Automatic Table Extension with Open Data

Benedikt Kleppmann
Technological University Dublin

Automatic Table Extension with Open Data

Benedikt Kleppmann

A dissertation submitted in partial fulfilment of the requirements of

Dublin Institute of Technology for the degree of

M.Sc. in Computing (Advanced Software Development)

2018

Declaration

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Stream), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research.

Signed: [illegible]

Date: 04.01.2019

Abstract

With thousands of data sources available on the web as well as within organizations, data scientists increasingly spend more time searching for data than analysing it. To ease the task of finding and integrating relevant data for data mining projects, this dissertation presents two new methods for automatic table extension. Automatic table extension systems take over the task of data discovery and data integration by adding new columns with new information (new attributes) to any table. The data values in the new columns are extracted from a given corpus of tables.

Existing table extension algorithms rely on the user entering keywords that describe the new columns. This dissertation presents two new table extension algorithms that no longer require knowing the right keywords. The first method, unconstrained table extension, extends the submitted table with all attributes for which sufficient data was found in the corpus. The second method, correlation-based table extension, extends the query table with all attributes that correlate with a specific attribute of the submitted table.

The new table extension algorithms were thoroughly evaluated. As criteria high usability, high availability, wide applicability, fast execution time and correctness of the extended table were used. The results show that the new algorithms provide a viable alternative to the existing methods and an interesting avenue for further research.

Keywords: Data discovery, big data integration, table extension, holistic matching, web tables

Acknowledgments

I would like to express my sincere thanks to my supervisor Professor Sarah Jane Delany, for her great assistance and guidance throughout the process of this dissertation.

I would like to thank all my instructors at DIT – Brenden Tierney, Damian Gordon, Deirdre Lawless, Luca Longo, Ciaran Cawley, Sarah Jane Delany, Robert Ross, Brian Gillespie, John Gilligan, Pierpaolo Dondio and Yupeng Liu. Thank you for all the things I learned at DIT.

Thank you to Christian Bizer for his guidance and support as supervisor of the DS4DM research project, for which much of the work presented here was done.

Thank you to Edwin Yaqub, Philipp Schlunder, Fabian Temme and Ralf Klinkenberger for evaluating the new table extension system and providing invaluable feedback.

Thank you to my parents who had to put up with me while I was writing this dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In 2009, on Barack Obama's first day in office as US president, he announced his Open Government Strategy. Key to this Open Government strategy was the large-scale publishing of governmental data. This inspired many other countries to launch similar Open Data initiatives: The United Kingdom in 2009, Australia, Denmark, Spain and the EU in 2010 (Huijboom & Van den Broek, 2011).

These days, many national, regional and communal governments, as well as international organizations (such as the world bank or UN) have published significant parts of their data online. With the amount of data published on every platform ranging from 100 to 1 000 000 datasets (Braunschweig, Eberius, Thiele & Lehner, 2012).
In science a comparable open science movement has caused massive amounts of data from many disciplines to be made publicly accessible - from the diverse data in cognitive psychology and ecology to the massive datasets in genetic biology, particle physics and astronomy (Hoover, 2017).

As Yakout, Ganjam, Chakrabarti & Chaudhuri state in 2012, "there is enormous potential in combining and re-purposing open data.". Lehmberg et al. (2015) for example illustrates an example where a company is able to do more effective marketing by extending their customer table with additional country information extracted from open data. Halevy, Rajaraman & Ordille (2006) even go so far as to say “Today, data integration is a necessity.”.

In practice the use of open data in companies has however been very limited. This is because integrating external data is very time intensive – McCue famously stated in 2007, that “the general rule is that the data mining process is 80% preparation and 20% analysis.”. Working with open data brings additional challenges. Janssen, Charalabidis & Zuiderwijk identified in 2012 further reasons why accessing and integrating open data is specifically work intensive. They are: the difficulty to discover appropriate data, no means of searching the data on the portals; the lack of explanation of the data on data portals; and non-standardized data formats.

There is therefore a great value in systems that facilitate the process of finding and integrating the desired open data. The most promising of these systems are so-called table extension systems. They were conceived for the common use case that an analyst in an organisation already has a relational table with information about some entities and would like to add additional columns to this table to gain additional information about these entities. These table extension systems find and extract the data for populating the additional columns from a corpus of open data tables.

1.2 Problem Description

Currently there are only three systems that fulfil the task of automatic table extension. They are: Octopus (Cafarella, Halevy & Khoussainova, 2009), InfoGather (Yakout, Ganjam, Chakrabarti & Chaudhuri, 2012) and the Mannheim Search Join Engine (Lehmberg et al., 2015). The table extension operations of these systems work according to the same basic principle – you submit a to-be-extended table and some keywords and the system adds exactly one column to your table, the column described by the keywords.

