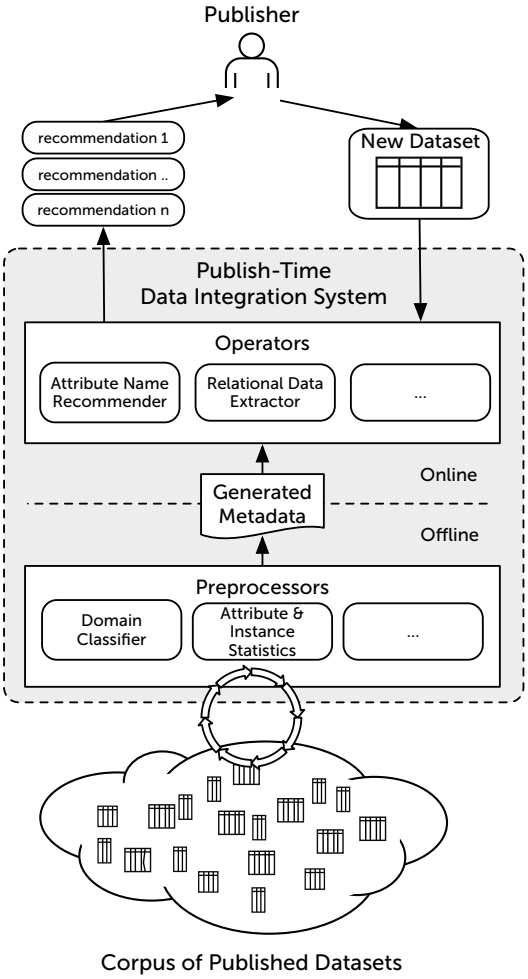


Figure 4.6: Architecture of a PTDI Platform

more recommendations for the user's consideration. We will discuss specific operators in Section 4.2.3. As mentioned, PTDI operators use statistics of the platform's content to guide proposal generation instead of using global schemata. These statistics, as well as other additional metadata, is generated offline using preprocessors that prepare metadata for the online recommendations generation of the PTDI operators. For example, in Section 4.3.2 we will discuss a preprocessing step that clusters the existing datasets into domains, i.e., groups of datasets with related schemata. Continuing the example, these groupings are used by the *attribute name recommender*, described in Section 4.3, to improve both runtime and precision of its results.

With the architecture for a PTDI-enabled data curation system in place, we will now discuss the functional core of the concept, the *PTDI operators*.

4.2.3 Publish-time Data Integration Operators

In general terms, a Publish-time Data Integration operator creates change recommendations that increase the reusability potential of a dataset. These changes may affect, in general, all parts of a dataset, both metadata and data, as well as the technical format it is stored in. While the generated recommendations may effect only one dataset, such an operator still may take the whole platform content into account to make its decision. For example, the *attribute name recommendations* operator we introduce in Section 4.3, proposes homogenizing changes to a dataset's attribute names by comparing the naming with similar datasets on the platform. Its basic idea is to advise publishers to use schema vocabulary that is already in use in the data curation system to which they publish. Its recommendations, however, still only affect a single dataset.

One might argue that global operations affecting multiple datasets may lead to better decisions than a chain of independent local steps. In fact, such global operations could be integrated with data curation systems. However, we maintain that local change recommendations are if higher importance for Publish-time Data Integration. Due to volume and diversity of the datasets stored, users of data curation systems can, in the general case, not be expected to care about the global usefulness of the managed corpus. On the contrary, clearly scoped local operations that require minimal effort, have a higher chance of being considered by publishers. Still, global changes to a data curation system's content over time, after applying a long string of local changes, are an interesting and worthwhile research topic. In this thesis, however, we leave it as future work discussed in Section 5.2.

Instead, we will describe two local PTDI operators in more detail, as proof-of-concept of the general paradigm. These two examples will illustrate how the principles of Publish-time Data Integration can be instantiated, and will be used to study their effectiveness on real Open Data platforms. The first is, as already mentioned, *attribute name recommendation*, in which recommendations for alternative attribute names are generated using statistics on attribute name usage in the existing platform content. The intent is to prevent needles growth of schema vocabulary when new datasets are published, without resorting to a global schema. From our observations of existing Open Data platforms, we found that the large number of authors on one platform leads to a large schema vocabulary, i.e., usage of different terms for equivalent attributes. Though this is the classic problem of schema matching, we approach it at publish-time instead of integration- or reuse-time. In Section 4.3 we describe our approach to this problem, and evaluate it on datasets based on the Open Data Platform (`opendata.socrata.com`) and relations extracted from Wikipedia.

The second operator is *extraction of relational data from partially-structured documents*. From our Open Data platform studies, we know that most data is not published in easily machine-readable and thus easily reusable formats. Instead, large quantities of interesting datasets are actually uploaded as raw spreadsheets. While spreadsheets are easy to create and manipulate for a business user, and are also self-documenting because they allow inline text and have many formatting options, they are not optimal for reuse because they conflate layout, metadata and data in one non-standardized format. In Section 4.4, we study `data.gov`, the US federal Open Data repository, as an example source for partially structured documents, and present a classification of typical problems in real-world spreadsheets that complicate reuse. We then

describe the *DeExcelerator*, a system concerned with automatic spreadsheet-normalization, i.e., extraction and separation metadata and relational content of spreadsheets. It produces purely relational data, e.g. an SQL schema and insert statements, plus additionally extracted metadata from a given partially-structured document such as an Excel file. We evaluate the quality of its extraction pipeline on real-world spreadsheets from `data.gov`.

Finally, in Section 4.5, we will study related work, show how it relates to Publish-time Data Integration, and how it could be applied in the context of a PTDI-enabled data curation system.

4.3 ATTRIBUTE NAME RECOMMENDATIONS

We have discussed that the free-for-all nature of data curation systems leads to strong heterogeneity in the corpora managed by these platforms. One specific problem of this heterogeneity is the schema vocabulary, i.e., the terms used as attribute names of the datasets on the platform. Since the data is published by various authors using different terms for the same concepts, this leads to the classic data integration problem of finding equivalences between attributes in different datasets.

While the conventional techniques of schema matching are in principle applicable to all of these problems, they are usually designed to be used at *reuse-time*, i.e., when it is clear which datasets should be integrated and reused together. These techniques will not help to prevent the increase of heterogeneity in repositories, where a growing number of authors publish a rising number of datasets.

One solution would be to force newly published datasets to conform to a centralized schema or ontology, in which case schema matching techniques could be applied. While this may be a viable approach for some repositories, enforcing or even just creating an integrated schema will be unfeasible for multi-domain repositories with authors acting totally independent of each other. Additionally, this would limit the number and diversity of published datasets, since the publishing effort would increase. Assuming that the repository in question does not have an integrated schema or ontology, data publishers are free to choose arbitrary attribute names, leading to degenerated schemata and thus increasing integration effort at reuse time.

In this section, we investigate how this problem can be approached within the framework of Publish-time Data Integration, by introducing a dataset-local PTDI operator for generating *attribute name recommendations*. When a user publishes a new dataset, this operator augments its schema with alternative attribute names, using statistics it maintains about the attributes and corresponding instances on the platform. These alternative attribute names are generated so that the datatset's schema fits better into the vocabulary already in use on the platform.

To illustrate this process consider Figure 4.7, which depicts a simple exemplary corpus $\mathcal{C}$ consisting of four datasets ds_1 to ds_4 in two different domains. Furthermore, consider the new dataset ds_+ that is to be added to the corpus. The operator should generate the output $\{\text{c\_name} \longmapsto (\text{Country, STATE})\}$, as the latter two attribute names are used in the existing corpus for country names. Since "Country" appears two times (first column in ds_1 and third column in ds_3) and "STATE" only one time (first column in ds_2), "Country" is ranked before "STATE". For the second attribute name "c_lang" in ds_+ there is no recommendation, because no similar attribute exists in the corpus.

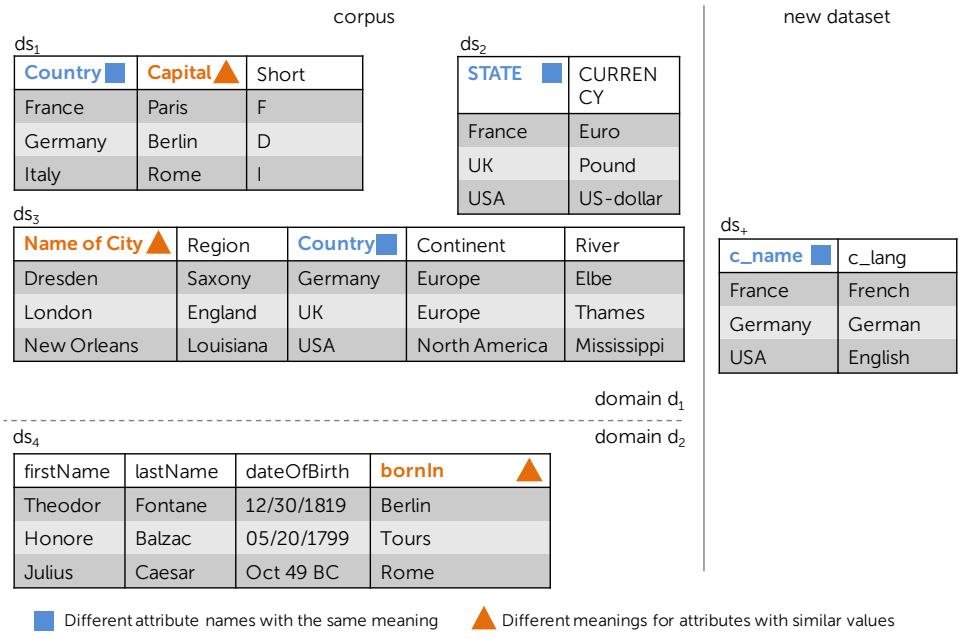


Figure 4.7: Example corpus $\mathcal{C}$ (4 datasets in 2 domains)

Note however, that fitting names for an attribute are not only dependent on its value set, but also on the *domain* it stems from. This should be illustrated by an additional dataset ds_4 belonging to a different domain (see Figure 4.7). Similar to ds_1 it contains an attribute with city names, but the meaning of the attributes differ (*bornIn* versus *Capital* and *Name of City*). Therefore, in order to give correct name recommendation, the operator also takes the existing domains on the platform and the domain of the new dataset into account. Specifically, it maps new datasets into one of the existing domains on the platform, before generating attribute recommendations based on statistics for this distinct domain. Finally, the operator should be capable of being run at publish-time, i.e., with minimal latency as required in Section 4.2.2. The algorithms used in the operator should therefore be fast enough to allow instant publisher feedback. The rest of this section is organized as follows: we will propose four different algorithms for generating attribute name recommendations based on corpus statistics and discuss our domain classification method in Section 4.3.1. In Section 4.3.4 we will report on our crowdsourcing-based evaluation, in which we measured quantity and quality of the generated attribute name suggestions using real-world corpora from `opendata.socrata.com` and Wikipedia.

Parts of the material presented in this section have already been published in (Eberius et al., 2013a).

4.3.1 System Overview

The attribute name recommendation system's architecture corresponds to the general PTDI architecture shown in Figure 4.6. The system works in two phases: an *offline* phase, consisting of two PTDI preprocessors, and an *online* phase, realized by one PTDI operator. In the online phase, the operator is triggered by a user adding a new dataset ds_+, and produces recommendations to improve the reusability of the dataset in the context of the corpus of already published datasets. In the offline phase, the datasets in the corpus are clustered into *domains* by one preprocessor, which is necessary to deal with the different meanings of terms in different contexts, as explained in Section 4.3. Furthermore, the clustering into domains is leveraged to reduce the effort and time needed to compute recommendations in the online phase, by computing per-domain statistics. Collecting domain-specific statistics allows us to map incoming dataset into one particular domain which is much smaller in size compared to the whole corpus. These processes are described in Section 4.3.2. The domain statistics themselves are gathered by a the second preprocessor, and are then utilized in the online phase to quickly generate attribute name recommendations for new datasets being added to the corpus.

In the online phase, the attribute name recommendations for the incoming dataset are created. This phase consists of two steps. First, the already mentioned mapping into a specific domain and second, the comparison between values associated with the new attribute and the domain statistics. To illustrate this process, in Section 4.3.3, we will discuss four specific algorithms with progressively higher sophistication, from a naïve approach to our final design. These approaches differ in the types of statistics they maintain, and the way the comparison between them and the new dataset is performed, but all follow the abstract process given in Figure 4.6.

4.3.2 Domains and Domain-Classification

As mentioned, automatically clustering datasets in the data curation system into *domains* is used to improve accuracy as well as latency of the recommendation generation. We refer to a *domain* as a set of datasets that have a common topic and thus use a common vocabulary. For the clustering of datasets into domains we follow (Mahmoud and Aboulnaga, 2010). Their approach clusters datasets based on attribute names by creating feature vectors for each dataset that represent the vocabulary used in its attribute names and then performing bottom-up clustering using these vectors. The output of the algorithm is a set of triples, containing a dataset, a domain, and a probability of the given dataset belonging to the domain. In our context, one dataset may belong to more than one domain, so domains may overlap.

To classify an incoming dataset ds_+ the same principles are applied. For ds_+ a feature vector must be created that represents the incoming dataset's vocabulary. Then, the most similar schema s_{sim} in the corpus is calculated by comparing the new feature vector to all existing vectors using a similarity function based on the Jaccard coefficient. If a schema with non-zero similarity is found, the algorithm returns the domain d_+ to which the dataset with the most similar schema s_{sim} belongs with the highest probability.

Algorithm 6 The Naïve-$\mathcal{C}$ approach to attribute name recommendation

```
1: function NaïveC(vs+)
2:     names ← [ ]
3:     for all (ai, vsi) ∈ C do
4:         if vsSim(vs+, vsi) ≥ thvsSim then
5:             names ← names : ai
6:     return freqSort(names)
```

4.3.3 Name Recommendation Algorithms

Given the domain d_+ for a new dataset ds_+ we now describe how we generate attribute name recommendations. To this end, we developed four algorithms that differ in the set of datasets they consider, and in the offline-generated statistics they use. Their input and output is defined in the same way. As input they assume a set of values vs_+, i.e., the values of an attribute k of the new dataset ds_+, which has been mapped into a domain d_+. They return a list of possible attribute name recommendations based on attributes existing on the platform. In the following, $\mathcal{C}$ denotes this corpus of existing datasets. Simplifying, we model $\mathcal{C}$ as a flattened set of pairs (a_i, vs_i) of attributes with associated sets of values.

Naïve-$\mathcal{C}$ and Naïve-d_+

The first approach Naïve-$\mathcal{C}$, which we use as a baseline for the other algorithms, does not use the offline phase at all, and accordingly does not create domain statistics, or even use the domain clusterings. It searches the complete corpus for value sets that are similar to vs_+, and returns all attributes names belonging to these value sets as candidates. The similarity $vsSim(vs_x, vs_y)$ between two value sets vs_x and vs_y is calculated by counting the number of values v_x from vs_x for which a similar value v_y can be found in vs_y and the other way around. The sum of both is then normalized with the sum of the cardinalities of the two value sets which leads to similarity score in the interval $[0, 1]$.

$$vsSim(vs_x, vs_y) = \frac{|\{v_x \in vs_x | \exists v_y \in vs_y : v_x \approx v_y\}| + |\{v_y \in vs_y | \exists v_x \in vs_x : v_y \approx v_x\}|}{|vs_x| + |vs_y|}$$

To calculate this value set similarity, the measure for individual value similarity $\approx$ has to be defined. Without loss of generality, we only study textual values and use a combination of well known string distance measures, specifically the average between the Longest Common Substring-, Levenshtein- and nGram-distance. The two values v_x and v_y are deemed similar according to $\approx$ if the value of this average is greater than a fixed threshold (see Experiments in Section 4.3.4). Based on this similarity measure the algorithm Naïve-$\mathcal{C}$ is constructed as shown in Algorithm 6. In this approach, the input value set is globally compared to all value sets in the corpus, saving all the attribute names of the other value sets with sufficient similarity. The resulting list of attribute names is ordered by their frequency in the corpus in the function $freqSort$. This naïve baseline does not not take domains into account, potentially reducing its accuracy, but also leading to poor performance, as each new value set vs_+ is compared to each existing one $vs_i \in \mathcal{C}$.

Therefore, as a first improvement to the baseline, instead of considering the whole corpus $\mathcal{C}$, the Naïve-d_+ algorithm only considers the value sets of the domain d_+ the incoming dataset was mapped into it. Of course this approach requires a correct classification of the datasets as described in Section 4.3.2. Limiting the search for candidate attribute names to the domain has the obvious advantage that fewer value sets have to be compared, which increases performance. On the other hand, it can be argued that correct recommendations might be lost when considering only a subset of the whole corpus. However, as illustrated in Figure 4.7, similar value sets do not necessarily indicate similar meaning if they originate from different domains. For this reason we expect that Naïve-d_+ will not only outperform Naïve-$\mathcal{C}$ in run time, but also in quality of the generated recommendations.

The common disadvantage of both algorithms presented so far is that they perform all of their work *online* which contrasts with our requirement that Publish-time Data Integration calls for fast response times, since data publishers are unlikely to cooperate if they suffer through long waits. Therefore, we developed the following two algorithms which use statistics created offline to improve the response time of the recommendation generation.

Clustering-d_+

The basic idea of the Clustering-d_+ algorithm is to add another clustering step on the domain level. In this preprocessing step, similar value sets within each domain are clustered, and a list of possible attribute names for each of these clusters is stored. Using these clusters, the online step can be reduced to comparing the incoming value set to one representative value set of each of these domain level clusters leading to substantial performance gains. For example, many datasets in a demographic statistics domain will have a column containing nation names, as illustrated in Figure 4.7. In the following, we will describe the offline and online phase of Clustering-d_+ in more detail.

Offline Preprocessing The first function shown in Algorithm 7 describes the preprocessing required for every domain d in the corpus $\mathcal{C}$. The value set clustering used on line 2 is performed as bottom-up agglomerative clustering(Kaufman and Rousseeuw, 1990), where initially every value set $vs_i \in \mathcal{VS}(d)$ in a domain is a cluster of its own. Then the two most similar clusters are successively merged until there is no pair of clusters with a similarity above a fixed threshold. The similarity between two value sets is calculated using the function $vsSim$ from Section 4.3.3.

For each of the clusters in the resulting set C a representative value set rvs is created using the algorithm sketched in Algorithm 8. This representative value set is used in the online phase to reduce the number of comparison that have to be performed with the incoming dataset's values. A representative value set is a subset of the union of values of all value sets in the cluster. The size of this subset is decided using a dynamic threshold, which depends on the most frequent value in the cluster. For each of the clusters in the resulting set C a representative value set rvs is created using the algorithm sketched in Algorithm 8. All the attribute names used for the value sets that were assigned to the same cluster are collected, ordered by frequency, and are saved as a list of candidate attribute names. These representative value sets plus their respective list of attribute names constitute the domain statistics that are used by

Algorithm 7 Offline and Online phases of Clustering-d_+

function ClusteringD-Offline(d)
  $C \leftarrow$ clusterValueSets($\mathcal{VS}(d)$)
  $DI_d \leftarrow \emptyset$ ▷ Output Domain Statistics
  for all $c \in C$ **do** ▷ For each value set cluster
    $rvs \leftarrow$ representativeVS(c) ▷ Most common values in cluster
    $names \leftarrow [\,]$
    for all $(a_i, vs_i) \in c$ **do**
      $names \leftarrow names : a_i$
    $e \leftarrow$ freqSort($names$)
    $DI_d \leftarrow DI_d \cup \{(rvs, e)\}$ ▷ Store repr. value sets and recommendations
  return DI_d

function ClusteringD-Online(vs_+, d_+)
  $(rvs_{max}, r_{max}) \leftarrow \underset{(rvs,r)\in DI_{d_+}}{\arg\max} \; rvsSim(vs_+, rvs)$ ▷ Most similar repr. value set
  if $rvsSim(vs_+, rvs_{max}) > 0$ **then**
    return r_{max}
  else
    return $[\,]$

Clustering-d_+.

Online Operation By leveraging these statistics, the online operation of the Clustering-d_+ algorithm, shown in the second function of Algorithm 7, can be made simpler and more efficient. Each incoming value set vs_+ now has to be assigned to the most similar representative value sets rvs, which is already precomputed. Note that the similarity measure for value sets needs to be modified for this algorithm.

$$rvsSim(vs, rvs) = \frac{|\{rv \in rvs | \exists v \in vs : rv \approx v\}|}{|rvs|}$$

In contrast to the similarity measure $vsSim$ from Section 4.3.3, we have to use $rvsSim$ which is only dependent on the number of values from rvs that have a partner in vs, but not the other way around. This is due to the construction of rvs which deliberately omits rare values, while the incoming value set vs may contain values of any frequency. Using the modified measure $rvsSim$, the most similar representative value set in the domain is identified, and the prepared list of possible recommendations is returned.

Analysis-d_+

The final algorithm, Analysis-d_+, also creates domain statistics to abstract the content of a domain, but uses an inverted preprocessing logic compared to Clustering-d_+. Instead of clustering value sets and saving attribute names that belong to them, this algorithm groups value

Algorithm 8 Extracting representative value sets

function RepresentativeVS(c)
  $uniqueValues \leftarrow \bigcup_{vs \in c} vs$
  $mostFreq \leftarrow \max_{v \in uniqueValues} |\{vs_i \in c | v \in vs_i\}|$
  $rvs \leftarrow \emptyset$
  for all $v \in uniqueValues$ **do**
    if $|\{vs_i \in c | v \in vs_i\}| \geq \frac{3}{4} \cdot mostFreq$ **then**
      $rvs \leftarrow rvs \cup \{v\}$
  return rvs

Algorithm 9 Offline and Online phases of Analysis-d_+

function AnalysisD-Offline(d)
  $G \leftarrow groupByAttrName(d)$
  $DI_d \leftarrow \emptyset$
  for all $(a_i, g) \in G$ **do**
    $vals \leftarrow \bigcup_{vs \in g} vs$
    $rvs \leftarrow representativeVS(vals)$
    $DI_d \leftarrow DI_d \cup \{(rvs, a, |g|)\}$
  return DI_d

function AnalysisD-Online(vs_+, d_+)
  $names \leftarrow [\,]$
  for all $(rvs, a, z) \in DI_{d_+}$ **do**
    if $rvsSim(vs_+, rvs) \geq th_{rvsSim}$ **then** $names \leftarrow names : \underbrace{a : \cdots : a}_{z-times}$
  return $freqSort(names)$

sets with the same attribute name. This leads to differences both in the form of the domain statistics and the way in which the online phase generates recommendations for the incoming dataset.

Offline Preprocessing The offline phase of the algorithm is actually simpler than Clustering-d_+. Instead of clustering value sets, it simply groups value sets by their attribute names, and then creates a representative value set for each group. In contrast to the domain statistics of Clustering-d_+, which consist of tuples of representative value set with a corresponding list of attribute names, Analysis-d_+ creates triples consisting of a representative value set for each group, the group key, i.e., a single attribute name, and the frequency of this attribute name, i.e., the group's size. Saving this frequency is necessary as the list of recommendations and its ranking is generated in the online phase of the algorithm, as opposed to offline with Clustering-d_+, where lists of recommendations are precalculated.

Online Processing In the online phase of Analysis-d_+, the incoming value set can again be compared to the representative value sets saved in the offline phase, instead of comparing it to all other value sets in the domain. In contrast to Clustering-d_+ however, each of these representative sets is only associated with one attribute name, not with a complete list of recommendations. For this reason, it is not enough to find the most similar triple in the domain statistics. Instead all triples where the similarity between rvs and the incoming vs is above a threshold have to be collected. The gathered triples can then be sorted by their frequency value, so that the attribute names that are most commonly used are ranked higher.

Having introduced four different algorithms that can implement the attribute name recommendation generation operator, we will present an experimental evaluation in the next section.

4.3.4 Experimental Evaluation

We conducted an experimental study to evaluate

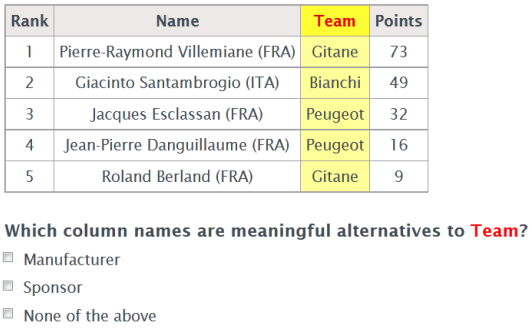
1. the runtime behavior of the domain clustering and the recommendation algorithms,
2. the result quality for the four different approaches, and
3. the impact of the existing corpus size on the number and quality of generated attribute name recommendations.

Experimental Setup

In this section, we will give the details of our experimental setup, such as dataset corpora and input datasets that were used, and discuss our result assessment method.

Test Corpus For our evaluation we conducted various experiments on real-world data. In detail, we used the following two sources to extract corpora of relational tables for the domain clustering:

- **Socrata:** *opendata.socrata.com* is an Open Data platform, which encompassed 16,591 datasets at the time we extracted our test corpus. From the complete list of datasets, we discarded all tables with less than 3 columns or 2 rows, as well as all tables with a size of more than 100MB. We also discarded nested tables, which are technically possible in Socrata's data model.
- **Wikipedia:** For our second corpus, we extracted tables from English Wikipedia pages, using techniques similar to the ones presented in (Cafarella et al., 2008). We extracted a subset of 1.7m relational Wikipedia tables from our Web table corpus DWTC (Eberius et al., 2015a), and again discarded tables that were either too small or too large. From the remaining tables we randomly selected three sets of different sizes. We created one set of 5,000 tables (*Wiki5k*), one of 10,000 tables (*Wiki10k*) and one set of 20,000 tables (*Wiki20k*), with *Wiki5k* $\subset$ *Wiki10k* $\subset$ *Wiki20k*. These sets of different sizes enabled us to analyze the impact of the corpus size on the recommendation quality.



Rank	Name	Team	Points
1	Pierre-Raymond Villemiane (FRA)	Gitane	73
2	Giacinto Santambrogio (ITA)	Bianchi	49
3	Jacques Esclassan (FRA)	Peugeot	32
4	Jean-Pierre Danguillaume (FRA)	Peugeot	16
5	Roland Berland (FRA)	Gitane	9

Figure 4.8: Layout of the CrowdFlower task

Test Datasets In addition to these corpora, we also extracted different classes of test data from both sources as input tables for the algorithms. For each potential test table, we counted the frequency with which the column name of each text attribute appeared in the respective corpus. We then used the highest encountered frequency to normalize the values and divided the resulting frequency range into six classes. The first class f_0 contains all attributes that appear in the test data, but not in the corpus (i.e., the frequency equals 0). Classes $f_1 - f_5$ are equal to the quintiles of the frequency distribution. Finally, we selected the tables so that the overall test set contained at least 100 attributes for each frequency class.

Hardware For all runtime experiments, we used a 64-bit Linux (Ubuntu 11.10) system with 8GB RAM and two 2.4Ghz Intel Xeon CPUs.

Recommendation Quality Assessment We performed several experiments to evaluate the quality of the recommendations produced by the system. In general, an impartial quality assessment requires the evaluation of the results by actual users of the system. However, such an evaluation can be very extensive and time consuming when performed on a large scale. Recently however, there have been successful advances in using crowdsourcing platforms to generate relevance assessments, e.g., (Blanco et al., 2011; Carvalho et al., 2011) and (Alonso et al., 2008). Therefore, we also utilized the functionality and services provided by crowd-sourcing platforms such as CrowdFlower[4] or Amazon's Mechanical Turk[5] for the quality assessment. Simple tasks, such as evaluating the correctness of a result, can be posted on these platforms to be processed by registered users worldwide for a small compensation. The distribution of the tasks, quality control and the aggregation of the results are handled by the crowd-sourcing platform, which makes it a convenient tool for large scale user evaluation.

Crowdsourcing Setup To evaluate the quality of the generated recommendations, we generated crowd-sourcing tasks, usually called HITs (human intelligence tasks), which were submitted to the crowdsourcing platform *CrowdFlower*. The layout of the tasks is depicted in Figure 4.8. For each attribute name, we displayed the original table and highlighted the column in question. For large tables, we limited the number of displayed rows to 20. Below the table we

[4] www.crowdflower.com
[5] www.mturk.com

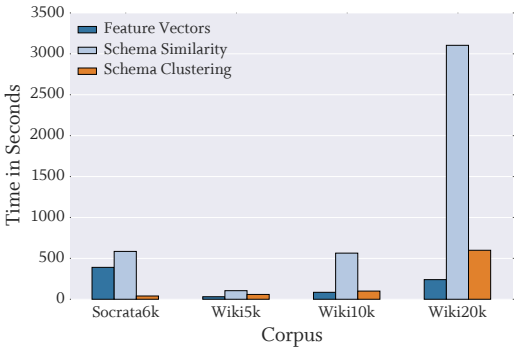


Figure 4.9: Runtime for individual processing steps

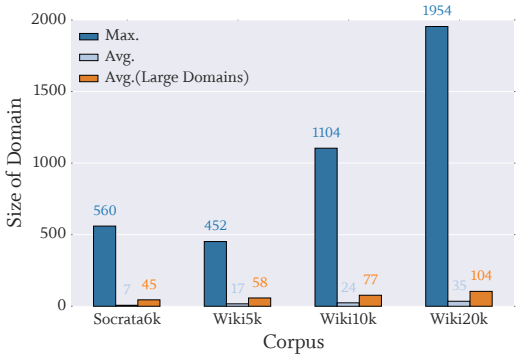


Figure 4.10: Max and average number domain sizes (domains with a size > 10 count as large domains)

displayed up to five alternative names recommended by our algorithms with checkboxes. We asked the crowd-workers to check all names which they thought were meaningful alternatives to the original name, or to check *None of the above* (see Figure 4.8). Each individual task was given to three different crowd-workers and the answers were accepted if at least two of them agreed on them.

Domain Clustering

At first, we performed the domain clustering as described in Section 4.3.2 for all test corpora. We chose the same parameter settings as suggested by (Mahmoud and Aboulnaga, 2010).

Runtime Requirements Figure 4.9 shows the runtime required for all the individual processing steps performed during the domain clustering. The results show that computation of the similarities between the document schemas is the most expensive step dominating the other processing stages. This is due to the quadratic complexity of the pairwise schema matching performed in this step. Since the domain clustering is performed once offline before performing the recommendation algorithms, a longer runtime for this step is not an immediate issue. However, for very large corpora the domain clustering has to be revised in terms of performance, e.g., by using suitable blocking strategies (McCallum et al., 2000; Jaro, 1989).
Concerning the extraction of the additional domain information required for Clustering-d_+ and Analysis-d_+, we found that this step requires significantly less time in Analysis-d_+ then in Clustering-d_+, where pairwise comparison of value sets are necessary.

Domain Size For each corpus the domain clustering algorithm generates a large number of domains of different sizes. The maximum and average domain sizes are presented in Figure 4.10. As expected, the average domain size increases with the corpus size, e.g., from 17 for the Wiki5k corpus to 35 for the Wiki20k corpus. The reason is that the number of domains is naturally bounded while the number of documents continuously increases with the corpus size. We further see that there are a few very large domains containing up to 1,954 documents

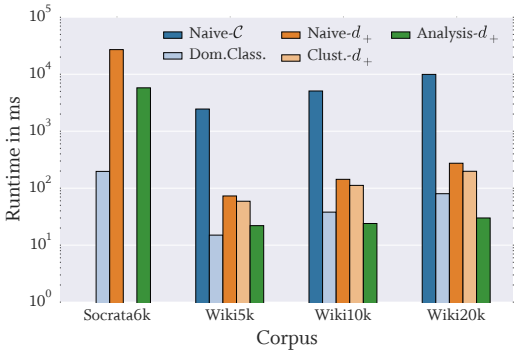


Figure 4.11: Average runtime for domain classification and attribute name recommendation (log scale)

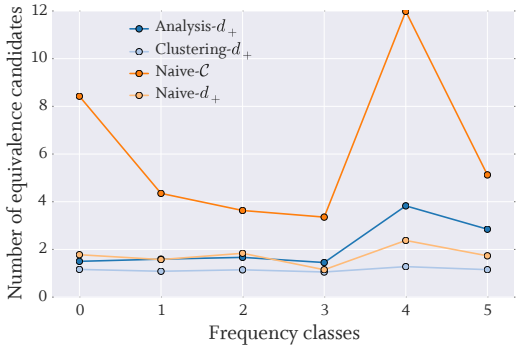


Figure 4.12: Avg. number of name candidates found for input attributes using corpus Wiki20k, by frequency class

for the Wiki20k corpus. However, there is also a long tail of domains which contain only a single document.

Name Recommendation Algorithms

Based on the domain clusters and domain information generated during the offline phase, we evaluated the different name recommendation approaches described in Section 4.3.1. For each test set (*Socrata* and *Wikipedia*) we tested all four algorithms. For *Wikipedia*, we also varied the corpus size as described earlier. Figure 4.11 shows the average runtime per table. The time required for domain classification must be added to the runtime of all $* - d_+$ algorithms. While the *Wikipedia* set shows acceptable runtime behaviour, the *Socrata* set requires considerably more time for the recommendations. This is caused by the fact that *Socrata* does contain some exceptionally large tables in the corpus. For two of our approaches, we even canceled the experiments due to these runtime issues. Overall, Analysis-d_+ is the fastest of the four algorithms, achieving a response time less than 100ms in most cases.
Figure 4.12 shows how many different name recommendations the respective algorithms were able to propose by frequency class, i.e., by the frequency of the original attribute in the corpus (see *Experimental Setup*). As expected Naïve-$\mathcal{C}$ generates much more diverse recommendations as it considers the whole corpus for each input set, while the other three algorithms just consider the one domain the input dataset is mapped into, which will contain a much smaller set of potential attribute names. However, as shown in Figure 4.9, this comes at the cost of a prohibitively high runtime. Furthermore, as we will see in the next section, precision also suffers from generating more recommendations.

Recommendation Quality

In addition to the runtime experiments, we also analyzed the quality of the recommended attribute names. We leveraged the crowd-sourcing platform *CrowdFlower* in order to have users evaluate the correctness of the recommendations. Before we ran the tasks on the crowdsourcing platform, we took the top 5 recommendations for each attribute, for which the four

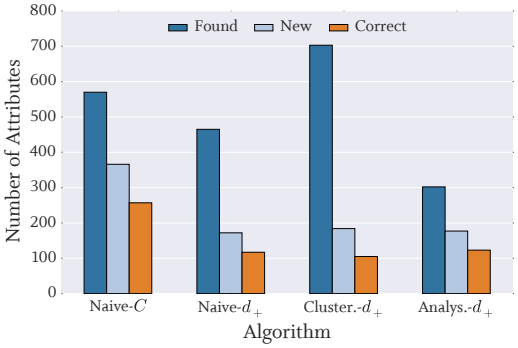


Figure 4.13: Overall, new and correct number of recommended attributes

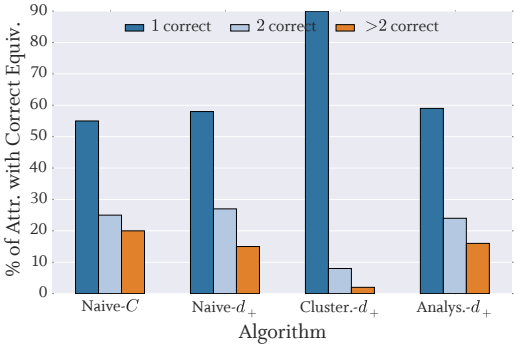


Figure 4.14: Percentage of attributes with exactly 1, exactly 2 or $>$ 2 correct recommendations in the top 5

algorithms generated results. If less than 5 attribute names were returned by an algorithm, we only used these. Since we were looking for meaningful alternatives to the original attribute name, we removed mentions of the original name from the top-5 set. Figure 4.13 shows the fraction of the recommendations which included new attribute names, i.e., not only the original attribute name. For this fraction of recommendations, we generated crowd-sourcing tasks, as described above in the experimental setup. First of all, we analyzed the number of attributes for which at least one attribute name was marked as correct by the crowd. Figure 4.13 shows the results for all four approaches using *Wiki20k* as the corpus. The dataseries *Found* describes the number of attributes for which recommendations could be found. *New* denotes the number of recommendations that were different from the original attribute name, whereas *Correct* counts the recommendations which have been verified positively by the crowd-workers. In most cases, correct recommendations were found for about 70% (e.g. 257 of 366 attribute names for Naïve-$\mathcal{C}$) of the attributes evaluated through the crowd. Only Clustering-d_+ showed a slightly lower precision. Still, the majority of recommendations contained at least one meaningful alternative.

Depending on the datasets available in the corpus, there can be more than one meaningful alternative in the top-5 recommendations. Therefore, we analyzed the crowd-sourcing results to see how many of the alternative names were found to be correct. Figure 4.14 shows the percentage of attribute names for which exactly one, exactly two or more than two correct recommendations were found in the top-5 recommendations. Again, Naïve-$\mathcal{C}$, Naïve-d_+ and Analysis$-d_+$ show very similar characteristics, with exactly one correct recommendation in 55-59% of the cases. Clustering-d_+, on the other hand, has a significantly smaller fraction of attributes with more than one correct recommendation. This result is clearly caused by the fact that this approach generates significantly less recommendations. Naïve-$\mathcal{C}$ produces the largest amount of correct name recommendations, which is expected, since it uses the most comprehensive matching approach.