These table extension systems are very dependent on the user knowing the right keywords for the columns he/she would like to add to their table. In practice however, many users do not know which columns the system is able to add to their table and what the best keywords are for finding these columns.

This begs the following question: can there be a table extension system that work comparably well while not relying on the user to input keywords?

The goal of this dissertation is to answer this question by developing and evaluating two new methods of tables extension that no longer require keywords:

- Unconstrained Table Extension
  This proposed solution no longer requires keywords, as it extends your table with all possible new attributes at once. It is easy for the user to afterwards remove unwanted columns, should the unconstrained table extension return too many columns.
- Correlation-based Table Extension
  This proposed solution works similarly to the unconstrained table extension, it however only adds columns to the table that correlate with a specified table column.

1.3 Research Aims and Objectives

In this dissertation the hypothesis is posed that these two new table extension algorithms – unconstrained table extension and correlation-based table extension – are a viable alternative to the existing table extension algorithms.

In section 3.1. of this dissertation, the following 5 key requirements for a viable table extension algorithm are identified: *High usability*, *High availability*, *Wide applicability*, *Fast execution time* and *Correctness*. The goal of this dissertation is to prove or disprove the viability of the new table extension algorithms by checking whether the new table extension algorithms fulfil these key requirements. Furthermore, the new table extension systems are required to have a comparable correctness to the existing table extension algorithms. (For the other table extension systems only the correctness had been evaluated).

In order to evaluate the two new table extension systems with respect to the 5 key requirements, a new table extension system was implemented. Users can interact with this new table extension system via a public-REST-API. They can create repositories of tables and use these repositories for running the table extensions.

1.4 Research Method

The five key requirements are evaluated in the following way:

High usability
An evaluation according to the methodology proposed by the famous usability researcher Jakob Nielsen (1994) was performed. Thereby, over a four-month period, regular on-site as well as remote evaluations were performed with four external evaluators. An evaluation was performed after every iteration of the table extension system. The feedback from these evaluations was used to improve the usability of the system until there were no more complaints and the system was shown to be efficient, learnable, memorable, hard-to-make-errors-with and satisfactory.

High availability
The table extension system was continuously running and being used by external users over a five month period. All outages that occurred during this period were recorded, along with the outage duration and the cause of the outages. From this data the commonly used metrics Operational Availability, Mean Time Between Failure (MTBF)

and Mean Time To Recover (MTTR) were calculated.

Wide applicability

The wide applicability is guaranteed by the fact that users can create their own repositories containing tables of an arbitrary domain. To verify that the new table extension algorithms is correct and performant for different domains, the evaluations of the correctness and of the execution times were performed on multiple repositories, and a range of different tables were extended for each evaluation.

Fast execution time

The execution times for a range of table extension operations were measured. To test the upper limit of the execution time, the evaluated table extension operations were performed with a very large repository containing 460 thousand tables. The execution times of the individual execution steps were also analysed.

Correctness

Two evaluations of the unconstrained table extension were performed and one of the correlation-based table extension. The evaluation methodology used for evaluating the new unconstrained table extension algorithm matches that of the existing table extension algorithms – multiple tables are extended and the precision and density of the new columns are calculated. For one of the evaluations of the unconstrained table extension, even exactly the same data was used as had been used for the evaluation of one of the existing table extension algorithms, thereby allowing for a direct comparison. The correlation-based table extension consists of the unconstrained table extension, with an additional step called correlation-based filtering. The quality of the new columns has already been evaluated with the evaluation of unconstrained table extension. The remaining step – the correlation-based filtering – is evaluated by comparing the columns that were found to be correlating with the truly correlating columns and calculating precision, recall- and F1- scores.

The evaluation methodologies are described in detail in section 3.3. of this dissertation.

1.5 Scope and Limitations

The research presented in this dissertation was performed as part of a research project at the university of Mannheim. The author of this dissertation worked from April 2017 to July 2018 for the Data Search for Data Mining (DS4DM) research project[1]. This research project had a wider scope than this dissertation – only the unconstrained- and correlation-based table extension are presented in this dissertation, which were implemented and evaluated between March and July 2018. A paper[2] presenting these two table extension algorithms was published in the proceedings of LWDA in August 2018[3].

1.5.1 Contributions by the author

The research project was a collaborative research project between the University of Mannheim and the company RapidMiner GmbH.

The author of this dissertation developed the table extension algorithms presented in this dissertation. He then made these table extension algorithms publicly usable by creating a web service with a REST API for these algorithms. The author also created all the additional, supporting API calls (see Appendix B), the documentation (see Appendix D) and this thesis. The evaluations presented in this dissertation were also performed entirely by the author.

The company RapidMiner GmbH then used the web service with these table extension functions to develop an extension to their popular data mining tool RapidMiner Studio. This RapidMiner extension is a visual front-end which calls the table-extension

[1]`https://dws.informatik.uni-mannheim.de/en/projects/ds4dm-data-search-for-data-mining/`

[2]`https://dws.informatik.uni-mannheim.de/fileadmin/lehrstuehle/ki/pub/KleppmannBizer-DensityAndCorrelationBasedTableExtension-LWDA2018.pdf`

[3]`http://ceur-ws.org/Vol-2191/`

functions of the web service. It allows users to interact with the web service in a more intuitive fashion and from within a powerful data mining environment. More details on the collaboration with RapidMiner can be found in Appendix D.
RapidMiner also provided great assistance by continuously giving feedback on the usability of the web service and the table extension algorithms (see section 4.3.1).

The author also received very helpful advice from the professors supervising this dissertation and supervising the DS4DM research project - Prof. Sarah Jane Delany and Prof. Christian Bizer. This advice concerned the design of the table extension algorithms, the evaluations, the thesis, the web service and the documentation.

It is also important to note, that the table extension algorithms were developed with the Java Library 'Winte.r' (more info in section 3.2.1). From this library the following two functions are used instead of being re-implemented:

- subject column detection
  This function is used for detecting the table column with entity names (see section 3.2.4).
- hungarian algorithm
  This combinatorial optimisation algorithm is used to identify the sets of instance/schema matches that maximize an overall similarity score (see section 3.2.6).

1.5.2 Scope and limitations of the dissertation

During the research project, the new table extension algorithms were tested in a range of different scenarios. For the main evaluations two different repositories of tables were used: one big repository containing 460 513 of tables which had been mined from Wikipedia (more info in section 4.3.5.1.1) and a repository with 31 product-data tables (see 4.3.5.1.2).
The table extension operation is not limited to any particular application. For the

evaluations it was used to add additional columns/attributes to a range of different tables, including tables containing information about mountains, airlines, fells, Irish political parties, airports, currencies, lakes, animals, sky scrapers, Roman emperors, hospitals, journals, museums, books, companies, countries, drugs, films, songs, soccer players, phones, TVs and headphones.

The research project was a success. RapidMiner is planning to move the table extension functionality to the core functionality of RapidMiner Studio in their next release.

1.6 Organization of Dissertation

This dissertation begins with the literature review in chapter 2. The literature review gives a historic overview of the research in data integration, starting with the development of manual data-integration methods in the mid-1990s and continuing with the development of big-data-integration methods from the mid-2000s onwards. Next, there is an in-depth analysis of the three existing table extension algorithms: Octopus, InfoGather and the Mannheim Search Join Engine.

Based on the learnings from the literature review, the two new table extension algorithms and their evaluation methodologies are designed in chapter 3. The design of the two new table extension algorithms is guided by the five key requirements which the system must fulfil: *High usability*, *High availability*, *Wide applicability*, *Fast execution time* and *Correctness*. To evaluate the new table extension algorithms, separate evaluations for each of these requirements are designed.

In chapter 4, the new table extension system is presented which was developed to evaluate the table extension algorithms. Next, the performed evaluations and the obtained evaluation results are described.

In chapter 5, the evaluation results are analysed and compared to existing work. The detailed analyses answer questions such as: ‘When do the errors occur?’, ‘What causes the errors?’, ‘How do the individual steps of the algorithm affect the performance?’ or, ‘How could the algorithms be improved?’.

Finally, chapter 6 concludes the dissertation by giving an overview of the key methods and findings of this dissertation, its impact, and ideas for further research in this field.

Chapter 2

Literature Review

2.1 Introduction

This chapter gives a historic overview over the research in the field of data integration. It presents the motivations behind the developments and shows the popular applications and uses for the developed algorithms and techniques. The historic overview covers both the early developments that were focussed around smaller amounts of data and the more recent developments in big data integration.
An in-depth analysis of the three existing table extension systems follows. Their design and algorithms are discussed in detail, as is the design of their evaluations. The analysis of the existing table extension algorithms and their evaluations provides a solid basis for designing the new table extension algorithms in the next chapter.

2.2 Historic Overview of Data Integration

2.2.1 The era of small data

Data Integration as a research discipline was established in the mid-1990s. The goal of data integration at the time was to build systems to interface multiple databases within a company (Golshan, Halevy, Mihaila & Tan, 2017).
Other areas where data integration was (and still is) of crucial importance include:

large-scale scientific projects, where multiple researchers independently produce datasets, and cooperation among government agencies with different data sources (Halevy, Rajaraman & Ordille, 2006).