To evaluate the quality of the ranking, we counted the number of tasks with at least one correct name recommendation, where a correct name was ranked in the top position. Figure 4.15 depicts the results, again with *Wiki20k* as the corpus. The results show that in most cases the

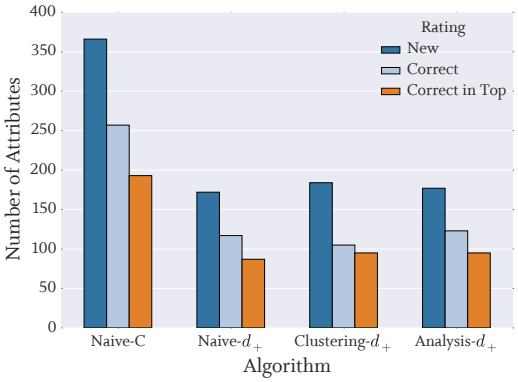


Figure 4.15: Number of attributes for which the top recommendation was correct

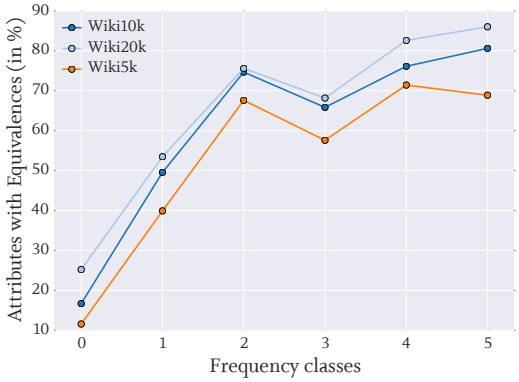


Figure 4.16: Fraction of attributes for which Clustering-d_+ was able to generate recommendations, by frequency class

correct name was ranked highest, which indicates a high ranking quality.

Corpus Size

Finally, we also analyzed the impact of the corpus size on the recommendation quality. It is expected that a larger corpus is more likely to contain tables that are similar or related to the test table. Therefore, a larger corpus should enable more name recommendations. To test this hypothesis, we used the three *Wikipedia* corpora, which are subsets of each other. We formed domain clusters for each corpus and used these to generate recommendations for the same set of test tables. Figure 4.16 presents the results only for Clustering-d_+, but the remaining algorithms show a similar behavior. We can see that with an increasing corpus size for each frequency class more attribute names are generated. Using the results from the crowd-based evaluation, we can also show that a larger corpus does not only result on more recommendations, but in more correct ones. From these results we can assume that Publish-time Data Integration as a concept will show even better results on larger, real-world data repositories. As a statistics-driven approach, the attribute name recommendations will show better quality and quantity the more data is available in the corpus.

4.3.5 Conclusions

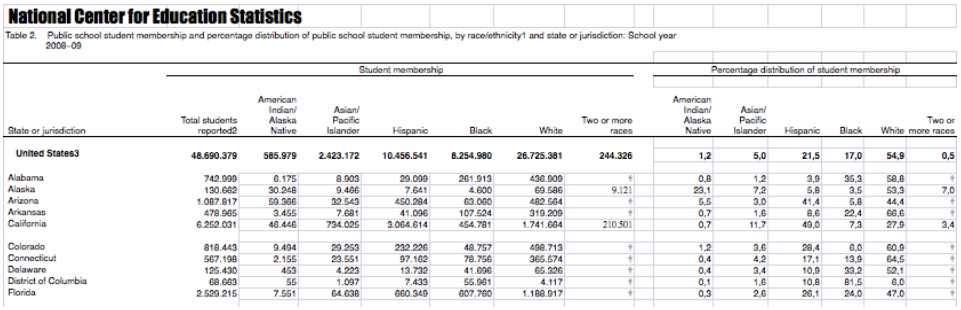
Collaborative data management platforms such as Open Data Repositories are becoming increasingly prevalent. As thousands of users continuously publish data to these platforms, the schema vocabulary steadily increases leading to highly heterogeneous corpora. In this section, we therefore proposed the a PTDI operator for generating attribute name recommendations, which aligns the attribute names of new dataset with the corpus to which the dataset should be added. In order to solve this problem we developed the four algorithms *Naïve-$\mathcal{C}$*, *Naïve-d_+*, *Clustering-d_+* and *Analysis-d_+* that leverage a varying degree of offline-generated statistics, resulting in both different recommendation quality and runtime behavior.

To ensure a sound quality assessment of these algorithms we utilized the human power provided by the CrowdFlower crowd-sourcing platform. We showed that Naïve-C produces the largest amount of correct name recommendations but at the same time is also orders of magnitudes slower than the other three approaches. However, Publish-time Data Integration algorithms must be fast enough to enable instant user feedback during the publishing process. Therefore, *Clustering-d_+* and *Analysis-d_+* provide a good mix between result quality and runtime. Using these algorithms we are able to provide attribute name recommendations during *publish-time* and thus to tame the growth of heterogeneity in collaborative data management platforms.
So far, we have discussed only PTDI operators that apply to structured data, i.e., dataset where attribute names, values associated with attributes and even meta data properties were explicit. The operator presented in the last section requires the datasets to be in relational form. However, datasets on data curation systems are not necessarily published in this way. Instead, the data is often published in formats optimized for human consumption, such as spreadsheets, HTML and PDF, as we have shown in Section 4.1. This makes transformation into a true relational form necessary, as it is a precondition not only for the other PTDI operators presented here, but for most kinds of data analysis and data integration. Therefore, in the next section, we will introduce another PTDI operator, called the *DeExcelerator*, which, on publication of a partially structured document to a data curation system, automatically proposes transformations into a relational form with explicit metadata to increase the dataset's reusability.

4.4 EXTRACTING DATA FROM PARTIALLY STRUCTURED DOCUMENTS

Datasets in data curation systems can be published in various formats. Whether it is a corporate data lake, or public Open Data platform, these repositories are designed to accept all kinds of data, without strict rules regarding the technical format. This leads, as shown for the case of Open Data repositories in Section 4.1, to a zoo of formats being represented on the platform. Some may be readily machine processable, and thus easily reusable, such as CSV files containing tabular data. Formats such as XML or JSON files can encode more complex, hierarchical structures. In addition to these structured formats, others more targeted at human consumption, such Excel spreadsheets or PDF and HTML documents, are preferred especially by non-technical publishers. For example, (Martin et al., 2011) showed that for European Open Data stakeholders, CSV/XLS, HTML and PDF are still the most common data formats used for publishing data.

In these formats, data is intermingled with formatting, layout and textual metadata, i.e., it is contained in *partially structured documents*. While this mix is what gives these formats their expressiveness and power, it also lowers reusability, as the data has to extracted from the document before it can be reused. To reuse the wealth of structured data contained in these datasets in a different context or to find relations between datasets, a human analyst will often have to transform these datasets manually. Furthermore, even though formats like spreadsheets are tabular in nature, many users, because of lack of easily usable alternatives, embed non-tabular data in these documents (Benson and Karger, 2014).



National Center for Education Statistics

Table 2. Public school student membership and percentage distribution of public school student membership, by race/ethnicity1 and state or jurisdiction: School year 2008–09

	Student membership							Percentage distribution of student membership					
State or jurisdiction	Total students reported2	American Indian/ Alaska Native	Asian/ Pacific Islander	Hispanic	Black	White	Two or more races	American Indian/ Alaska Native	Asian/ Pacific Islander	Hispanic	Black	White	Two or more races
United States3	**48.690.379**	**585.979**	**2.423.172**	**10.456.541**	**8.254.980**	**26.725.381**	**244.326**	**1,2**	**5,0**	**21,5**	**17,0**	**54,9**	**0,5**
Alabama	742.999	6.175	8.903	29.099	261.913	436.909	†	0,8	1,2	3,9	35,3	58,8	†
Alaska	130.662	30.248	9.466	7.641	4.600	69.586	9.121	23,1	7,2	5,8	3,5	53,3	7,0
Arizona	1.087.817	59.366	32.543	450.284	63.060	482.564	†	5,5	3,0	41,4	5,8	44,4	†
Arkansas	478.965	3.455	7.681	41.096	107.524	319.209	†	0,7	1,6	8,6	22,4	66,6	†
California	6.252.031	46.446	734.025	3.064.614	454.781	1.741.664	210.501	0,7	11,7	49,0	7,3	27,9	3,4
Colorado	818.443	9.494	29.253	232.226	48.757	498.713	†	1,2	3,6	28,4	6,0	60,9	†
Connecticut	567.198	2.155	23.551	97.162	78.756	365.574	†	0,4	4,2	17,1	13,9	64,5	†
Delaware	125.430	453	4.223	13.732	41.696	65.326	†	0,4	3,4	10,9	33,2	52,1	†
District of Columbia	68.683	55	1.097	7.433	55.961	4.117	†	0,1	1,6	10,8	81,5	6,0	†
Florida	2.529.215	7.551	64.638	660.349	607.760	1.188.917	†	0,3	2,6	26,1	24,0	47,0	†

Figure 4.17: Example for a partially structured Open Data document

Consider Figure 4.17 as an example. It shows a real-world Open Data file, part of the `data.gov` evaluation corpus used in this section. Even though it encodes strictly relational data, that could be easily loaded into typical data analysis tools, it is not easily reusable in its present form. For one thing, the author chose to embed metadata such as title, subtitle, and even time of validity into to the cell structure of the document. Furthermore, the document actually contains two tables, one with absolute values on the left and one with percentages on the right. The separation between those is implicitly encoded using empty cells. Furthermore, some cells in numeric columns contain symbols instead of numbers, while others contain derived data instead of raw data. In short, manual cleaning will be necessary before the relation contained in the document can be used in further analysis. However, one of the central aims of data curation systems is enabling reuse by all concerned parties with minimal effort. If a document is published in this form, the required cleaning effort may even discourage reuse. Even worse, one user may clean the dataset for his or her own purposes, i.e., transform it into CSV with an accompanying metadata file, without republishing the derived dataset. This will lead to duplicate efforts if another user again works with the original spreadsheet in the future.

Therefore, in this section, we will explore *relation extraction* from *partially structured* documents as a PTDI operator. The presented operator, callsed the *DeExcelerator*, will automatically recommend transformation steps for extracting data and metadata from such documents and making it available explicitly. The aim of applying all transformation steps is to turn the input document into a first normal form relation, ideally without any user interaction except verification. We call this process *spreadsheet normalization*, although the process applies not only to spreadsheets, but to similar data formats, such as HTML-embedded tables, as well. Although there is some support for performing this task semi-automatically, which we study in Section 4.5, there exists no fully automatic system for extraction of relational data from such documents.

As a first step, we study `data.gov` as an example source for partially structured documents. We then propose a classification of normalization problems typical for such documents, i.e., common practices in such datasets that deviate from standard relational form, preventing efficient reuse of the data. Next, we present the DeExcelerator, a PTDI operator for extracting relations from partially structured documents such as spreadsheets and HTML tables. We evaluate the system on real-world open data published on `data.gov` in the form of Excel files, and check the correctness of the extraction results with a user study in Section 4.4.3.

Note, that parts of the material presented in this section have already been published in (Eberius et al., 2013b).

4.4.1 Spreadsheet Normalization

By manually studying a corpus of real-world Open Data Excel spreadsheets on the platform `data.gov` (see Section 4.4.3), we identified a set of typical denormalizations that typically appear in spreadsheets. Our goal was to transform the corpus of documents from this platform into a set of tables that can be handled by an off-the-shelf relational database management system. While we originally studied spreadsheets to identify these characteristics, other tables created for human consumption, e.g. tables embedded in HTML documents on the Web, show most of the identified characteristics, as well. Note, that most of these denormalizations are not introduced because of lack of database know-how, but because spreadsheets are designed to be comprehended visually by a human, not to be processed by a DBMS. We found the following list of challenges for spreadsheet normalization.

Multiple Tables per Sheet While spreadsheet software usually offers the possibility to save multiple different sheets in one document with some form of logical separation, users often just copy multiple tables into one physical sheet for convenience.

Tabelle 1			
Attribut A	Attribut B	Attribut C	Attribut D
A_1	B_1	C_1	D_1
Tabelle 2			
Attribut E	Attribut F		
E_1	F_1		

(a) Two logical tables

Tabelle 1a			
Attribut A	Attribut B	Attribut C	Attribut D
A_1	B_1	C_1	D_1
Tabelle 1b			
Attribut A	Attribut B	Attribut C	Attribut D
A_2	B_2	C_2	D_2

(b) One logical table

Figure 4.18: Detection of physical and logical tables

This is usually done to group related tables visually for easier comprehension of their connection. Concerning data reuse however, it implies that the document will contain multiple headers, multiple blocks of values, and multiple groups of metadata cells. Figure 4.18a shows an example of two tables with different schemata grouped in one physical spreadsheet. They are separated through empty rows and rows with merged cells. In these cases, two tables should be detected, and all other transformations should be applied to both of them separately.

However, this can not be generalized for real-world data. Consider Figure 4.18b, where the two tables are separated in the same way, but actually have the same schema and form only one logical table. This effect often occurs as a visual artifact, when authors copy the header after a fixed amount of rows to avoid excessive scrolling. It may, however, also occur to mark partitions of a table, e.g, one table for each year of data. In both cases, only one table should be detected, and further transformations should be applied to a merged table.

Inline Metadata Metadata is essential to the interpretation of the spreadsheet. For example, information regarding provenance or time of validity is necessary for understanding and there-

fore correct reuse of a dataset. However, such information is often just copied into the cells of the spreadsheet. See Figure 4.19 (or 4.17) for an example.

CARE FOOD PROGRAM				
State	Meals Served			
or District	Total	Homes	Adult	Centers
Alabama	1062	542	509	11
Alaska	866	414	448	4
District				
of Columbia	792	357	433	2
FT 2011 data are preliminary; all data are subject to revision.				
RP = Reduced Price				

Figure 4.19: Metadata Detection

Here the textual metadata appears as a title above the table, and a footer with explanations below the data part of the table. Some of this data might be highlighted visually, e.g. a table title might be underlined, but other metadata might appear in the next cell, without any clear separation. In the step of metadata extraction, the problem is not to semantically understand the metadata, but only to separate metadata from the table header and the data, as a prerequisite for further processing. This step is challenging, as no information should be lost, i.e., the metadata should be detected completely, but also no parts of the table header or the data should be cut of and moved into the metadata.

Layout Elements Many authors add empty cells to improve legibility, even if this distorts the column/row structure of the table. Data columns might be separated by empty columns to create visual padding, or rows might be inserted to simulate line breaks in cells with long text content.

CARE FOOD PROGRAM				
State	Meals Served			
or District	Total	Homes	Adult	Centers
Alabama	1062	542	509	11
Alaska	866	414	448	4
District				
of Columbia	792	357	433	2
FT 2011 data are preliminary; all data are subject to revision.				
RP = Reduced Price				

Figure 4.20: Layout Element Removal

The challenge in this transformation step is detect and remove layout elements without loosing information. The result of the transformation should be that the processed table forms a mostly contiguous block of cells. For example, empty or mostly empty columns can be removed in most cases without loss of data. However, in the example in Figure 4.20, removing the almost empty row would eliminate the word “District”, which is actually part of the value in the cell below.

Implicit Schema In this step, the header of the table, i.e., its schema, has to be detected. While attribute names will often be highlighted visually using bold face or colors, spreadsheets offer no machine readable distinction between data and schema information. Sometimes the attribute name might be the first cell of a data column, as in correctly formated CSV files, but this simple heuristic will fail most of the time with real-world spreadsheets.

State	Meals Served			
or District	Total	Homes	Adult	Center
				s
Alabama	1062	542	509	11
Alaska	866	414	448	4
District				
of Columbia	792	357	433	2

(a) Header Separation

State	Meals Served			
or District	Total	Homes	Adult	Center
				s
Alabama	1062	542	509	11
Alaska	866	414	448	4
District				
of Columbia	792	357	433	2

(b) Header Analysis

Figure 4.21: Header Recognition

For an example, see Figure 4.21. Assuming the metadata and layout rows have been correctly extracted in a previous transformation, the attributes would indeed start at the first line. However, just taking the first cell of each column as its header would not be correct. In this example, the full attributes are scattered both horizontally and vertically over adjacent cells. Firstly, the attributes span several rows, i.e., finding the separation line between data and header is a prerequisite for recognizing the actual attributes, as shown in Figure 4.21a. Even if the bottom bound of the header is recognized, the question of how to merge cells to atomic attribute names has to be answered, as shown in Figure 4.21b. We observed two main types of scattered attribute names. The first type are simple line breaks, such as in the first column of Figure 4.21b, which can be solved by merging the cells. The other type are hierarchical attributes, that are typically represented with multi-line merged cells in spreadsheets, such as the "Meals Served" hierarchy in the example, represented by a merged cell across four columns, that splits into four specific child attributes in the cells below. Recognizing such cells as hierarchical attributes is a main challenge in spreadsheet header recognition.

Data Type Recognition Spreadsheets often contain data type information on a cell-by-cell basis. If the previous challenges have been solved successfully so that clean, self-contained columns have been identified, one might expect that this challenge becomes trivial. Still, many spreadsheets will contain inline textual annotations invalidating type information the spreadsheet tool might be able to provide, while documents from other sources, e.g. tables extracted from HTML documents have no type information at all. So the challenge in this transformation is both to recognize data types on the level of initial cells if no such information is present, and on the level of columns, which may require some fuzziness to tolerated inline annotations.

Ambiguous Null Value There is no standard way of marking null values in spreadsheets. Some authors will leave cells empty, other will use more or less well-known markings such as "N/A". As shown in the initial example (Figure 4.17), cells may even contain graphical markings and symbols. This is further complicated by the fact that empty cells do not necessarily signal null values, but may also mark implied values, which are described in the next section.

Implicit Values Since spreadsheets often contain multidimensional data in one denormalized table, many authors skip repeated values that occur because of denormalization for visual clarity.

Attribut A	Attribut B	Attribut C	Attribut D
A1	B1	C1	D1
		C2	D2
	B2	C3	
		C4	D3
	B3	C5	D4

(a) Implied Values

Attribut A	Attribut B	Attribut C	Attribut D
A1	B1	C1	D1
		C2	D2
	B2	C3	
		C4	D3
	B3	C5	D4

(b) Null Values

Figure 4.22: Detection of Implied and Null Values

An example is shown in Figure 4.22a, where the value of Attribute A is A_1 for all lines, but the value was made explicit only in the first row, and is implied in the rest of the column. The aim of the Value Extrapolation step is to make all implied values explicit. In many cases, simply filling neighboring empty cells with previous values might solve this problem. Still, care has to be taken to distinguish between implied values and null values, which may be signaled by empty cells as well, as shown in Figure 4.22b.

Dependent Rows/Columns Easily adding derived rows and columns such as *Total* cells and summation rows that are automatically updated is a main selling point of spreadsheet software. These additional cells are typically used like pre-defined views on the raw data, which by embedding them within the same document, aid the users understanding or steer him or her toward specific insights in the data.

CARE FOOD PROGRAM				
State	Meals Served			
or District	Total	Homes	Adult	Centers
Alabama	1062	542	509	11
Alaska	866	414	448	4
District				
of Columbia	792	357	433	2
FT 2011 data are preliminary; all data are subject to revision.				
RP = Reduced Price				

Figure 4.23: Derived Values Removal

Figure 4.23 highlights such a summation column in the example table. However, when the data is published including these derived columns it contains redundant information, that may not be necessary for reusers. It may even complicate re-use because of additional effort necessary to understand and clean the dataset.

Even when all these transformations have been applied successfully, the resulting table might still be only in first normal form, i.e., not in optimal form for integration with another database. However, automatic decomposition of the resulting table is out of scope of the operator discussed in this section. This problem is treated, for example, in (Braunschweig et al.,

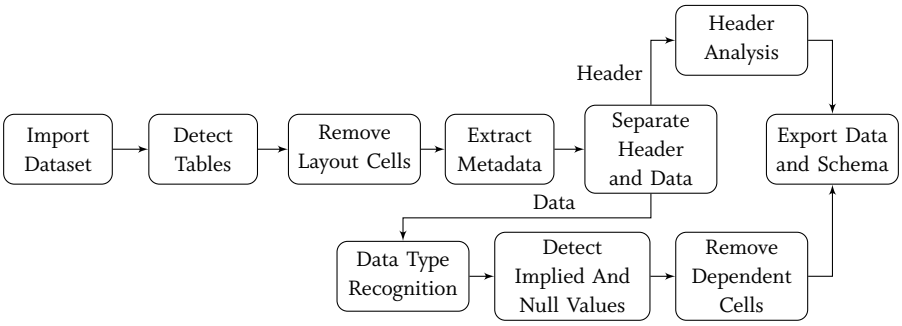


Figure 4.24: The DeExcelerator Process

2015). Having established the specific problems and transformation we aim at, in the next section we will discuss the DeExcelerator, a framework that implements these transformations.

4.4.2 The DeExcelerator

The DeExcelerator PTDI operator is aimed at transforming partially structured documents, i.e., documents that contain some relational data, into a first normal form relation that can be imported into a relational database. It implements a pipeline of abstract transformation phases, each one cleaning or removing one of the artifacts and denormalizations described in Section 4.4.1. The phases and their order roughly correspond to the challenges given there and are visualized in Figure 4.24.

The first phase is an *import* step, in which input files are transformed into a generic representation. As the minimal, common representation of our problem space we chose a two dimensional array of strings, with optional cell-level metadata. It is simple enough to be used with HTML tables as input, which do not have much information attached to them except the structure of row and cell tags, but also allows to capture the metadata available in spreadsheets, such as formatting or explicit data types. In addition, an initially empty global key/value metadata structure is initialized, which is used to store metadata and schema information discovered during transformation steps. All further steps are defined as transformations of this matrix, with optional output into the metadata structure.

After the import step all cells of the matrix are not yet differentiated: the matrix will consist of cells with metadata text, headers or data values at any position. The other phases are defined by the expected state of the matrix after they are applied, but their specific implementations are not prescribed. For instance, after the next phase, *Table Detection*, the matrix is expected to contain exactly one table, while after the *Extract Metadata* step, the matrix is expected not to contain any textual information above and below the table in question. The idea is to offer an extraction framework, in which domain knowledge about partially structured documents can be encoded. For each of these phases, the DeExcelerator allows to define multiple implementations, i.e., multiple heuristics for solving the task that defines each phase.

The DeExcelerator not only specifies this framework, but also contains basic implementations of the abstract extraction operations, which are based on our study of real-world datasets, as well as on table extraction techniques from the literature (see Section 4.5). Note, that due

to the framework character of the DeExcelerator, additional heuristics can be easily added to each phase, which allows to tune the operator and use available domain knowledge specific to the data curation system. Extraction heuristics implemented in the DeExcelerator operate on the string matrix only, but may use the cell-level metadata attached by the import step, e.g., use color information defined on the original spreadsheet cells as evidence for the header recognition. All implemented heuristics will return a *transformed matrix*, as well as a *confidence* value. Each transformation may also attach new metadata to the cells of the matrix, or into the global metadata structure maintained for the process. The confidence values are used by the DeExcelerator to decide on the extraction output in case of conflicting results, for example if two different sets of attribute names are found using two different heuristics in the *Header Analysis* phase.

There are several ways to resolve such conflicts: One is fully *automatic*, by using the confidence scores to decide on which heuristic's result to use in each phase. In the context of a PTDI platform, however, another possibility is submit the results in the form of PTDI recommendations to the author at publish-time, ranked by their confidence values. In this case, two further forms can be distinguished. One is *fine grained*, i.e., submit a PTDI-recommendation for each conflicting result of any phase of the DeExcelerator process. The other would be *coarse grained*, i.e, use an automated decision for resolving the conflicts in each individual phase, then submit the final extraction result as a PTDI recommendation to the data curation system user. In our evaluation, which will be presented in Section 4.4.3, we decided to generate fully automated results, and then let test users judge each individual step's results.

Transformation Heuristics

Having introduced the general functionality of the DeExcelerator, we will now describe the individual phases in more detail, and discuss the set of default heuristics implemented for each step.

Import In this step, all input file formats are converted to the abstract representation described above: a matrix of string values with optional per-cell metadata, and a global key/value metadata structure for the table. The DeExcelerator currently supports CSV, XLS and HTML files as input. Conversion from CSV is straightforward, while in the Excel case, data types, formatting options, attached formulas and cell merges are saved as additional per-cell metadata.

The HTML case is more complicated. Table detection in an HTML document can be performed seemingly trivially by extracting all DOM subtrees below `<table>` tags, but as shown in (Cafarella et al., 2008), most tables in HTML documents are actually used for layout. The DeExcelerator therefore uses trained classifiers to distinguish layout from relational Web tables and extracts only the relevant ones. These classifiers correspond to those we employed for the extraction of the Dresden Web Table Corpus, so details about them can be found in (Eberius et al., 2015a).

In the case of Excel and HTML documents this step may produce more than one input matrix, even before the explicit Table Detection in the next step. For Excel, this is due the fact that single files may explicitly contain several spreadsheets, which in turn may contain several tables. The spreadsheets are separated in the import phase, because they are separated explicitly

in the file format and can be therefore split trivially. In the HTML case, multiple `<table>` tags lead to multiple input matrices and therefore multiple instances of the DeExcelerator process well.

Table Detection The aim of this step is to guarantee that all further processing steps work on exactly one table, i.e., there is one connected area of data and only one header in the matrix under consideration. No information is dropped in this phase, the matrix is only split if necessary. The following heuristics are part of the DeExcelerator core.

- Empty Regions: The matrix is traversed row-wise, and contiguous regions of rows with no more than one non-empty cell are marked. A region ends as soon as a row with more than one non-empty cell is encountered.
- Title Rows: Rows containing a single non-empty cell with above average string length and at least one formatting option are signals of a table title.

If more than one of these signals is detected, this indicates that more than one table is present. As discussed above, even if several tables are detected, in many real-world spreadsheets they may actually be partitions of the same logical table. Therefore, as a secondary step, for a region following each boundary another check is performed. Cells with equal position with respect to each boundary are compared for fuzzy equality. In case equalities are found, equal rows (likely header and title) are eliminated, and the matrix is not split, but continues further processing as a unit.

Layout Cell Removal This phase is aimed at removing rows and columns that do not carry neither data nor metadata. As a first heuristic, completely empty rows and columns are trivially removed, as they contain no information and were likely introduced for layout purposes. Further, the DeExcelerator core contains another heuristic for merging rows that were introduced to force line breaks, as shown in Figure 4.20. For all rows in the matrix that have below average number of non-empty cells, these cells are checked for a possible merge with the respective cell below them. The DeExcelerator uses data from a large nGram corpus[6] to decide whether the merged cell content would form more meaningful content than the individual contents. If all cells on such a sparsely populated row can be meaningfully merged with the cells below them, then the containing row can be removed by merging those cells.

Metadata Extraction The aim of this step is to remove all cells from the matrix that contain textual metadata. Firstly, we observed that accompanying description text is typically placed before the header, and footnotes and remarks below. The DeExcelerator core contains a simple heuristic to capture these classes of metadata. The matrix is traversed row-wise, both bottom-up and top-down. As long as rows with only one non-empty cell are encountered, these rows are removed and their content is stored in the global metadata structure of the extraction process. Secondly, many tabular datasets contain inline comments in numeric columns, usually comments on specific values. So as a second heuristic, the DeExcelerator checks for columns in which the majority of cells starts with a numeric value. In such columns, it moves existing

[6] Microsoft N-Gram Service: `http://weblm.research.microsoft.com/`

textual suffixes in the number columns into the per-cell metadata structure. This does not only increase reusability by separating metadata and data, but also supports the correct data type recognition later in the process.

Finally, the DeExcelerator also contains a specialized heuristic for recognizing the name of a table as an explicit special type of textual metadata. It analysis the formatting applied to the first few rows of the matrix, i.e., it looks for underlined, bold or higher font size cells.

Header Separation This phase splits the remaining matrix into a header, containing only attributes, and a data region, containing only data tuples, as shown in Figure 4.21a. These two parts are then processed separately in the DeExcelerator process (Figure 4.24). The DeExcelerator currently implements the following list of heuristics, whose output is combined to decide on a separator.

- Date: As soon a date cell appears, the header ends one row above it.
- Background color: A change in background color (in a spreadsheet) signals the beginning of the data part.
- Number sequence: The beginning of a numeric sequence in one column signals the beginning of the data segment.
- Number: A number in a cell is also evidence for the beginning of the data section, albeit a weaker one.
- String series: A sequence of cells in a column with the same value signals the beginning of the data block. Such sequences are usually a product of denormalization, e.g. they are keys in some hierarchical dimension.
- String length: The same logic can be applied to several cells in one column with equal length. These are usually constant length identifier codes.

If multiple heuristics discover conflicting splits, a decision is reached as described at the beginning of this section. After a split has been applied, the following phases can process data and header rows separately.

Header Analysis The previous phases have removed textual metadata above the header, and separated the data below the header. The aim of header analysis is to transform the remaining region of cells into the correct flat attribute names for each column. Consider again Figure 4.21b as an example. The DeExcelerator uses the following heuristic for flattening the header: First, all merged cells are split, projecting their content onto the newly created cells. Next, all cells are merged from top to bottom, concatenating their text content to create flat attribute names for each column.

Data Type Recognition The DeExcelerator contains parsers for various data types such as dates, currencies and URLs, and differentiates between various floating point and integer types. It also defines type compatibility hierarchies, i.e., it defines which types can be expressed in terms of which other types. The parsers are used to find a most specific type for each cell in the data region. Then for each column, a single type is determined by majority voting, albeit taking compatibility hierarchies in account to find a type that is general enough to express all values in the column.

Implied and Null Value Detection The aim of this phase is to fill in the last empty cells in the data while differentiating between implied values and null values. The DeExcelerator uses the following heuristic: First, it traverses matrix columns left to right. In each visited column, values are then extrapolated top to bottom, i.e., each cell projects its value to all empty cells below it. However, as soon as a non-string column is encountered, the procedure changes and empty cells are not filled, but marked as null values instead. The intuition between this simple heuristic is that the left-most columns of tables typically contain identifiers or entities, which apply to following rows as well if cells are left blank. This approach assumes that as soon as a non-string column is encountered, the rest of the table contains measures and not identifiers.

Dependent Cell Removal Finally, removing functionally dependent rows and columns is the aim of the last phase. Fully automatic table normalization is out of scope for the DeExcelerator, but it contains heuristics for detecting and eliminating sum rows and columns, which are very common in real-world tables. To this end, the DeExcelerator core contains a brute-force and a label-based heuristic. The brute-force approach takes the power set of columns, and for each combination of columns checks whether their sum corresponds to any of the columns not part of the respective combination. Obviously, this brute-force approach only lends itself to small numbers of column, e.g., less then ten columns, as the powerset grows exponentially. Note, however that this sum has to be checked only for a few rows, because a dependency, if there is one, will likely hold for any row of the table.

Additionally, for larger tables where the brute-force approach does not scale, a second heuristic is used to prune most columns, by checking the identified attribute names: the DeExcelerator uses a dictionary of typical signaling words and phrases, such as "total" and "sum of" to identify candidate sum columns, pruning most of the search space. Of course, both of these approaches can not recognize complex functional dependencies, but for the ubiquitous summation columns of most real-world tables, it is very applicable.

Export The output of the DeExcelerator pipeline is data in relational form, a schema in the form of per-column attribute names and data types, as well as global and per-cell textual metadata. It offers several formats for export that are optimized for various reuse cases.

- **CSV + SQL**: The data region is dumped as a properly formatted CSV, the attribute names and data types as a SQL `CREATE TABLE` statement. This format is meant for reuse in a relational database or warehouse context.

- **JSON**: In this format, a single JSON document is created, which contains the data as a multi-dimensional JSON array with correct typing, and the extracted schema and meta data as further key value structures in this JSON document. This format is most useful for reuse in general programming and analysis environments, as the majority of them offer functionality for deserializing JSON data into native objects of respective environment.

Having detailed the phases of the DeExcelerator process and discussed the set of heuristics bundled with it, we will now present the results of our evaluation.

4.4.3 Evaluation

We conducted an evaluation of the DeExcelerator using about 2,000 Excel files randomly downloaded from `data.gov`. Our evaluation has two parts:

- **Relevance**: To evaluate the relevance of the challenges we identified in Section 4.4.1, we ran a fully automated normalization of all 2,000 Excel files using the DeExcelerator. We collected statistics on how often each challenge was encountered, and which of the heuristics were applied.

- **Quality**: While the first experiment shows the relevance of the DeExcelerator and provides an understanding of the different denormalizations addressed, it is no indication of the quality of the actual results produced. We therefore conducted an user study, in which ten database students where asked to rate the success of the various extraction and cleaning steps for 50 sheets from our evaluation set. This experiment gives an impression of the efficiency of the abstract process, the included concrete heuristics, and also what is possible in general with a fully automated extraction system when applied to real-world spreadsheets.

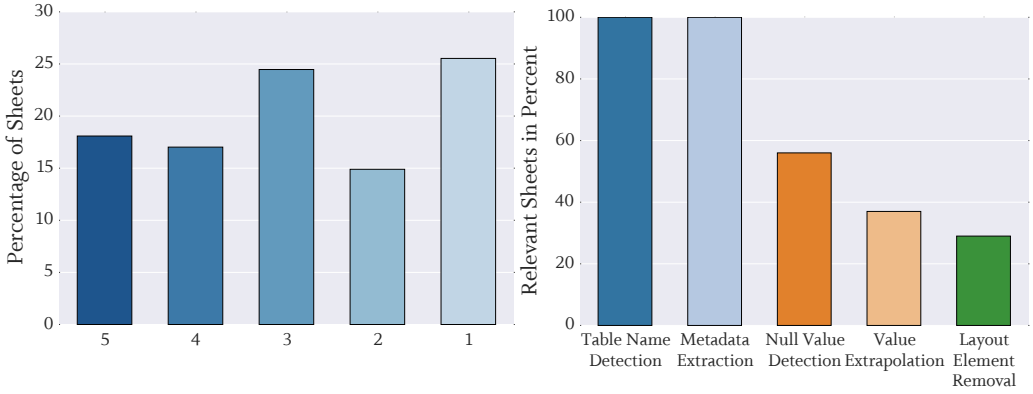
Relevance Evaluation

We processed 2,000 Excel documents from `data.gov`, which contained about 4,600 sheets. Of those 4,600 sheets, only 86% contained tables, while the others mostly contained single textual metadata cells describing the other sheets. Those about 4,000 non-empty sheets, however, contained about 5,000 tables, which shows that spreadsheet authors in general do not follow an "one sheet, one table" usage pattern. More precisely, 387 sheets (9%) contained more than one table, which indicates the necessity of the *Table Detection* step.

The *Layout Cell Removal* step was relevant in 30% of the documents, and about 1,200 layout lines were detected and removed. To give an example for the *Metadata Extraction* phase: In over 3,300 sheets, at least a title could be extracted from the sheet itself, which demonstrates the usefulness of searching for textual metadata within the sheet. Concerning the *Header Separation*, a notable result was that the header was confined to exactly the first row of the spreadsheet in only 10% of the surveyed cases. In the vast majority of cases, however, the header was on a different row, or even spanned several rows. Therefore, manual preprocessing effort would be necessary to resolve the schema of the spreadsheet, complicating reuse. We made similar observations with respect to the other phases, e.g during *Depend Cells Removal*, about 350 sum columns and about 4,000 sum rows were identified. However, to evaluate the quality of the proposed transformations for the individual steps, we will now discuss the user study we performed.

Quality Evaluation

To evaluate the quality of the proposed transformations, we handed out 50 sheets, each two times, to ten volunteer database students. Since the real-world relevance of the normalization problems we tackle was already shown in the previous section, the subset used for the



(a) Perceived Difficulty of Extraction, from Very Difficult (5) to Very Easy (1)

(b) Applicability of Transformation Steps

Figure 4.25: General Properties of the Evaluation Sheets

quality evaluation was deliberately chosen so that each sheet contained as many of the studied defects as possible. The students were provided with the original table, as downloaded from `data.gov`, as well as with the relation, schema and metadata as extracted by the DeExcelerator. These results were likewise presented in a spreadsheet format, for easy side-by-side comparison. The students were furthermore provided with a questionnaire. One question was to rate the perceived "difficulty" of automatically extracting a perfect relation from each document, on a scale from 5 (very difficult) to 1 (very easy). Since we only invited students with a solid database background, all of them were familiar with relational normal forms and problems of data cleaning, i.e., they were able to understand the studied problems with little introduction into the experiment. Furthermore, they were encouraged to look at all ten sheets assigned to them to get an impression of the relative difficulties. The results of these difficulty assessments are shown in evaluation Figure 4.25a. It shows a relatively even distribution of difficulties, indicating a good mixture of sheets picked for evaluation.

Apart from this initial difficulty ratings, the candidates were asked to rate success of the following transformations: *Table Name Extraction*, general *Metadata Extraction*, *Null Value Detection*, *Implied Value Extrapolation*, *Layout Cell Removal*, *Header Recognition* and *Data Type Recognition*. Note, that not all of the transformations where necessary in all sheets. The two steps *Header Recognition* and *Data Type Recognition* apply to all tables per definition, but the other five steps not necessarily do so. Therefore, the volunteers were also able to answer "not applicable". Figure 4.25b shows the applicability of these five steps as judged by the participants. We can see that all 50 sheets in our test set contained textual metadata, and all of them had an embedded title that the system had to extract. Null values were present in 56% of the sheets, while implied values and layout cells where part of 37% and 29% respectively. This confirms the results of our automatic relevance evaluation, showing that the studied normalization problems regularly appear in real-world spreadsheets.