All approaches at the time involved a mediated schema (or 'global schema'). This mediated schema is a unified, holistic view, combining the data from all the sources. Mappings between sources and the mediated schema define how the sources relate to the mediated schema.
When a user queries the mediated schema, the data integration system uses the mappings to reformulate the query as a combination of queries to the different sources, retrieves the necessary data from these sources, and combines them.

2.2.1.1 Data integration approaches

According to Xu & Embley (2004), there exist two general approaches to data integration:

Global-as-view
Here the mediated schema (global schema) is created as a view of the individual sources. The mappings define how the global schema is built from the local sources. More specifically, the mappings define what transformations the local data have to undergo to be loaded and joined into the global schema.

Local-as-view
Here the mappings specify the local data sources as views of the mediated schema. This has several advantages:

- It is much easier to integrate new sources, as you only need to know how the new source relates to the mediated schema (Global-as-view requires knowledge of all data sources).
- Local-as-view allows more precise resource descriptions. In particular, restric-

tions of the data sources can be expressed.

Local-as view however requires more complex query reformulation – the views of the local sources must be inverted to correctly fetch, transform and join the data into the mediated schema. A detailed comparison of Local-as-view and Global-as-view can be found in (Lenzerini, 2002, Levy, 2000).

2.2.1.2 Mapping languages

The formalization of the data integration processes and the need for complex query reformulation lead to the development of various languages for creating mappings and transformations. The most prominent are:

Source Description Languages
These languages are derived from Description Logic – from the field of Knowledge Representations in AI (Catarci & Lenzerini, 1993). Here data sources are represented declaratively. It offers a flexible mechanism for representing schemas and for semantic query optimization (Calvanese, De Giacomo, Lembo, Lenzerini & Rosati, 2013) and allows for AI planning and adaptive planning to be used in the query optimization.

GLAV
GLAV stands for 'Global-Local-as-view' (Friedman, Levy & Millstein, 1999). It is a mediation language between datasets which can be used for both Local-as-view and Global-as-view architectures (hence the name). Instead of being based on description logic, it is based on views, which makes it easier to use for non-experts. GLAV has found commercial application in data warehouse solutions – see section 2.3.1.3.

Model algebra
Model algebra was created to provide a mathematical foundation for data transformations. Here, complex operations on data sources are described as a sequence of basic operators. The maths helps to simplify transformations and to invert and compose them (Fagin, Kolaitis, Popa & Tan, 2004, Madhavan & Halevy, 2003).

The choice of mapping language comes down to a question of expressiveness vs. tractability of query execution. Examples of the trade-offs are: dealing with incomplete data sources (Abiteboul & Duschka, 1998), binding-pattern restrictions (Florescu, Levy, Manolescu & Suciu, 1999), and using more complex data sources (Levy, Rajaraman & Ullman, 1996).

2.2.1.3 Commercial applications

Since the late 1990s, companies have had a great need to integrate data from various sources inside and outside the company – they still do: sales of data integration tools exceeded $3.3 billion in 2009 (Brodie, 2010). There are two commercial technologies that provide this capability:

Enterprise Information Integration (EII)

These software solutions were derived directly from the data integration research mentioned above. An important feature was that the data does not have to be migrated from the sources to a centralized data warehouse, instead when the mediated schema is queried, the necessary data is retrieved from the sources dynamically. The advantage of this over the data warehouse solutions is that changes to the sources are captured in real time (Halevy et al., 2005).

Data warehouses

With data warehouses, the data is materialized in the mediated schema. ETL processes extract the data from the sources, transform them and load them into the centralized data warehouse. Advantages of these data warehouses are:

- the faster execution time, due to the data being stored locally
- increased reliability, due to not relying on all sources to be permanently online

Especially as data warehouses started to offer more real-time data-updating capabilities, they began to dominate the market and have done so since.

All these data integration projects rely on manually creating mappings for the data sources. This is manageable for traditional data integration applications that typically involve no more than 20-30 data sources (Levy, Rajaraman & Ordille, 1996). However, this can no longer be done for big data.

2.2.2 The era of big data

The age of big data is upon us. Studies from Microsoft and Google report that there are hundreds of millions of high-quality datasets on the web (Balakrishnan et al., 2015, Chakrabarti, Chaudhuri, Chen, Ganjam & He, 2016).

Many companies are trying to extract value from this data by integrating it. For example, search engine results nowadays routinely include data. "As of 2006, the large search companies are performing several efforts to integrate data from the multitude of data sources available on the web" (Halevy, Rajaraman & Ordille, 2006). Other examples are question answering systems – such as Watson or chatbots. Question answering systems integrate huge amounts of openly available data to find answers to questions asked by users (Dong & Srivastava, 2013).