Finally, for all applicable transformations of a sheet, the participants rated the success of the transformation on a scale from 5 (very successful) to 1 (not successful). The results of this main part of the questionnaire are shown in Figure 4.26. Generally, most tasks were solved

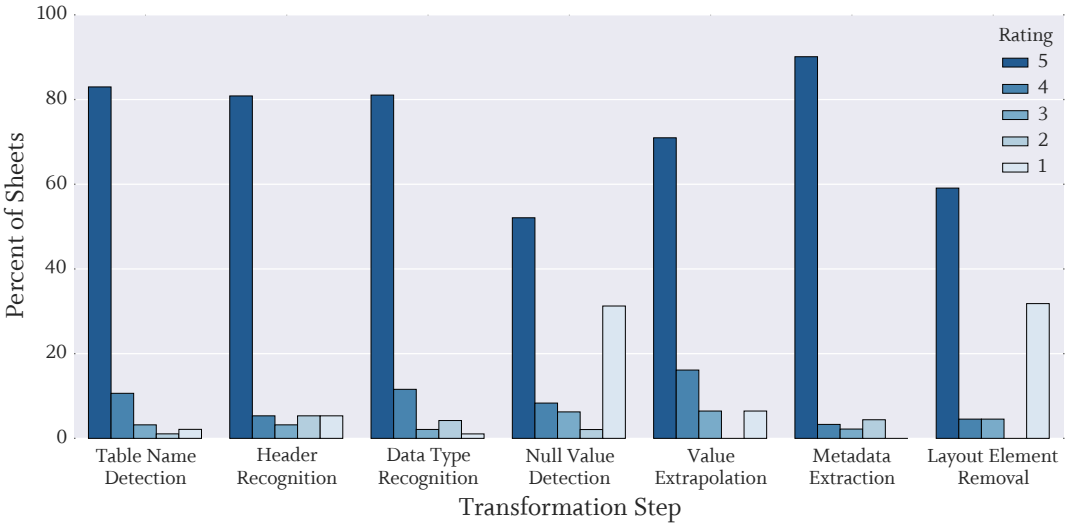


Figure 4.26: Extraction Success from Very Successful (5) to Unsuccessful (1)

well, especially considering they were solved by a fully automated system. For all transformation steps, the rating "very successful" was chosen in more than 50% of the cases. In four categories, the top rating was even given to more than 80% of the sheets. More so, these four most successful transformation steps are the most relevant and important steps for the general case: *Table Name Detection*, *Header Recognition*, *Data Type Recognition* and *Metadata Extraction*. These are the most important steps, as they are relevant to all tables, and not only a subset of them, as was shown in Figure 4.25b. Typical failure cases in these important categories where failure to recognize title if there were multiple title parts, or failure to recognize headers consisting of mostly numbers, which the currently implemented heuristics recognized as data values.

The biggest problems were found in *Null Value Detection* and *Layout Cell Removal*. In both cases, for roughly 31% of the sheets these transformations were rated as "unsuccessful". In the former case, the heuristics existing in the DeExcelerator were not sophisticated enough to deal with various textual forms of null values, such as "n/a", "–", or the use of inline footnotes in cells. In the case of Layout cells, there are too many ways in which a human authors can use whitespace or line breaks to visually structure the spreadsheet to capture all of them with simple heuristics. For these cases, more sophisticated techniques would need to be integrated into the DeExcelerator framework. We will present an overview of candidate techniques in Section 4.5.

Implementation and Additional Materials

In addition to the concepts and evaluation data we presented in this section, we publish a usable implementation of the DeExcelerator, including a Web interface which visualizes the steps of the extraction pipeline, and shows the input document as it is iteratively transformed into a relational table. The interface also displays the results of the heuristics implemented in each

step, and thus allows the user to understand each phase. The DeExcelerator website[7] features a screencast, the URL of the actual running DeExcelerator GUI, as well as the URL of the source code.

4.4.4 Conclusion

In this section, we have presented the DeExcelerator, a framework for extracting relations from partially structured documents. We surveyed a corpus 2,000 of real-world public spreadsheets, and compiled a set of typical spreadsheet denormalizations, i.e., usage patterns that hinder automated reuse by third parties. From this set we derived a framework of consecutive abstract transformation steps aimed at converting a partially structured document into a traditional relation with accompanying schema and metadata. We then described the set of concrete implementations for each transformation that are part of the DeExcelerator core, and were used in the evaluation. Finally, we performed a two-part evaluation, in which relevance of the studied denormalizations as well as the effectiveness of the proposed transformation pipeline were examined. We could show that the problems tackled by the DeExcelerator are relevant on a real-world Open Data platform, and that the majority of these problems can be solved using our automated heuristic transformations.

The DeExcelerator is integrated with the Publish-time Data Integration concept in the form of a PTDI operator. It produces transformations that can be presented to publishers as PTDI recommendations. These transformations are generated automatically, with no user effort except verification. This effort is also scalable in the sense that users can be asked to verify the whole process, individual transformation steps, or even choose from possible results produced by multiples heuristics in an individual phase. It facilitates reuse of published data by transforming it into a standardized form. Since it is designed as an extensible framework, it can also be adapted to evolving usage of the data curation system.

4.5 RELATED WORK

In this chapter, we have discussed data integration challenges that arise on new forms of collaborative data management platforms. We proposed the concept of a Publish-time Data Integration platform, on which authors, as well as other users, are encouraged to increase the reusability of individual datasets as they are published on it. Essentially, the proposed concept aims at saving cleaning and integration effort for data reusers. This is done by automatically creating various improvement recommendations for datasets at publish-time, when the superior background knowledge of the publisher is still available, as opposed to at reuse-time, when this knowledge has to be reconstructed to understand a dataset.

However, the aspect of publish-time integration is not the only feature we consider necessary for a successful data curation system, and the methods we propose could be improved significantly. To put this work into context, we will study various areas of data integration, especially of non-traditional forms that do not assume full integration of data sources as their aim.

[7] `http://wwwdb.inf.tu-dresden.de/edyra/DeExcelerator/`

4.5.1 Schema Matching, Entity Resolution, Data Fusion

The methods presented in this chapter are based on similarity measures from schema- and instance matching research. The aim of this research is to automate the discovery of correspondences between semantically equivalent elements of different schemata. This large field of research that has been in focus of database research as an independent topic for more than ten years (Bernstein et al., 2011). The more general schema integration problem, however, has been studied for much longer, as the survey (Batini et al., 1986) shows. Research into heterogeneous, federated database systems that tackles similar problems goes back into the same era, e.g., (Sheth and Larson, 1990). The goal of schema integration is generally to form a unified schema for several data sources to enable joint queries over them. This is typically accomplished by either creating mappings from the individual schemata to a defined global schema, an approach often called local-as-view, or by creating a merged, global schema from the individual schemata, often called global-as-view (Lenzerini, 2002). Although our study of Open Data platforms in Section 4.1 established that platforms with integrated schemata are preferable to loose dataset collections, full integration is not realistic for many data platforms.

Returning to the more fundamental step of schema matching, however, one central aspect of this research is an important building block of our methods, namely similarity measures for schema elements. There is a large array of schema similarity measures that have been proposed, based on linguistic or structural properties of elements, on external reference data such as ontologies, or even on usage of the elements in queries. There are several well-known surveys that give an introduction into this area, such as (Rahm and Bernstein, 2001) and (Shvaiko and Euzenat, 2005) for schema matching, but also in the field of ontology matching, such as (Shvaiko and Euzenat, 2013).

A equally important body of research aims at identifying equality relations between entity references in different databases, i.e., matching on the level of instances. This process is referred to by various names, which include entity matching, deduplication, record linkage, reference reconciliation or entity resolution. An early foundational work that models the problem using a probabilistic model is (Fellegi and Sunter, 1969). Early mentions in the database community include (Hernández and Stolfo, 1995), while current surveys and benchmarks can be found in (Köpcke et al., 2010; Köpcke and Rahm, 2010) and (Koudas et al., 2006). Though the base similarity measures used are alike to those used in schema matching, the field is characterized by its own set of problems. This includes dealing with the larger number of comparisons that become necessary compared to matching schema elements, or taking the attribute structure of entities into account. As with schema matching techniques, efficient and effective entity matching methods are a prerequisite for identifying similarities between datasets, and thus for our methods of generating PTDI recommendations. In our evaluations, we used only basic string- and token-based similarity measures, and naïve cross product comparisons between datasets, as we consider these problems orthogonal to the ones studied here. However, a fully developed PTDI platform would need to incorporate state-of-the-art schema and entity matching methods for reasons of efficiency and effectiveness.

In general, the most complete form of integration is called *data fusion* (Bleiholder and Naumann, 2009; Dong et al., 2014b). It aims at presenting the user with a unified views of a set of heterogeneous data sources, which allows to retrieve data from all of them with a single query,

and also with deduplicated and merged instances extracted from the various sources. The data curation systems that we studied in this chapter have much different goals. They aim at providing a platform for publishing reusable individual datasets that can not realistically be fused due to a multitude of domains or lack of resources for a full-scale integration.

A final class of related work from schema matching that we need to mention is *corpus-based* or *statistical* schema matching. While most schema matching methods essentially focus on pairwise similarity computations between two schemata, these techniques leveraging additional knowledge provided by a large corpus of schemata to improve their matching process, and thus share basic concepts with our approach presented in Section 4.3. Firstly, (Madhavan et al., 2005) applied machine learning techniques on an existing schema corpus with known schema mappings in order to improve accuracy of further matches. Secondly, in (He and Chang, 2003), the authors observed that with a growing number of data sources in a corpus, the aggregated schema vocabulary converges. Therefore, they try to find a hidden probabilistic model within a large set of input schemata to match all of them, instead of creating matches between each pair of them.

4.5.2 Iterative and User-driven Integration and Cleaning

A majority of the work in the areas mentioned so far focuses on "one-shot" integration, i.e., on supporting scenarios in which all databases that are to be integrated are known a-priori. This kind of process has a clearly defined end: the complete integration of the chosen systems, be it the consolidation of two warehouses after a company merger, or the creation of a unified query interface for a set of existing systems. However, there is also related work into more open-ended integration scenarios, similar to publish-time integration in data curation systems. Early research includes for example (Rosenthal and Seligman, 1994), that studies rules for general data reusability over many integration scenarios and proposes creating repositories for metadata that can be reused over many such integration processes.

More work proposing iterative processes can be found in the area of *data cleaning*, which is the area of research dealing with detecting and removing errors in data, with the aim of improving quality (Rahm and Do, 2000). Classic examples of such errors include misspellings or missing values in newly entered data.

A exemplary work in interactive data cleaning is (Raman and Hellerstein, 2001), in which the steps of error detection and transformation authoring are unified in an easy-to-use spreadsheet like interface. The system automatically infers structures for values while the user is browsing them, flags discrepancies with the inferred structure, and allows the user to graphically model transformations using a set of simple operators. The structure inference is iteratively applied on the user-transformed results to test whether the transformed data is now regularly structured or further transformations are necessary. This results in a closed loop of the system detecting anomalies and the user reacting to them. This is similar to the methods proposed in this chapter, where the system proposes changes to datasets as they are published, asking the users to verify those change recommendations.

Furthermore, there are other approaches that seek to profit from iterative integration and cleaning by involving user-feedback. An example would be (Yakout et al., 2011), which applies active learning with user-feedback to data cleaning, specifically to repairing a database

with conditional functional dependencies. In the proposed *guided-data repair* system, a system-generated list of possible repairs is ranked so that the user's feedback is most likely to have positive impact on the overall database state. The feedback is not only used to verify candidate repairs, but also to update the list of generated repairs using the active learning feedback. Thus, a closed feedback-loop is used to iteratively improve the quality of the database. This approach is very similar in spirit to the PTDI-approach where system-generated recommendations are verified by the user to improve dataset reusability with minimal user effort.

There are more recent approaches to integration and cleaning on large data collections, that operate iteratively over time, instead of using monolithic full-integration processes. These approaches stem from the areas of *dataspace systems* and *pay-as-you-go integration*, which we will survey next.

4.5.3 Dataspace Systems and Pay-as-you-go Integration

New forms of data management such as dataspaces and pay-as-you-go data integration have been discussed in recent database research. The term *dataspace* originates in (Franklin et al., 2005), where the authors formulate their vision of dataspaces as an alternative to traditional databases, and propose *dataspace support systems* for providing query and integration services in this context. Dataspaces follow a data-coexistence approach, where heterogeneous data sources can coexist in one system, without explicit up-front integration. They are strongly related to our notion of data curation systems in that they assume large sets of disparate data sources lacking a global schemata, which still should be queried uniformly. Also similarly to our concept, the overall integration quality of the collection is continuously improved while it is already being used.

The methodology for integrating data sources only as they are used, i.e., only as the integration becomes relevant, is typically called *Pay-as-you-go integration*, a term introduced in (Madhavan et al., 2007). These methods are applied when the data that is managed is to large or to heterogeneous to be fully integrated up-front. An example would be an integrated query interfaces for the Deep Web, which contains millions of possible data sources with different schemata from different domains, making a global schema infeasible. The original paper proposes concepts such as schema clustering instead of mediated schemata, approximate schema matches instead of schema integration, keyword queries instead of structured queries, and ranked heterogeneous results instead of single definite query answers. Concerning gradual integration, the authors propose a combination of automated schema and instance matching techniques, whose results are to be verified by the dataspace's users.

While (Madhavan et al., 2007) only gives a high-level architecture and a motivation for such pay-as-you-go systems instead of detailed algorithms or techniques, it inspired several follow-up works. One of the first implementations of a dataspace system was (Vaz Salles et al., 2007). It combines a general graph data model for ingesting various data sources with a query expansion and rewriting system based on so-called *trails*, hints that unify attribute mappings and keyword expansions. The system allows to gradually and declaratively add schema and integration information via these trails to improve query results over time.

In (Jeffery et al., 2008), the authors tackle the problem of entity resolution between various data sources in a dataspace by utilizing user feedback to confirm possible matches. Since

the number of potential matches in a dataspace is assumed to be to high to verify all of them manually, the paper focuses on ranking potential matches for user verification. They propose a method based on the *value of perfect information* to rank a set of candidate matches for verification in order of utility for the dataspace as a whole. Specifically, the expected utility gain of confirming a match is estimated using the increase in answer size after confirmation of the match, given a specific query workload. In the PTDI context, methods such as this one could be applied for ranking system-generated recommendations based on the global utility for the data curation system.

Another pay-as-you-go approach that aims at piggy-backing a data integration task on other user interactions was presented in (Tran et al., 2012). In this work, the authors present an entity search engine, which presents a list of entity-clusters as a query results, instead of a list of correctly reconciled single entities. Each clusters represents a set of possibly equivalent entities as determined by the underlying consolidation system. By selecting clusters from the list, the user implicitly confirms that the clustered entities refer to a single real-world entities, i.e., the user search interaction is used as a validation of the automatic entity reconciliation. This is similar to our notion of linking one user action, the publication of a new dataset, with another one that benefits the system as a whole, the improvement of dataset-reusability.

Finally, there is recent work on integrating many individual approaches into a comprehensive, usable software platform that can be applied to a real-world "open" or Web data management scenario in (Ives et al., 2015). The aim is to produce an end-to-end solution for the new forms of data integration as presented in this section, enabling researchers to study real usage patterns of such techniques, and enable field experiments with these new methods on real data with real users. A practical, commercial implementation of a platform for Web-based, collaborative data management called *Google Fusion Tables* was presented by (Gonzalez et al., 2010). It allows users to upload datasets, share, annotate, combine and visualize them, while focusing on usability. Its main contribution is an intuitive, fully featured Web interface designed to make data management accessible for new classes of users. In (Stonebraker et al., 2013), the authors discuss a commercial offering, called *Tamr*, which implements a similar platform, here called *data curation system*, and describe their experiences in several industrial scenarios. Tamr includes methods for ingesting data in various formats, matching their schemata and the contained entities, while allowing various levels of integration from no central schema up to a global, mediated schema.

In conclusion, there is a large body of work from the domain of dataspaces, pay-as-you-go integration and similar systems for "open" integration scenarios, which is highly related to our work on Publish-time Data Integration for data curation systems, which illustrates the relevance developing new methods in this area.

4.5.4 Crowdsourced and Community-driven Data Integration

There is an increasing number of works on incorporating human work directly into computations, for example about processing database queries (Franklin et al., 2011), sorting and joining (Marcus et al., 2011), or graph search (Parameswaran et al., 2011), and even particularly for data integration using the crowd, such as in (McCann et al., 2008; Hedeler et al., 2011) or (Wang et al., 2012).

In general, these techniques differ from the user-driven methods discussed above in that they do not utilize the capabilities of a single, dedicated user, but of a large community or a large crowdsourced work force. The difference thus lies between including the user that is already actively involved in using a system directly in its processing versus including a large group of potentially unconcerned workers to support some process. A typical technique used in these works is to use a crowdsourcing platform, Amazon Mechanical Turk[8] being the most well-known one. On these platforms, one group of users can submit so-called *human intelligence tasks* or HITs, which are defined as a minimal unit of human work, usually to be accomplished in a span of seconds to a few minutes. Typical examples include making one multiple-choice decision, e.g. picking a category for some object, or entering one piece of information, e.g., a short description for an image. Another group of users, called *workers*, solves these tasks, usually for monetary compensation. Methods for crowdsourced data integration could be applied to data curation systems if there is sufficient motive for the users to participate in integration tasks. For example, on Open Data platforms that are of public interest, these methods might replace of supplement the paid workers of a crowdsourcing platform.

Research into crowdsourcing for data integration has been conducted in many directions. For example, there are end-to-end solutions to crowd-based entity resolution such as (Gokhale et al., 2014), which aim at lowering the skill requirements for applying such methods, e.g. removing the need for any coding on the part of the user. It is clear that such methods are to be favored for data curation systems, where the dataset publisher as the user can not be expected to have adequate skills to handcraft a crowdsourcing solution, but may want to utilize one to improve reusability of his or her contributions. Furthermore, crowdsourcing techniques must be applied very carefully in order to keep control of the costs. One critical issue are the user interfaces and questionnaires with which the HITs are performed by the workers. Their design can have a huge impact on the costs and the result accuracy. Marcus et al. (Marcus et al., 2011) demonstrated how different UI design results in more than an order-of-magnitude cost reduction while maintaining accuracy and latency. Another problem is the decomposition of tasks in smaller subtasks that can be handled by the crowd. CrowdForge (Kittur et al., 2011) is addressing this issue by leveraging a MapReduce-style system for task decomposition and verification. Both the question of cost management and task granularity are relevant to data curation systems as well, as user integration effort is a scarce resource, and wrongly posed tasks will discourage engagement with the system.