Within companies the amount of data also has skyrocketed. An extreme example is Google LLC, which reported to use 26 billion datasets in its search system (Halevy et al., 2016). Many other companies are collecting huge amounts of data. For them "addressing the big data integration challenge is critical to realizing the promise of Big Data" (Dong & Srivastava, 2013).

2.2.2.1 Table search

With these vast amounts of data, finding a dataset with the desired data becomes a big challenge. Table search engines work very similar to conventional search engines – they use a search index. However, instead of returning websites they return tables.

An example for such a system is Google Tables[1]. "In fact, search for datasets is becoming so important that each of the cloud service providers are now starting to support search over datasets." (Golshan, Halevy Mihaila & Tan, 2017)

Types of table search are:

- Keyword-based table search

  This uses keywords as search term

- Table-based table search

  This uses a table as search term and tries to find similar tables. Users see this type of data search less frequently, it is however often used inside data integration applications. The table extension system presented in this dissertation uses such a table search.

2.2.2.2 Automated mappings

For small numbers of datasets, analysts can manually create mappings between these datasets and the mediated schema. This quickly becomes unfeasible as the number of datasets increases. Therefore, algorithms were developed that find mappings automatically.

The types of mappings that these algorithms can find are:

- Instance correspondences

  An instance correspondence identifies two rows from two different tables to contain information about the same entity. E.g. it says that the row with data on 'Afghanistan' in the countries table corresponds to the row with data on 'AFG' in the nations table.

- Schema correspondences

  A schema correspondence identifies two columns from two different tables to

[1]`https://research.google.com/tables`

contain the same attribute. E.g. it says that the 'Population'-column of the countries-table corresponds to the 'number of inhabitants'-columns of the nations table.

- Schema mappings
  Sometimes relations between table columns are more complex than a one-to-one correspondence. E.g. the two columns 'First Name' and 'Last Name' from the one table might correspond to the column 'Name' in the other table, with the first name additionally being abbreviated to only the first letter.

Finding such complex schema mappings is hard. It is only possible, when the two databases that are being mapped contain exactly the same information (Fagin, Kolaitis, Popa & Tan, 2010, Alexe, Ten Cate, Kolaitis & Tan, 2011). As this is not the case for the automated table extension system presented in this dissertation, it only employs instance- and schema- correspondences.

2.2.2.3 Interlinked network instead of mediated schema

Another issue that big data integration systems face is the creation of a global, mediated schema. It might not even be possible: Freebase, the broadest schema, has been shown to cover only a small fraction of the attributes used in HTML tables (Gupta, Halevy, Wang, Whang & Wu, 2014).

The solution to this is to interlink the tables to a network: The nodes of the network are the tables and the edges are instance- and schema- correspondences between the tables. The goal is to create well-connected neighbourhoods of similar tables. So-called semantic paths can then be used to create additional correspondences.

Pay-as-you-go systems are a common example of systems that employ such networks of tables (Ives et al., 2008).

2.2.2.4 Methods for efficient data processing

To deal with the vast amounts of data, researchers have also developed methods for efficiently processing the data. These are:

Blocking

Schema matching is the process of looking for schema correspondences. Comparing every column with every other column in the datasets is computationally very expensive – it has a complexity of $O(n\hat{2})$. Therefore, blocking was developed. This is the process of using a computationally less expensive algorithm to find candidate schema correspondences and then only doing the rigorous schema correspondence finding on these candidate correspondences. These candidate schema correspondences are usually found by clustering similar columns. The clusters of potentially corresponding columns are sometimes called blocks, hence the name. The table extension system presented in this dissertation also uses blocking – see section 3.2.3.

Parallel processing

Parallel processing has also been employed to manage the vast amounts of data. For instance, the matching systems presented in (Dong & Srivastava, 2013) and (Elsayed, Lin & Oard, 2008) are implemented with map-reduce.

2.2.2.5 Commercial applications

Data lakes

Data lakes (in research known as 'data spaces') have become popular in recent years. For big companies they have an interesting proposition: the generators of data within the company don't have to deal with the users of the data, they directly save their data to the data lake in whatever format they like. It's up to the users of data to find the right data for their application and integrate it. The data lake will usually have functionality to help with this, such as data search, storage of table metadata and monitoring of datasets for changes.

2.3 Table Extension Systems

In the field of big data integration, table extension is the task of adding a new column to a relational table with additional information about the entities of the table. The data for populating this new attribute is extracted from a big repository of tables covering a whole range of topics.

There are many cases where getting additional information on some entities is very useful. Imagine for example an analyst in a company that has the task of helping customer relations by clustering the company's cooperate customers. The existing data however does not contain information about the customers' company sizes. Having this information would help immensely for estimating the potential engagement with this company.

Without an automatic table extension system, the analyst would have to spend many days searching the web for datasets with company sizes, transforming them into a useful format and writing some script to find instance correspondences and merge the data with the existing table. An automatic table extension system does this work in 2-10 seconds.

Although a lot of research has been done on table search and adding columns to tables, there are only three systems that provide table extension operations: Octopus (Cafarella, Halevy & Khoussainova, 2009), InfoGather (Yakout, Ganjam, Chakrabarti & Chaudhuri, 2012) and the Mannheim Search Join Engine (Lehmberg et al., 2015).

In the following sections the design and evaluation of these three systems is explained and contrasted.

2.3.1 Repository of tables

As previously mentioned, the table extension algorithms use a repository of open data tables from which the values for the additional column are extracted.

Octopus

The Octopus system uses a corpus of 200 million relational tables which were extracted from websites by a google web crawl. To allow the quick finding of a specific column, a search index was created with a record for each column-header in the 200 million tables.

InfoGather

InfoGather uses a corpus of 573 million tables that were extracted from websites by a Bing search engine crawl in July 2011. To speed up the table extension algorithm, two search indexes were created:

- one for searching for rows – there is a document for each row of every table
- one for searching for columns – there is a document for each column

Also, the similarity of any two tables in the repository is calculated and saved in an interlinked network.

MSJE

The MSJE can work with many different repositories of tables. It was evaluated with both corpuses of relational tables and knowledge bases. Tables that are uploaded to a repository are pre-processed. This pre-processing includes:

- Removing too small tables (smaller than 3x5)
- String normalisation (stop words and brackets removal, converting to lower case)
- Creating two search indexes. As with InfoGather, there is one search index for the rows and one for the columns. The search index for the rows is however different, instead of the records containing all the rows' values, they contain

only the entity names.

A range of different heuristics is used to identify the columns with the entity names. The heuristics include the number of distinct values in each column, whether the column values are strings or other data types, whether the words 'name' or 'title' appear in the column header, and even the position of the column. These algorithms for identifying the columns with the entity names are also known as *subject-column-detection* algorithms (Lehmberg, Brinkmann & Bizer, 2017).

2.3.2 The table extension algorithm

For all three table extension systems, the table extension algorithm receives from the user a table and some keywords describing the new column that should be added to the table. The table extension algorithms then go through the following steps to create the extended table:

Octopus

Here the submitted table may consist of only one column: the column with the entity names ('subject column'). The table extension steps are the following:

1. Search in the repository for tables with a column-header resembling the submitted keywords (by using the index of column headers – see pre-processing)
2. For the tables found this way, check if there is a column that has many values in common with the subject column of the submitted table. If none is found, remove this table from the pre-selection.
3. Cluster the tables in the pre-selection according to table-similarity. Choose the cluster where the most distinct entity-names equal an entity-name from the submitted table.
4. Populate the new column: If there are several values in the cluster for the same entity, choose the value from the column whose header had the biggest similarity

to the keywords in step 1.

Infogather

Here the submitted table can already have multiple columns, the subject column however must be the left-most column. The table extension steps are:

1. Search for tables with at least one row that is similar to any of the rows of the submitted table

2. For the tables found in the first search, find additional potentially matching tables that are similar to the previously found tables, using the linked network (see pre-processing).

3. For all potentially matching tables found above, check if there is a column that is similar to the keywords. If not, remove the table from the list of matching tables.

4. For each of the matching tables, calculate the table-similarity to the submitted table (to be used later).

5. Cluster the entities of all matching tables with an agglomerative hierarchical clustering algorithm (Rokach & Maimon, 2005), whereby the jaccard string similarity (Levandowsky Winter, 1971) between the entity-names is used as distance metric between the entities.

6. The cluster that corresponds to a certain entity of the submitted table is the cluster whose entity-names have the greatest overall string similarity to that entity-name.

7. For every cluster choose the new attribute value that came from the table with the biggest similarity to the submitted table (see step 4).

MSJE

For the Mannheim Search Join Engine (MSJE) the submitted tables may have any form, as a subject column detection algorithm is used to identify the subject column automatically. The table extension steps are:

1. Apply the same string normalisation to the values of the submitted table as was applied to the tables in the repository
2. Search for tables with a column similar to the subject column of the submitted table
3. Keep those tables where the keyword appears in one of the attribute headers
4. Use instance matching with data-type-specific similarity measures to calculate a similarity score between the entities of the found tables. Use these similarities to do clustering.
5. For every entity of the submitted table, choose the most similar cluster and choose the value of that cluster that is in the middle i.e. the value that is most similar to all the other values of the cluster.