These are just some examples of relevant work in crowdsourcing in data management. Multiple surveys on the subject, such as (Doan et al., 2011) and (Chen et al., 2015), are already available. In conclusion, since we assumed data curation systems to be multi-user collaborative environments, data integration methods based on crowdsourcing can be applied to solve problems arising from the free-for-all nature of these repositories.

4.5.5 Linked Open Data

One group that is particularly committed to improving Open Data publishing is the Linked Data and Semantic Web community, which utilizes technical standards such as RDF and SPARQL. These technologies aim at publishing structured data on the Web, while enabling uniform ac-

[8] `https://www.mturk.com`

cess, semantic annotation, and linking of data. The universal data model used breaks all data down to subject-predicate-object triples. These triples are used to express both data, i.e., facts and links to other data, as well as metadata, such as links to ontologies (also expressed in triples), that define the semantics of the data. A good, concise introduction is provided in (Bizer et al., 2009a).

As mentioned, the community is very engaged in research regarding publishing of reusable data, i.e., in the publishing of Open Data. (Berners-Lee, 2010) introduced the "five stars of linked open data", which define levels of openness, that constitute a high-level guide for publishing reusable data. The levels in ascending order are: making data available on the Web, making it available in a machine-readable format, choosing a non-proprietary format, using URLs to identify objects, and finally linking the data to other data published on the Web, to provide context.

To reach these goals, a large body of research has been conducted. To construct a "nucleus", or general purpose global schema, for the Web of Data, (Auer et al., 2007; Lehmann et al., 2015) presented the DBPpedia, a large-scale general ontology extracted from Wikipedia. At the time of writing it contains about 4.5 million entities[9] with the majority being mapped to a consistent ontology, all of which become centrally identifiable through an unique URL. It now serves as a hub in the Web of Linked Data, being the dataset with the highest in-degree[10], i.e., it serves as a central entity reference for other datasets published as Linked Data. Note, that there are various other general, public ontologies besides DBpedia, e.g., YAGO (Suchanek et al., 2007) or the Google's Knowledge Vault (Dong et al., 2014a). For data curation systems such ontologies could be an alternative to a specifically designed global schema, by linking attributes and entities of published datasets to a common ontology, aiding discoverability and recombination of datasets.

For supporting the publishing process, there are various projects aimed at making specific non-linked Data available in the Semantic Web, e.g., for transforming relational databases to triples, as surveyed in (Sahoo et al., 2009; Spanos et al., 2012). Furthermore, there are works aimed at automatically creating links between datasets, e.g., general link discovery systems such as (Bizer et al., 2009b), but also systems with the user in the loop (El-Roby and Aboulnaga, 2015), similar to the related work we discussed above. Finally, in the Linked Data community, there are works on experiences and best practices of publishing reusable data, and case studies that show the use of Open Data and potential benefits (Shadbolt et al., 2012; Lopez et al., 2012; Böhm et al., 2010).

Though there is a large body of work on Linked Data, only a minority of the data published on Open Data portals, or in scientific research is published in this form, as shown for example in (Martin et al., 2011) or our own Open Data study (Section 4.1). Traditional formats such as relational databases, spreadsheets or even just CSV files are still easier to produce and handle in most contexts, and require no Linked Data-specific knowledge or tools. More importantly, almost all widely-used methods and technologies for data analysis, be it traditional warehousing, data mining, machine learning or predictive analytics are based on relational data, with some methods also allowing multidimensional or hierarchical data. This even results in research on how to transform Linked Data back into a form usable for OLAP (Kämpgen and Harth, 2011).

[9] `http://wiki.dbpedia.org/about`

[10] `http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state`

This is a major reason for making Linked Data an uncommon choice in the enterprise context. However, the experiences gained in (Linked) Open Data research are highly relevant to designing data curation systems, even if Linked Data as a technical format is not the best choice for the general case.

4.5.6 Attribute Synonym Discovery

While the above section dealt with work related to PTDI-enabled data curation systems in general, in this section we will specifically discuss work related to finding attribute name equivalences as in Section 4.3.

In the *WebTables* paper, (Cafarella et al., 2008) discuss how to extract a large corpus of millions of relational data tables from the Web, and give several application scenarios for the novel kind of data corpus created this way. Specifically, one application was to use attribute co-occurrence statistics collected from their large-scale Web table corpus to discovery attribute synonyms. The intriguing aspect of this solution is not only that it discovers new synonyms for single attributes in isolation, but also that it is context-sensitive with respect to the other attributes in a given schema. For example, the two attributes *name* and *filename* are synonymous, but only in the context of file related schemata. This approach could be combined with ours, as it focuses entirely on the schema, using only co-occurrence statistics and ignoring instances, while our approach considers the schema only in the domain classification step (Section 4.3.2), and then relies solely on the instances for the equivalence discovery.

Both (Cafarella et al., 2008) and our approach, create their output based entirely on a corpus of heterogeneous datasets, and do not require external knowledge or global schema as a reference. There are other approaches from the area of Web data integration that aim at recovering the semantics of tables extracted from the Web, which often means finding meaningful attribute names for table columns with no or indescriptive headers. Examples are (Limaye et al., 2010) and (Venetis et al., 2011), which we already discussed in Section 2.6.1. These methods have their reliance on large and clean external knowledge bases in common, while our methods were designed to work directly with the knowledge implicit in the corpus of datasets on a given data curation system. Still, since large-scale knowledge bases such as YAGO (Suchanek et al., 2007) are increasing in size and quality, combining these methods for even better attribute name recommendations seems like a worthwhile next step. Furthermore, there are approaches aimed at detecting *linkage points* between Web data sources, aimed at enabling joins between heterogeneous sources (Hassanzadeh et al., 2013).

In (Chen et al., 2011), a system called *Schemr* is proposed, which is a tool for exploring and locating schemata or schema fragments in large schema collections in order to reuse them in another context. With this system, the user has to provide an incomplete schema as well as optional search terms. By combining schema matching and text search techniques with a structurally-aware scoring metric, Schemr is able to return a set of schemata that potentially match the input schema. Whereas Schemr is more a schema exploring tool serving different types of applications focus we focus on the specific problem of proposing single attributes during data publication.

4.5.7 Spreadsheet Data Management

While the last section dealt with work related specifically to our attribute equivalence algorithms as presented in Section 4.3, we will now discuss related work that is specific to relation extraction from partially structured documents as discussed in Section 4.4.

There are two classes of related work: first, there is a number of tools build to enable a user to extract and clean relational data from source documents, for example Wrangler (Kandel et al., 2011) and OpenRefine (Verborgh and Wilde, 2013), the successor to Google Refine. Both systems offer tooling to help a user working manually on a document cleaning project. In contrast, the DeExcelerator (Section 4.4) is a predefined pipeline that can be applied to large heterogeneous documents collections automatically. The user can be involved by creating new implementations of the extraction steps.

The second category of related work covers table recognition algorithms. They exist for various types of input, e.g. Web tables, Web lists, PDF files and even images, and use a large variety of heuristic, learning-based and even many visual techniques. An extensive overview of these approaches is given in (Zanibbi et al., 2004). There is also more recent work that specializes on data extraction from spreadsheets. (Cunha et al., 2009) describes the transformation of spreadsheets into relational database. They focus much stronger on the intricacies of relational schema design, such as functional dependencies and normal forms. They use the normalized database schema they create to make the spreadsheet queryable, and even to export the data back to a spreadsheet format in third normal form. However, their approach assumes "clean" spreadsheets that are basically in first-normal form and contain none of the denormalizations we identified in Section 4.4.1. The approaches could therefore be combined to allow complete normalization of data even from very dirty input.

In (Chen and Cafarella, 2013), an automatic method for extracting relations from spreadsheets is proposed that is most similar to ours. Similarly to our work, a large corpus of spreadsheets are analyzed to gain an understanding of spreadsheet usage on the Web. Their survey confirms many of our findings, for example the fact that a large percentage of spreadsheets has a multi-dimensional or hierarchical attributes, and thus the need for transformation tools to import the data into relational systems. However, in contrast to our work, it focuses heavily on the detection and extraction of hierarchical or multi-dimensional attribute headers, and do not include other transformation that are part of the DeExcelerator, such as layout cell removal or data type and Null value detection. Still, the approaches presented for attribute detection and hierarchy extraction could be easily included into the corresponding transformation steps of the DeExcelerator due to its framework character. In (Chen and Cafarella, 2014) the same authors extend their techniques with a more sophisticated graphical model for the hierarchy extraction, as well as a semi-automatic component for user-interactive repair, and finally a method for utilizing attribute co-occurrence statistics from a corpus of spreadsheets to further improve the precision of the extraction. These advanced methods lend themselves well to inclusion within the PTDI paradigm, both in using the statistics given on a data curation system, and in including the user in the cleaning process, similarly to our methods in Section 4.3.

As a final note, research aimed at understanding Web tables, as presented in the previous section, is also related to the problem of extracting data from spreadsheets. For example, approaches that map encountered entities into knowledge bases such as (Venetis et al., 2011) are

applicable to spreadsheets as well. While these approaches focus more on specifics of identifying relational tables in large corpora, and semantic annotation, respectively, the DeExcelerator focuses on syntactic artifacts that occur in tables meant for human consumption. Therefore, the DeExcelerator could be seen as a preprocessing step to these techniques.

4.6 SUMMARY AND DISCUSSION

In this chapter, we have studied data curation systems, which we defined as schema-less, collaborative, multi-domain dataset repositories. We discussed examples for such systems, including Open Data platforms, scientific data repositories, and enterprise data lakes. These platforms are characterized by the fact that data is stored there mostly in its raw form, as provided by source systems or human publishers, and is not organized with a central schema. The motivation is to store all incoming data and make it available, even if the future use case is not always known.

We first surveyed a large set of public Open Data platforms to gain real-world usage insights into usage patterns and problems of this new form of data management. The survey results show that those platforms, while containing vast amounts of potentially reusable data, do not support this re-use as well as they could. Especially automated ad-hoc discovery and integration of Open Data, as we aim for in this thesis, is hindered by the lack of unified metadata or even standardized data and file formats. Since traditional full integration is not always applicable for lack of resources on such platforms, the full effort of searching through, cleaning and integrating the data falls to the future reuser. To alleviate these problems, we first introduced requirements for a data curation system that facilitates reuse. To fulfill these requirements, and enable light-weight integration before reuse, we proposed the *Publish-time Data Integration* (PTDI) paradigm. Under this paradigm, the publisher of a dataset is encouraged to optimize the reusability of newly published dataset through automatically generated recommendations, which just have to be accepted or declined by the user.

We introduced PTDI operators, components which are triggered when a new dataset is published, and which generate various types of recommendations to improve the dataset's reusability. We then introduced two specific PTDI operators, one for *generating attribute name recommendations* and one for *extracting relational data from partially structured documents* such as spreadsheets or HTML.

The idea of the first operator is to use statistics on attribute names usage with the datasets already existing in the data curation system, to recommend attribute names that fit well with the vocabulary in use in the system. The motivation for this operator is to constrain heterogeneity, without using a global schema. The generated recommendations are based on instance set overlap as well as a preceding domain classification, in which datasets in the data curation system are automatically clustered according to their schema to identify related datasets. We further presented several versions of the method that use different preprocessing steps to reduce the run-time requirements of the method while keeping precision on the same level as the base method. We evaluated the method concerning run-time, number of recommendations and their precision on real-world Open Data, and used crowdsourcing to evaluate the quality of our recommendations.

The second operator, called *DeExcelerator*, is aimed at transforming partially structured documents, i.e., document in which structured data is freely intermingled with textual or layout elements, into pure relational data with accompanying metadata. We analyzed a collection of real-world Open Data spreadsheets to identify a set of typical spreadsheet *denormalizations*, which are usage patterns that hinder automatic reuse of the data contained in the document. Based on these, we proposed a pipeline of abstract operators that successively remove these denormalizations and transform the spreadsheet. We evaluated the relevance of the denormalizations we identified as well as the correctness of the generated transformations on real world spreadsheet by means of a user study.

To conclude this chapter, we will review the requirements for a system that facilitates dataset reuse in a data curation system, from Section 4.2.1. Firstly, we enable *Standardization without Schema* by generating recommendations that encourage users to publish in a way that facilitates reuse, without forcing them to adhere to a specific schema or strict regulation. In other words, all data is welcome, but the user is supported in increasing data quality. In addition, by basing the generated recommendations on existing content in the system, lightweight integration can be performed without a defined global schema. Furthermore, through encouraging publishers to enrich their data at publish-time, when full knowledge of the dataset is still available, we avoid burdening the reuser with all cleaning and integration effort, thus fulfilling the *Integration before Reuse* requirement. However, the publisher is only likely to accept recommendations and make changes to the data if there are only *Minimal Requirements for the User*. This requirement is reached through fully automated generation of a ranked list of recommendations, involving the user with selection and verification while minimizing recommendation latency. Concerning *Adaptation to Evolving Usage*, we discussed two approaches. The first is a side-effect of basing recommendations on system content statistics. In this way, if the platform's content gradually changes, the recommendations produced by the operators will change as well. Furthermore, by keeping both the library of operators and the operators themselves extendable, PTDI systems can be adapted to changing use cases. Finally, *Scalable Integration* with respect to user effort is reached through using a loosely coupled library of operators and preprocessors, that produce independently usable recommendations, instead of a monolithic integration process that each dataset has to go through. This enables users to pick and choose which recommendations to follow up on, and invest as much effort into the data's reuseability as they are willing to.

Having reviewed our initial requirements, we can conclude that the methods introduced in this chapter allow to greatly increase reusability of data published in data curation systems, without changing their free-for-all nature, and while requiring minimal user effort.

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In the era of Big Data, the number and variety of data sources is increasing every day. However, not all of this new data is available in well-structured databases or warehouses. Rather, data is collected at a rate that often precludes traditional integration with ETL processes and global schemata. Instead, heterogeneous collections of individual datasets are becoming more prevalent, both inside enterprises in the form of *data lakes*, and in public spaces such as Open Data repositories. This new wealth of data, though not integrated, has enormous potential for generating value in situational or ad-hoc analysis processes, which are becoming more common with increasingly agile data management practices. However, in today's database management systems there is a lack of support for ad-hoc data integration of such heterogeneous data sources. Instead, integration of new sources into existing data management landscapes is a laborious process that has to be performed ahead-of-time, i.e., before queries on the combined data can be issued.

In this thesis, we introduced the *Query-time Data Integration* paradigm as an alternative concept. It aims at enabling users to express queries on their own data as if all potential other data sources were already integrated, without declaring specific sources and mappings to use. As a specific example for this query paradigm, we studied *Open World SQL queries*, in which the user may reference arbitrary additional attributes that need not to be defined in the queried database's schema. Relevant sources are then automatically retrieved and integrated at query processing-time, without further input from the user than the SQL query. The ambiguity resulting from the coarse query specification, as well as the uncertainty introduced by relying on automatically integrated data is compensated by returning a ranked list of possible results, instead of a single deterministic result as in a regular SQL query. This allows the user to choose the best alternative for the problem at hand.

To achieve this goal, we developed and evaluated several new methods, algorithms and systems. Firstly, in Chapter 2, we introduced a novel method for *Top-k Entity Augmentation*, which is able to construct a top-k list of consistent integration results from a large corpus of heterogeneous data sources. We showed that our methods improves both consistency and minimality of the augmentation results significantly, without loss of precision or coverage, and while producing diversified result alternatives for the user to choose from. This technique forms the basis for our *DrillBeyond* system, which we introduced in Chapter 3, and which is able to process Open World SQL queries, i.e., queries referencing arbitrary attributes not defined in the queried database. We could show that its hybrid augmentation/relational query processing enables the use of ad-hoc data search and integration for data analysis queries, and improves both performance and quality when compared to using separate systems for the two tasks. Finally, we also studied the management of large-scale dataset corpora such as data lakes or Open Data platforms, as these are used as data sources for our augmentation methods. In Chapter 4, we introduced *Publish-time Data Integration* as a new technique for data curation systems managing such corpora, which aims at improving the individual reusability of datasets without forcing global integration. We designed and evaluated two specific variations of this method, *attribute name recommendation* and *relation extraction*, to demonstrate the viability of the concept.

In conclusion, we have shown how ad-hoc data search and integration in modern data architectures with large collections of heterogeneous data sources can be enabled. First, we

introduced novel, automatic data augmentation methods that harness the large variety of data sources, while requiring minimal user effort. Second, we incorporated those methods into traditional relational DBMS, to enable their efficient use in analytical query contexts. Third, we provided tool support for improving the reusability of newly published datasets. To conclude this thesis, we will now sketch opportunities to extend and build on our contributions in future work.

5.2 FUTURE WORK

In this thesis, we laid the foundation for Query-time Data Integration, by introducing novel methods for ad-hoc data search and integration. However, we still see many opportunities for further research, from enabling more types of Open World queries, over utilizing more types of data sources, to extending our techniques to more use cases. In the following, we will outline possible continuations of our work in a per-chapter basis.

Top-k Consistent Entity Augmentation Possible future work includes a more theoretic analysis of the top-k consistent set cover problem, as well the investigation of the applicability of further set covering heuristics. While we showed that the general mapping of the augmentation problem to the set cover problem is useful in practice, we only conducted limited theoretical analysis, and did not investigate the full spectrum of optimization methods that could be applied to the problem. Further, the specific scoring and similarity functions we utilized were focused on the Web tables scenario. However, we argue that top-k consistent set covering is a general technique that can be applied to many different forms of data sources. For example, there we discussed related work on augmentation based on information extraction from general Web page text. Applying our approach to generate minimal but diverse covers based on Web pages instead of Web tables would be a promising approach to increase coverage. Furthermore, our approach does not yet consider correlations between sources as a factor of trust. Approaches from data fusion literature that detect and utilize such source correlations could be combined with our set covering approach to increase the precision of the generated covers.

Processing Open World SQL Queries With respect to the DrillBeyond system, a possible avenue for future work would be to investigate which parts of the concept could be adapted to modern analytical RDBMS architectures to increase efficiency. In this thesis, we integrated entity augmentation with a classical, single-node row store DBMS. However, in many contemporary scenarios, analytical queries are executed on highly parallel, distributed column stores. Investigating how our proposed architecture and optimizations apply to these systems would increase our method's practical applicability. Furthermore, DrillBeyond so far only allows the usage of additional attributes, i.e., it allows only horizontal table augmentation. However, methods for vertical augmentation, or in other words, the ad-hoc integration of further tuples of an existing relation, have also been discussed in related work. These approaches could be integrated with relational query processing as well. Similarly, the materialization of a completely new relations from Web Data using just a schema description has also been studied in isolation, but could be integrated with general query processing as well. Furthermore, although DrillBeyond does support joins over open attributes, we did not study the optimization of such joins.

In summary, in future work the idea of Open World SQL queries could be generalized from additional attribute to all aspects of SQL and relation query processing.