2.3.3 Evaluation of the algorithms

2.3.3.1 Evaluation design

For the evaluation of each of the table extensions algorithms, 6 – 7 tables were taken from trustworthy sites such as Wikipedia, freebase or the IMDB-database (for films). The list of tables used for these evaluations is presented in Appendix A.
From each of these tables a column was removed, and the table extension system was given the task of reconstructing the deleted column – using as keywords the header name of the deleted column. By comparing the reconstructed column with the original column, the following two metrics were calculated for every table extended this way:

$$precision = \frac{TP}{TP + FP} = \frac{\#values\_correctly\_populated}{\#values\_populated} \tag{2.1}$$

$$density = \frac{\#populated\_fields}{\#fields\_to\_be\_populated} \tag{2.2}$$

(Whereby TP stands for true positive and FP stands for false positive.)

2.3.3.2 Evaluation results

The evaluation results for the three different table extension systems are shown in Table 2.1. For the evaluation of Octopus and MSJE seven different Tables were used; for the evaluation of InfoGather six tables were used. The used tables cover a range of topics, such as Cities, Films, Cameras, Countries and Songs. A detailed description of them can be found in Appendix A.

	Octopus	**InfoGather**	**MSJE**
precision	39%	78%	79%
density	38%	96%	97%

Table 2.1: Evaluation scores for the three table extension systems.

The precision and density scores here are the average of the precision and density scores for each evaluation table.

2.3.3.3 Discussion

It is clearly visible that Octopus performs worse than the other two systems. Reasons for this are:

- Octopus initially searches for tables that have a column header resembling the keywords, MSJE and InfoGather on the other hand initially search for tables that have the same entities as the submitted table. Keywords are more error-prone than entity-names, through Octopus' initial restrictions on tables with the right keywords in a header, many tables containing the right data are missed.

- Octopus requires entity-name matches to be exact string matches. This is too restrictive – many correspondences are lost this way.

- Octopus assumes that the matching tables might belong to various topics and only one topic is the topic of the submitted table. In step 3 it therefore tries to cluster the found tables according to topics and only keeps the tables of one topic. This seems to be a false assumption.

There are slight differences between InfoGather and MSJE. They are due to the fact that InfoGather finds additional, indirectly matching tables (with the interlinked network), MSJE however applies string normalisation to all tables and uses more sophisticated fuzzy matching techniques with e.g. data-type-specific similarities.

2.4 Conclusion

This chapter gives an overview of the developments and applications in the field of data integration. It shows the techniques that were developed since the mid-1990s and how new algorithms and techniques had to be developed to tackle the new challenges of big data integration.
The second section describes how these big data integration methodologies are applied to automatic table extension and how the existing table extension algorithms were evaluated.
The evaluations of the existing table extension algorithms show very promising results. They indicate that these algorithms have the potential for mass market adoption and are a promising field of research. In the next chapter the knowledge gained from this chapter will be applied to develop new table extension algorithms.

Chapter 3

Design and Methodology

3.1 Introduction

The last chapter gave a general overview of data integration systems and an analysis of the existing table extension systems. All existing table extension algorithms and systems are so-called keyword-based table extension systems. This means that the user provides a table and some keywords which describe the column that should be added. The table extension algorithm then adds the described column to the table. The user faces some key challenges when working with such a system:

1. The user has no way of knowing which columns the system can add to the table and might miss some columns that he/she would have found useful.

2. The performance of the table extension system is very sensitive to the actual keywords used for describing the new column. The user has no way of knowing which exact keywords will produce the best result.

Due to these limitations of the existing approaches, this dissertation presents two new algorithms for table extension which do not require keywords: the unconstrained table extension and the correlation-based table extension.

Instead of adding only one column to the provided table, the *unconstrained table extension* system adds as many columns as possible to the table, without requiring any keywords at all. Should the user require fewer than the provided columns, he or she

can easily delete them with any data analysis/transformation tool.
The *correlation-based table extension* is very similar to the unconstrained table extension, it also extends the submitted table with multiple columns, here however only columns will be added that correlate with a user-specified column from the submitted table – the ‘correlation attribute’.

In many scenarios, using the unconstrained table extension will be a better option for a data analyst, than using the correlation-based table extension. The correlation-based table extension however saves the analyst work in some scenarios where the analyst would otherwise have to manually filter out non-correlating columns:

- In the case that the analyst wants to use the table for machine learning, then only those additional columns/features will improve the model that have a correlation with the Y-variable/dependent variable.
- In the case that the analyst wants to determine columns/variables that influence a specific variable of interest (e.g. sales of some product), then only those variables/columns are possible influencers that correlate with the variable of interest.