Publish-time Data Integration The Publish-time integration concept is a high-level methodology, whose success in practice is strongly tied to the specific operators that implement it. In this thesis, we studied two exemplary operators to demonstrate the concepts viability. However, the treated aspect, attribute names and data formats, are only the tip of the iceberg. Many other aspects of data integration that have been studied in literature could be transferred to publish-time. For example, the publisher could get offered recommendations for improved dataset metadata, such as title or tags, or could be asked to verify links to other datasets already existing in the data curation system. Furthermore, all operators we treated in this thesis are local, in the sense that they generate recommendations that concern a single dataset. In future work, it could be beneficial to study global operators, that generate recommendations spanning multiple datasets. For example, if many similar datasets are published to the system over time, a possible future operator could identify their commonalities and automatically suggest the creation of a new dataset category on the platform. More generally, continuous monitoring of the curation system's content could be used to generate additional recommendations, complementing those generated at publish-time. In this way, changing usage patterns or trends in published datasets could be detected that would be unheeded by purely local recommendations.

BIBLIOGRAPHY

A. Abello, O. Romero, T. Bach Pedersen, R. Berlanga, V. Nebot, M.J. Aramburu, and A. Simitsis. Using Semantic Web Technologies for Exploratory OLAP: A Survey. *Knowledge and Data Engineering, IEEE Transactions on*, 27(2):571–588, Feb 2015.

Charu C Aggarwal and Philip S Yu. A survey of uncertain data algorithms and applications. *Knowledge and Data Engineering, IEEE Transactions on*, 21(5):609–623, 2009.

Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying Search Results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 5–14, New York, NY, USA, 2009. ACM. http://doi.acm.org/10.1145/1498759.1498766.

Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: enabling keyword search over relational databases. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 627–627, New York, NY, USA, 2002. ACM. http://doi.acm.org/10.1145/564691.564782.

Anastasia Ailamaki, Verena Kantere, and Debabrata Dash. Managing Scientific Data. *Commun. ACM*, 53(6):68–78, June 2010. http://doi.acm.org/10.1145/1743546.1743568.

Lorena Etcheverry Matteo Golfarelli Jose-Norberto Mazón Felix Naumann Torben Bach Pedersen Stefano Rizzi Juan Trujillo Panos Vassiliadis Gottfried Vossen Alberto Abelló, Jérôme Darmont. Fusion Cubes: Towards Self-Service Business Intelligence. *International Journal of Data Warehousing and Mining (IJDWM)*, (accepted), 0 2012.

Omar Alonso, Daniel E. Rose, and Benjamin Stewart. Crowdsourcing for Relevance Evaluation. *SIGIR Forum*, 42(2):9–15, November 2008. http://doi.acm.org/10.1145/1480506.1480508.

Sihem Amer-Yahia, Pat Case, Thomas Rölleke, Jayavel Shanmugasundaram, and Gerhard Weikum. Report on the DB/IR panel at sigmod 2005. *ACM SIGMOD Record*, 34(4):71–74, 2005.

Paul André, Jaime Teevan, and Susan T Dumais. From x-rays to silly putty via Uranus: serendipity and its role in web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2033–2036. ACM, 2009.

Lyublena Antova, Christoph Koch, and Dan Olteanu. 10$^{\hat{1}}$10$^{\hat{6}}$ worlds and beyond: efficient representation and processing of incomplete information. *The VLDB Journal*, 18(5):1021–1040, October 2009. http://dx.doi.org/10.1007/s00778-009-0149-y.

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, Berlin, Heidelberg, 2007. Springer-Verlag. http://dl.acm.org/citation.cfm?id=1785162.1785216.

Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 261–272, New York, NY, USA, 2000. ACM. http://doi.acm.org/10.1145/342009.335420.

Holger Bast and Ingmar Weber. The CompleteSearch Engine: Interactive, Efficient, and Towards IR & DB integration. In Gerhard Weikum, editor, *CIDR 2007 : 3rd Biennial Conference on Innovative Data Systems Research*, pages 88–95, Asilomar, CA, USA, 2007. VLDB Endowment.

C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4):323–364, December 1986. http://doi.acm.org/10.1145/27633.27634.

Joaquín Bautista and Jordi Pereira. A GRASP algorithm to solve the unicost set covering problem. *Computers & Operations Research*, 34(10):3162–3173, 2007.

J.E Beasley and P.C Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392 – 404, 1996. http://www.sciencedirect.com/science/article/pii/037722179500159X.

Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. ULDBs: databases with uncertainty and lineage. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 953–964. VLDB Endowment, 2006. http://dl.acm.org/citation.cfm?id=1182635.1164209.

Edward Benson and David R. Karger. End-users Publishing Structured Information on the Web: An Observational Study of What, Why, and How. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1265–1274, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2556288.2557036.

Sonia Bergamaschi, Francesco Guerra, and Giovanni Simonini. Keyword Search over Relational Databases: Issues, Approaches and Open Challenges. In Nicola Ferro, editor,

Bridging Between Information Retrieval and Databases, volume 8173 of *Lecture Notes in Computer Science*, pages 54–73. Springer Berlin Heidelberg, 2014. http://dx.doi.org/10.1007/978-3-642-54798-0_3.

Tim Berners-Lee. Linked data-design issues. *URL http://www. w3. org/DesignIssues/LinkedData.html*, 2010.

Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic Schema Matching, Ten Years Later. *VLDB*, 2011.

Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *In ICDE*, pages 431–440, 2002.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

Christian Bizer, Julius Volz, Georgi Kobilarov, and Martin Gaedke. Silk - A Link Discovery Framework for the Web of Data. In *18th International World Wide Web Conference*, April 2009. http://www2009.eprints.org/227/.

Roi Blanco, Harry Halpin, Daniel M. Herzig, Peter Mika, Jeffrey Pound, Henry S. Thompson, and Thanh Tran Duc. Repeatable and Reliable Search System Evaluation Using Crowdsourcing. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 923–932, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/2009916.2010039.

Jens Bleiholder and Felix Naumann. Data Fusion. *ACM Comput. Surv.*, pages 1–41, 2009. http://doi.acm.org/10.1145/1456650.1456651.

Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. SODA: Generating SQL for Business Users. *Proc. VLDB Endow.*, 5(10):932–943, June 2012. http://dx.doi.org/10.14778/2336664.2336667.

Christoph Böhm, Felix Naumann, Markus Freitag, Stefan George, Norman Höfler, Martin Köppelmann, Claudia Lehmann, Andrina Mascher, and Tobias Schmidt. Linking open government data: what journalists wish they had known. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 34:1–34:4, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1839707.1839751.

Ilaria Bordino, Yelena Mejova, and Mounia Lalmas. Penguins in Sweaters, or Serendipitous Entity Search on User-generated Content. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 109–118, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2505515.2505680.

Katrin Braunschweig, Julian Eberius, Maik Thiele, and Wolfgang Lehner. The State of Open Data - Limits of Current Open Data Platforms. In *Proceedings of the 21st World Wide Web Conference, Web Science Track*, 2012.

Katrin Braunschweig, Maik Thiele, and Wolfgang Lehner. From Web Tables to Concepts: a Semantic Normalization Approach. *34th International Conference on Conceptual Modeling*, 2015.

Michael J Cafarella, Christopher Re, Dan Suciu, Oren Etzioni, and Michele Banko. Structured querying of Web text. In *3rd Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, California, USA*, 2007.

Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. WebTables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1:538–549, August 2008. http://dx.doi.org/10.1145/1453856.1453916.

Michael J. Cafarella, Jayant Madhavan, and Alon Halevy. Web-scale extraction of structured data. *SIGMOD Rec.*, 37(4):55–61, March 2009. http://doi.acm.org/10.1145/1519103.1519112.

Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the Set Covering Problem. *Annals of Operations Research*, 98(1-4):353–371, 2000. http://dx.doi.org/10.1023/A%3A1019225027893.

Jaime Carbonell and Jade Goldstein. The Use of MMR, Diversity-based Reranking for Reordering Documents and Producing Summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 335–336, New York, NY, USA, 1998. ACM. http://doi.acm.org/10.1145/290941.291025.

Vitor R. Carvalho, Matthew Lease, and Emine Yilmaz. Crowdsourcing for Search Evaluation. *SIGIR Forum*, 44(2):17–22, January 2011. http://doi.acm.org/10.1145/1924475.1924481.

Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and Efficient Fuzzy Match for Online Data Cleaning. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 313–324, New York, NY, USA, 2003. ACM. http://doi.acm.org/10.1145/872757.872796.

Surajit Chaudhuri, Raghu Ramakrishnan, and Gerhard Weikum. Integrating DB and IR technologies: What is the sound of one hand clapping. In *In CIDR*, pages 1–12, 2005.

Kuang Chen, Akshay Kannan, Jayant Madhavan, and Alon Halevy. Exploring Schema Repositories with Schemr. *SIGMOD Rec.*, 40(1):11–16, July 2011. http://doi.acm.org/10.1145/2007206.2007210.

Lei Chen, Dongwon Lee, and T. Milo. Data-driven crowdsourcing: Management, mining, and applications. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1527–1529, April 2015.

Zhe Chen and Michael Cafarella. Automatic Web Spreadsheet Data Extraction. In *Proceedings of the 3rd International Workshop on Semantic Search Over the Web*, SS@ '13, pages 1:1–1:8, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2509908.2509909.

Zhe Chen and Michael Cafarella. Integrating Spreadsheet Data via Accurate and Low-effort Extraction. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1126–1135, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2623330.2623617.

V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. http://dx.doi.org/10.1287/moor.4.3.233.

Joel Coffman and Alfred C. Weaver. A framework for evaluating database keyword search strategies. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 729–738, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1871437.1871531.

Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, and Caleb Welton. MAD skills: new analysis practices for big data. *Proc. VLDB Endow.*, 2:1481–1492, August 2009. http://portal.acm.org/citation.cfm?id=1687553.1687576.

Graham Cormode, Howard Karloff, and Anthony Wirth. Set Cover Algorithms for Very Large Datasets. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 479–488, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1871437.1871501.

Eric Crestan and Patrick Pantel. Web-scale table census and classification. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 545–554. ACM, 2011.

Jácome Cunha, João Saraiva, and Joost Visser. From Spreadsheets to Relational Databases and Back. In *Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM '09, pages 179–188, New York, NY, USA, 2009. ACM. http://doi.acm.org/10.1145/1480945.1480972.

Edward Curry, Andre Freitas, and Sean O'Riáin. The Role of Community-Driven Data Curation for Enterprises. In David Wood, editor, *Linking Enterprise Data*, pages 25–47. Springer US, 2010. http://dx.doi.org/10.1007/978-1-4419-7665-9_2.

Bhavana Bharat Dalvi, William W. Cohen, and Jamie Callan. WebSets: Extracting Sets of Entities from the Web Using Unsupervised Information Extraction. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 243–252, New York, NY, USA, 2012. ACM. http://doi.acm.org/10.1145/2124295.2124327.

Nilesh Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–12. ACM, 2007.

Nilesh Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proc. VLDB Endow.*, 5(7):680–691, March 2012. http://dl.acm.org/citation.cfm?id=2180912.2180920.

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, New York, NY, USA, 2012. ACM. http://doi.acm.org/10.1145/2213836.2213962.

Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.

Thomas H Davenport and DJ Patil. Data scientist: the sexiest job of the 21st century. *Harvard business review*, 90(10):70—6, 128, October 2012. http://europepmc.org/abstract/MED/23074866.

Elena Demidova, Peter Fankhauser, Xuan Zhou, and Wolfgang Nejdl. DivQ: Diversification for Keyword Search over Structured Databases. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 331–338, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1835449.1835506.

Gail W. DePuy, Reinaldo J. Moraga, and Gary E. Whitehouse. Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E: Logistics and Transportation Review*, 41(2):115 – 130, 2005. http://www.sciencedirect.com/science/article/pii/S1366554504000146.

Amol Deshpande, Zachary Ives, and Vijayshankar Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

Giusy Di Lorenzo, Hakim Hacid, Hye-young Paik, and Boualem Benatallah. Data Integration in Mashups. *SIGMOD Rec.*, 38(1):59–66, June 2009. http://doi.acm.org/10.1145/1558334.1558343.

Irit Dinur and David Steurer. Analytical Approach to Parallel Repetition. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 624–633, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2591796.2591884.

Paul Dixon. Basics of oracle text retrieval. *IEEE Data Eng. Bull.*, 24(4):11–14, 2001.

Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing Systems on the World-Wide Web. *Commun. ACM*, 54(4):86–96, April 2011. http://doi.acm.org/10.1145/1924421.1924442.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2623330.2623623.

Xin Luna Dong and Divesh Srivastava. Big data integration. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1245–1248. IEEE, 2013.

Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: selecting sources wisely for integration. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 37–48. VLDB Endowment, 2013. http://dl.acm.org/citation.cfm?id=2448936.2448938.

Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From Data Fusion to Knowledge Fusion. *PVLDB*, 7(10):881–892, 2014. http://www.vldb.org/pvldb/vol7/p881-dong.pdf.

Marina Drosou and Evaggelia Pitoura. Search Result Diversification. *SIGMOD Rec.*, 39(1): 41–47, September 2010. http://doi.acm.org/10.1145/1860702.1860709.

Marina Drosou and Evaggelia Pitoura. DisC Diversity: Result Diversification Based on Dissimilarity and Coverage. *Proc. VLDB Endow.*, 6(1):13–24, November 2012. http://dl.acm.org/citation.cfm?id=2428536.2428538.

Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. DrillBeyond: Enabling Business Analysts to Explore the Web of Open Data. *PVLDB*, 2012.

Julian Eberius, Patrick Damme, Katrin Braunschweig, Maik Thiele, and Wolfgang Lehner. Publish-time Data Integration for Open Data Platforms. In *2nd International Workshop on Open Data*, 2013.

Julian Eberius, Christoper Werner, Maik Thiele, Katrin Braunschweig, Lars Dannecker, and Wolfgang Lehner. DeExcelerator: a framework for extracting relational data from partially structured documents. In *22nd ACM International Conference on Information and Knowledge Management, CIKM*, 2013. http://doi.acm.org/10.1145/2505515.2508210.

Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. Building the Dresden Web Table Corpus: A Classification Approach. In *2nd IEEE/ACM International Symposium on Big Data Computing, BDC*, 2015.

Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. Top-k Entity Augmentation using Consistent Set Covering. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management, SSDBM*, 2015. http://doi.acm.org/10.1145/2791347.2791353.

Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. DrillBeyond: processing multi-result open world SQL queries. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management, SSDBM*, 2015. http://doi.acm.org/10.1145/2791347.2791370.

Ahmed El-Roby and Ashraf Aboulnaga. ALEX: Automatic Link Exploration in Linked Data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1839–1853, New York, NY, USA, 2015. ACM. http://doi.acm.org/10.1145/2723372.2749428.

Stefan Endrullis, Andreas Thor, and Erhard Rahm. Entity search strategies for mashup applications. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 66–77. IEEE, 2012.

Stefan Endrullis, Andreas Thor, and Erhard Rahm. Wetsuit: an efficient mashup tool for searching and fusing web entities. *Proceedings of the VLDB Endowment*, 5(12):1970–1973, 2012.

Lorena Etcheverry and AlejandroA. Vaisman. Enhancing OLAP Analysis with Web Cubes. In Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 469–483. Springer Berlin Heidelberg, 2012. http://dx.doi.org/10.1007/978-3-642-30284-8_38.

Ju Fan, Meiyu Lu, Beng Chin Ooi, Wang-Chiew Tan, and Meihui Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 976–987, 2014. http://dx.doi.org/10.1109/ICDE.2014.6816716.

Uriel Feige. A Threshold of Ln N for Approximating Set Cover. *J. ACM*, 45(4):634–652, July 1998. http://doi.acm.org/10.1145/285055.285059.

Ivan P. Fellegi and Alan B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969. http://dx.doi.org/10.2307/2286061.

Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

David A Ferrucci, Anthony Levas, Sugato Bagchi, David Gondek, and Erik T Mueller. Watson: beyond jeopardy! *Artif. Intell.*, 199:93–105, 2013.

Sheldon Finkelstein. Common expression analysis in database applications. In *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, pages 235–245. ACM, 1982.

Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34:27–33, December 2005. http://doi.acm.org/10.1145/1107499.1107502.

Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/1989323.1989331.

Fred Glover. Tabu search-part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: Hands-off Crowdsourcing for Entity Matching. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD

'14, pages 601–612, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2588555.2588576.

L. Golab, F. Korn, Feng Li, B. Saha, and D. Srivastava. Size-Constrained Weighted Set Cover. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 879–890, April 2015.

Sreenivas Gollapudi and Aneesh Sharma. An Axiomatic Approach for Result Diversification. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 381–390, New York, NY, USA, 2009. ACM. http://doi.acm.org/10.1145/1526709.1526761.

Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, 2010.

Georg Gottlob and Roberto Zicari. Closed World Databases Opened Through Null Values. In *VLDB*, volume 88, pages 50–61, 1988.

G. Graefe and K. Ward. Dynamic Query Evaluation Plans. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, pages 358–366, New York, NY, USA, 1989. ACM. http://doi.acm.org/10.1145/67544.66960.

Lars Grammel and Margaret-Anne Storey. A Survey of Mashup Development Environments. In *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 137–151. 2010.

Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. Scientific Data Management in the Coming Decade. *SIGMOD Rec.*, 34(4):34–41, December 2005. http://doi.acm.org/10.1145/1107499.1107503.

Ohad Greenshpan, Tova Milo, and Neoklis Polyzotis. Autocompletion for mashups. *Proc. VLDB Endow.*, 2:538–549, August 2009. http://portal.acm.org/citation.cfm?id=1687627.1687689.

Rahul Gupta and Sunita Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proc. VLDB Endow.*, 2(1):289–300, August 2009. http://dl.acm.org/citation.cfm?id=1687627.1687661.

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. VLDB Endowment, 2006. http://dl.acm.org/citation.cfm?id=1182635.1164130.

James R. Hamilton and Tapas K. Nayak. Microsoft SQL Server Full-Text Search. *IEEE Data Eng. Bull.*, 24(4):7–10, 2001. http://sites.computer.org/debull/A01DEC-CD.pdf.

Mahbub Hasan, Abdullah Mueen, Vassilis Tsotras, and Eamonn Keogh. Diversifying Query Results on Semi-structured Data. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 2099–2103, New York, NY, USA, 2012. ACM. http://doi.acm.org/10.1145/2396761.2398581.

Oktie Hassanzadeh, Ken Q. Pu, Soheil Hassas Yeganeh, Renée J. Miller, Lucian Popa, Mauricio A. Hernández, and Howard Ho. Discovering Linkage Points over Web Data. *PVLDB*, 6 (6):444–456, 2013. http://www.vldb.org/pvldb/vol6/p445-hassanzadeh.pdf.

Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 217–228, New York, NY, USA, 2003. ACM. http://doi.acm.org/10.1145/872757.872784.

Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. BLINKS: ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 305–316, New York, NY, USA, 2007. ACM. http://doi.acm.org/10.1145/1247480.1247516.

Cornelia Hedeler, Khalid Belhajjame, Norman W. Paton, Alvaro A.A. Fernandes, Suzanne M. Embury, Lu Mao, and Chenjuan Guo. Pay-as-you-go mapping selection in dataspaces. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 1279–1282, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/1989323.1989476.

Mauricio A. Hernández and Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, pages 127–138, New York, NY, USA, 1995. ACM. http://doi.acm.org/10.1145/223784.223807.

Dorit S. Hochbaum. Approximation Algorithms for NP-hard Problems. chapter Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems, pages 94–143. PWS Publishing Co., Boston, MA, USA, 1997. http://dl.acm.org/citation.cfm?id=241938.241941.

Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *In VLDB*, pages 670–681, 2002.

Qiang-Sheng Hua, Yuexuan Wang, Dongxiao Yu, and Francis C. M. Lau. Dynamic Programming Based Algorithms for Set Multicover and Multiset Multicover Problems. *Theor. Comput. Sci.*, 411(26-28):2467–2474, June 2010. http://dx.doi.org/10.1016/j.tcs.2010.02.016.

Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. MayBMS: a probabilistic database management system. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 1071–1074. ACM, 2009.

Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, October 2008. http://doi.acm.org/10.1145/1391729.1391730.

Tomasz Imielinski, Shamim Naqvi, and Kumar Vadaparty. Incomplete object—a data model for design and planning applications. *ACM SIGMOD Record*, 20(2):288–297, 1991.

Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Daniel S. Weld. An Adaptive Query Execution System for Data Integration. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 299–310, New York, NY, USA, 1999. ACM. http://doi.acm.org/10.1145/304182.304209.

Zachary G. Ives, Zhepeng Yan, Nan Zheng, Brian Litt, and Joost B. Wagenaar. Looking at Everything in Context. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015. http://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper10.pdf.

H. V. Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big Data and Its Technical Challenges. *Commun. ACM*, 57(7):86–94, July 2014. http://doi.acm.org/10.1145/2611567.

Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J. Haas. MCDB: A Monte Carlo Approach to Managing Uncertain Data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 687–700, New York, NY, USA, 2008. ACM. http://doi.acm.org/10.1145/1376616.1376686.

Matthew A. Jaro. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):pp. 414–420, 1989. http://www.jstor.org/stable/2289924.

Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 847–860, New York, NY, USA, 2008. ACM. http://doi.acm.org/10.1145/1376616.1376701.

Benedikt Kämpgen and Andreas Harth. Transforming Statistical Linked Data for Use in OLAP Systems. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 33–40, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/2063518.2063523.

Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI'11.*, pages 3363–3372, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/1978942.1979444.

R. M. Karp. Reducibility among Combinatorial Problems. *Complexity of Computer Computations*, 1972.

Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1 edition, March 1990. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0471735787.

Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. CrowdForge: Crowdsourcing Complex Work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 43–52, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/2047196.2047202.

Hanna Köpcke and Erhard Rahm. Frameworks for Entity Matching: A Comparison. *Data Knowl. Eng.*, 69(2):197–210, February 2010. http://dx.doi.org/10.1016/j.datak.2009.10.003.

Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of Entity Resolution Approaches on Real-world Match Problems. *Proc. VLDB Endow.*, 3(1-2):484–493, September 2010. http://dx.doi.org/10.14778/1920841.1920904.

Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record Linkage: Similarity Measures and Algorithms. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 802–803, New York, NY, USA, 2006. ACM. http://doi.acm.org/10.1145/1142473.1142599.

Alexandros Labrinidis and H. V. Jagadish. Challenges and Opportunities with Big Data. *Proc. VLDB Endow.*, 5(12):2032–2033, August 2012. http://dx.doi.org/10.14778/2367502.2367572.

Guanghui Lan, Gail W. DePuy, and Gary E. Whitehouse. An effective and simple heuristic for the set covering problem. *European Journal of Operational Research*, 176(3):1387 – 1403, 2007. http://www.sciencedirect.com/science/article/pii/S0377221705008313.

Doug Laney. 3D data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6:70, 2001.

Larissa R Lautert, Marcelo M Scheidt, and Carina F Dorneles. Web table taxonomy and formalization. *ACM SIGMOD Record*, 42(3):28–33, 2013.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, et al. DBpedia–A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. The Mannheim Search Join Engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2015.

Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 233–246, New York, NY, USA, 2002. ACM. http://doi.acm.org/10.1145/543613.543644.

Fei Li, Tianyin Pan, and Hosagrahar V. Jagadish. Schema-free SQL. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1051–1062, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2588555.2588571.

Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: is the problem solved? In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 97–108. VLDB Endowment, 2013. http://dl.acm.org/citation.cfm?id=2448936.2448943.

Yiping Li, Jianwen Chen, and Ling Feng. Dealing with Uncertainty: A Survey of Theories and Practices. *Knowledge and Data Engineering, IEEE Transactions on*, 25(11):2463–2482, Nov 2013.

Zhixu Li, Shuo Shang, Qing Xie, and Xiangliang Zhang. Cost Reduction for Web-Based Data Imputation. In SouravS. Bhowmick, CurtisE. Dyreson, ChristianS. Jensen, MongLi Lee, Agus Muliantara, and Bernhard Thalheim, editors, *Database Systems for Advanced Applications*, volume 8422 of *Lecture Notes in Computer Science*, pages 438–452. Springer International Publishing, 2014. http://dx.doi.org/10.1007/978-3-319-05813-9_29.

Zhixu Li, Mohamed A. Sharaf, Laurianne Sitbon, Shazia W. Sadiq, Marta Indulska, and Xiaofang Zhou. A web-based approach to data imputation. *World Wide Web*, 17(5):873–897, 2014. http://dx.doi.org/10.1007/s11280-013-0263-z.

Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, 3(1-2):1338–1347, September 2010. http://dl.acm.org/citation.cfm?id=1920841.1921005.

Jing Liu, Xin Dong, and Alon Y Halevy. Answering Structured Queries on Unstructured Data. In *WebDB*, volume 6, pages 25–30. Citeseer, 2006.

Vanessa Lopez, Spyros Kotoulas, MarcoLuca Sbodio, Martin Stephenson, Aris Gkoulalas-Divanis, and PólMac Aonghusa. QuerioCity: A Linked Data Platform for Urban Information Management. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, JosianeXavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2012*, volume 7650 of *Lecture Notes in Computer Science*, pages 148–163. Springer Berlin Heidelberg, 2012. http://dx.doi.org/10.1007/978-3-642-35173-0_10.

Alexander Löser, Fabian Hueske, and Volker Markl. Situational Business Intelligence. In Malu Castellanos, Umesh Dayal, and Timos Sellis, editors, *Business Intelligence for the Real-Time Enterprise*, volume 27 of *Lecture Notes in Business Information Processing*, pages 1–11. Springer Berlin Heidelberg, 2009. http://dx.doi.org/10.1007/978-3-642-03422-0_1.

Alexander Löser, Christoph Nagel, and Stephan Pieper. Augmenting Tables by Self-supervised Web Search. In Malu Castellanos, Umeshwar Dayal, and Volker Markl, editors, *Enabling Real-Time Business Intelligence*, volume 84 of *Lecture Notes in Business Information Processing*, pages 84–99. Springer Berlin Heidelberg, 2011. http://dx.doi.org/10.1007/978-3-642-22970-1_7.

Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Halevy. Corpus-Based Schema Matching. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 57–68, Washington, DC, USA, 2005. IEEE Computer Society. http://dx.doi.org/10.1109/ICDE.2005.39.

Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (luna Dong, David Ko, Cong Yu, Alon Halevy, and Google Inc. Web-scale Data Integration: You Can Only Afford to Pay As You Go. In *In Proc. of CIDR-07*, 2007.

Hatem A. Mahmoud and Ashraf Aboulnaga. Schema clustering and retrieval for multi-domain pay-as-you-go data integration systems. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 411–422, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1807167.1807213.

Albert Maier and David E. Simmen. DB2 Optimization in Support of Full Text Search. *IEEE Data Eng. Bull.*, 24(4):3–6, 2001. http://sites.computer.org/debull/A01DEC-CD.pdf.

James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H. Byers. Big data: The next frontier for innovation, competition, and productivity, May 2011. http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation.

Gary Marchionini. Exploratory Search: From Finding to Understanding. *Commun. ACM*, 49 (4):41–46, April 2006. http://doi.acm.org/10.1145/1121949.1121979.

Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. *Proc. VLDB Endow.*, 5:13–24, September 2011. http://dl.acm.org/citation.cfm?id=2047485.2047487.

Rafael Martí, Mauricio G.C. Resende, and Celso C. Ribeiro. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, 226(1):1 – 8, 2013.

Michael Martin, Martin Kaltenböck, Helmut Nagy, and Sören Auer. The Open Government Data Stakeholder Survey. In *Proceedings of the Open Knowledge Conference in 2011*. Open Knowledge Foundation, June 2011. http://okcon.org/2011/programme/the-open-government-data-stakeholder-survey.

Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, 2013.

Andrew McAfee and Erik Brynjolfsson. Big data: the management revolution. *Harvard business review*, 90(10):60—6, 68, 128, October 2012. http://europepmc.org/abstract/MED/23074865.

Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM. http://doi.acm.org/10.1145/347090.347123.

R. McCann, Warren Shen, and AnHai Doan. Matching Schemas in Online Communities: A Web 2.0 Approach. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 110 –119, april 2008.

Sean M. McNee, John Riedl, and Joseph A. Konstan. Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM. http://doi.acm.org/10.1145/1125451.1125659.

Hrushikesha Mohanty, Prachet Bhuyan, and Deepak Chenthati. *Big Data: A Primer*. Springer India, 2015.

John Morcos, Ziawasch Abedjan, Ihab Francis Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. DataXFormer: An Interactive Data Transformation Tool. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 883–888. ACM, 2015.

Kristi Morton, Magdalena Balazinska, Dan Grossman, and Jock Mackinlay. Support the Data Enthusiast: Challenges for Next-generation Data-analysis Systems. *Proc. VLDB Endow.*, 7(6): 453–456, February 2014. http://dx.doi.org/10.14778/2732279.2732282.

Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The Pipelined Set Cover Problem. In Thomas Eiter and Leonid Libkin, editors, *Database Theory - ICDT 2005*, volume 3363 of *Lecture Notes in Computer Science*, pages 83–98. Springer Berlin Heidelberg, 2005. http://dx.doi.org/10.1007/978-3-540-30570-5_6.

Daniel E. O'Leary. Embedding AI and Crowdsourcing in the Big Data Lake. *IEEE Intelligent Systems*, 29(5):70–73, 2014. http://dx.doi.org/10.1109/MIS.2014.82.

Aditya Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: it's okay to ask questions. *Proc. VLDB Endow.*, 4:267–278, February 2011. http://dl.acm.org/citation.cfm?id=1952376.1952377.

Rakesh Pimplikar and Sunita Sarawagi. Answering Table Queries on the Web using Column Keywords. In *Proc. of the 36th Int'l Conference on Very Large Databases (VLDB)*, 2012.

Jeffrey Pound, Ihab F. Ilyas, and Grant Weddell. Expressive and Flexible Access to Web-extracted Data: A Keyword-based Structured Query Language. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 423–434, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1807167.1807214.

Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001. http://dx.doi.org/10.1007/s007780100057.

Erhard Rahm and Hong Hai Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin*, 23:2000, 2000.

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, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. http://dl.acm.org/citation.cfm?id=645927.672045.

Raymond Reiter. *On closed world data bases*. Springer, 1978.

Arnon Rosenthal and Leonard J. Seligman. Data Integration in the Large: The Challenge of Reuse. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB

'94, pages 669–675, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. http://dl.acm.org/citation.cfm?id=645920.672973.

Mark A Roth, Herry F Korth, and Abraham Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems (TODS)*, 13(4):389–417, 1988.

Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and Extensible Algorithms for Multi Query Optimization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 249–260, New York, NY, USA, 2000. ACM. http://doi.acm.org/10.1145/342009.335419.

Satya S. Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, and Ahmed Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF, 01 2009. http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf.

Mariam Salloum, Xin Luna Dong, Divesh Srivastava, and Vassilis J. Tsotras. Online Ordering of Overlapping Data Sources. *Proc. VLDB Endow.*, 7(3):133–144, November 2013. http://dx.doi.org/10.14778/2732232.2732233.

Sunita Sarawagi and Soumen Chakrabarti. Open-domain Quantity Queries on Web Tables: Annotation, Response, and Consensus Models. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 711–720, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2623330.2623749.

Anish Das Sarma, Omar Benjelloun, Alon Halevy, and Jennifer Widom. Working models for uncertain data. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 7–7. IEEE, 2006.

Anish Das Sarma, Xin Luna Dong, and Alon Halevy. Data Integration with Dependent Sources. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 401–412, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/1951365.1951414.

Fabian Schomm, Florian Stahl, and Gottfried Vossen. Marketplaces for data: An initial survey. *ACM SIGMOD Record*, 42(1):15–26, 2013.

Timos K Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

N. Shadbolt, K. O'Hara, T. Berners-Lee, N. Gibbins, H. Glaser, W. Hall, and M.C. Schraefel. Linked Open Government Data: Lessons from Data.gov.uk. *Intelligent Systems, IEEE*, 27(3):16–24, May 2012.

Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.*, 22(3):183–236, September 1990. http://doi.acm.org/10.1145/96602.96604.

P. Shvaiko and J. Euzenat. Ontology Matching: State of the Art and Future Challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):158–176, Jan 2013.

Pavel Shvaiko and Jérôme Euzenat. A Survey of Schema-Based Matching Approaches. In Stefano Spaccapietra, editor, *Journal on Data Semantics IV*, volume 3730 of *Lecture Notes in Computer Science*, pages 146–171. Springer Berlin / Heidelberg, 2005. http://dx.doi.org/10.1007/11603412_5.

David E. Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh. Damia: data mashups for intranet applications. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, New York, NY, USA, 2008. ACM. http://doi.acm.org/10.1145/1376616.1376734.

Dimitrios-Emmanuel Spanos, Periklis Stavrou, and Nikolas Mitrou. Bringing Relational Databases into the Semantic Web: A Survey. *Semant. web*, 3(2):169–209, April 2012. http://dl.acm.org/citation.cfm?id=2590194.2590199.

Michael Stonebraker, George Beskales, Alexander Pagan, Daniel Bruckner, Mitch Cherniack, Shan Xu, Verisk Analytics, Ihab F. Ilyas, and Stan Zdonik. Data Curation at Scale: The Data Tamer System. In *CIDR 2013*, 2013.

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 '07, pages 697–706, New York, NY, USA, 2007. ACM. http://doi.acm.org/10.1145/1242572.1242667.

Mohammed Amin Tahraoui, Karen Pinel-Sauvagnat, Cyril Laitang, Mohand Boughanem, Hamamache Kheddouci, and Lei Ning. A survey on tree matching and XML retrieval. *Computer Science Review*, 8:1–23, 2013. http://dx.doi.org/10.1016/j.cosrev.2013.02.001.

Nguyen Thanh Tam, Nguyen Quoc Viet Hung, Matthias Weidlich, and Karl Aberer. Result selection and summarization for Web Table search. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 231–242, 2015. http://dx.doi.org/10.1109/ICDE.2015.7113287.

Andreas Thor, David Aumueller, and Erhard Rahm. Data integration support for mashups. In *Workshops at the Twenty-Second AAAI Conference on Artificial Intelligence*, 2007.

Thanh Tran, Yongtao Ma, and Gong Cheng. Pay-less Entity Consolidation: Exploiting Entity Search User Feedbacks for Pay-as-you-go Entity Data Integration. In *Proceedings of the 4th Annual ACM Web Science Conference*, WebSci '12, pages 317–325, New York, NY, USA, 2012. ACM. http://doi.acm.org/10.1145/2380718.2380759.

Marcos Antonio Vaz Salles, Jens-Peter Dittrich, Shant Kirakos Karakashian, Olivier René Girard, and Lukas Blunschi. iTrails: Pay-as-you-go Information Integration in Dataspaces. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 663–674. VLDB Endowment, 2007. http://dl.acm.org/citation.cfm?id=1325851.1325927.

R.R. Vemuganti. Applications of Set Covering, Set Packing and Set Partitioning Models: A Survey. In Ding-Zhu Du and PanosM. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 573–746. Springer US, 1999. http://dx.doi.org/10.1007/978-1-4613-0303-9_9.

Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proc. VLDB Endow.*, 4(9): 528–538, June 2011. http://dl.acm.org/citation.cfm?id=2002938.2002939.

Ruben Verborgh and Max De Wilde. *Using OpenRefine*. Packt Publishing, 1st edition, 2013.

Chi Wang, Kaushik Chakrabarti, Yeye He, Kris Ganjam, Zhimin Chen, and Philip A. Bernstein. Concept Expansion Using Web Tables. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1198–1208, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee. http://dl.acm.org/citation.cfm?id=2736277.2741644.

Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. CrowdER: Crowdsourcing Entity Resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012. http://dx.doi.org/10.14778/2350229.2350263.

Gerhard Weikum. DB&IR: both sides now. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 25–30. ACM, 2007.

Jennifer Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. Technical Report 2004-40, Stanford InfoLab, August 2004. http://ilpubs.stanford.edu:8090/658/.

Wayne L Winston and Jeffrey B Goldberg. *Operations research: applications and algorithms*, volume 3. Duxbury press Boston, 2004.

Ping Wu, Yannis Sismanis, and Berthold Reinwald. Towards keyword-driven analytical processing. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 617–628, New York, NY, USA, 2007. ACM. http://doi.acm.org/10.1145/1247480.1247549.

Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. Guided Data Repair. *Proc. VLDB Endow.*, 4(5):279–289, February 2011. http://dx.doi.org/10.14778/1952376.1952378.

Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. InfoGather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 97–108, New York, NY, USA, 2012. ACM. http://doi.acm.org/10.1145/2213836.2213848.

Xiaoxin Yin, Wenzhao Tan, and Chao Liu. FACTO: a fact lookup engine based on web tables. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 507–516, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/1963405.1963477.

Richard Zanibbi, Dorothea Blostein, and R. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. *Int. J. Doc. Anal. Recognit.*, 7(1):1–16, March 2004. http://dx.doi.org/10.1007/s10032-004-0120-9.

Meihui Zhang and Kaushik Chakrabarti. InfoGather+: semantic matching and annotation of numeric and time-varying attributes in web tables. In *Proceedings of the 2013 international conference on Management of data*, SIGMOD '13, pages 145–156, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2463676.2465276.

Jingren Zhou, Per-Ake Larson, Johann-Christoph Freytag, and Wolfgang Lehner. Efficient Exploitation of Similar Subexpressions for Query Processing. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 533–544, New York, NY, USA, 2007. ACM. http://doi.acm.org/10.1145/1247480.1247540.

Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving Recommendation Lists Through Topic Diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM. http://doi.acm.org/10.1145/1060745.1060754.

List of Figures

CONFIRMATION

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

Dresden, December 12, 2015