The task of this dissertation is to investigate whether these new table extension algorithms provide a viable alternative to the existing table extension algorithms.

For the new table extension algorithms to be a viable alternative to the existing ones, they have to provide a truly useful service. From here we derive the following five key requirements for the table extension system:

1. *High usability* – the system should not require extensive setup, be intuitive to operate, etc.
2. *High availability* – the system should be usable at any time, i.e. not crash easily.
3. *Wide applicability* – it should be possible to extend tables from a wide range of domains.

4. *Fast execution time* – the table extension process should take no longer than 12 seconds.

5. *Correctness* – the columns that were added to the table have to be well populated and correctly populated.

The first section in this chapter will illustrate how the new table extension system is designed to achieve these five requirements. The second section of this chapter describes how the different evaluations of the new table extension system were designed in order to validate the five key requirements stated above.

3.2 Design of the new Table Extension System

3.2.1 General design choices

Java was chosen as programming language for the table extension. Java is sufficiently performant for processing the large amounts of data in the required time. As Java is the most popular programming language it is well documented, has a large community and many useful libraries exist for it. These libraries will further help with achieving requirements of speed and correctness.
We are specifically interested in using the following two Java libraries:

- *Apache Lucene*[1] This will be used for creating search indexes, which will make the search through the huge corpus of tables massively more efficient. The next section contains more information on the search indexes.

- *WInte.r*[2] This library implements several methods for big data integration. It was created by the same research group that had created the Mannheim Search Join Engine (MSJE) two years earlier (Lehmberg, Brinkmann & Bizer, 2017). This library will be used to detect the subject-columns of tables i.e. identify which column of a table contains the entity names. The library will also be used

[1]`https://lucene.apache.org/`
[2]`https://github.com/olehmberg/winter`

to create new, sophisticated instance- and schema- matching functions for the table extension algorithms. These instance and schema- matching functions are described in the sections 3.2.6 and 3.2.7.

For making the table extension operations easily useable, they were integrated as part of a web service with a REST-API. The web service is implemented with the *Java Play Framework*[3]. It allows users to submit a table via http request and receive the extended table as reply. Users are also able to upload tables to create their own corpus of tables. A complete list of all operations can be found in Appendix B. The Java Play Framework helps to ensure high availability of the system – if an individual table extension operation or table upload operation fails, only that one http request fails, the system continues to run normally.

3.2.2 Table repositories

The role of a table repository is to store hundreds-of-thousands of data tables and make them easily accessible to the table extension algorithms. This new table extension system is designed to allow for multiple table repositories, each containing a different corpus of tables. In fact, the user can create new repositories and upload tables to a repository via the REST-API. This ensures the wide applicability of the table extension system – if there is no repository to support table extension operations in a certain domain, then the user can create this repository her-/himself.

In the table extension system, the repository is implemented as just a sub-folder of the repositories-folder on the server (that runs the table extension system). As the user sends tables to the table extension systems (using the upload_table API call), these tables are converted from a json format to a csv format and saved as csv files in this sub-folder.

[3]`https://www.playframework.com/`

3.2.3 Search index

Search indexes were briefly introduced in sections 2.2.2.1 and 2.3.1. To achieve the fast runtime requirement of the table extension system, the tables in each repository are indexed in a search index. This search index is automatically created when the user creates a repository, and when the user uploads tables to this repository, the tables are automatically added to the search index.

The search index used in this system is called SubjectColumnIndex. It contains a record for every table in the repository. In every record, the following information is saved: subjectColumnString (= a list of all the values in the table's subject column concatenated into one long string), subjectColumnIndex, tableName. The Subject-ColumnIndex is used by the table extension algorithm (section 3.2.4) to find tables with similar subject columns to that of the submitted table. This is done in the following way:

1. The entity names in the subject-column of the to-be-extended table are concatenated to a query-string.
2. This query string is then used to look for tables with similar subject columns in the search index. For determining the subject-column similarity, the search index calculates a tf-idf cosine similarity; this is ideal for finding subject columns with many entity names in common.

3.2.4 Unconstrained table extension

The unconstrained table extension algorithm probably presents the biggest innovation of this system. As mentioned, it adds as many new columns to a table as possible. To run it, the user sends a http request to the REST-API of the web service. This http request must contain the to-be-extended table as well as the name of the repository that should be used for the table extension. After 0-10 seconds the response message will arrive containing the extended table i.e. the submitted table with the additional columns that were populated with data from the specified repository.

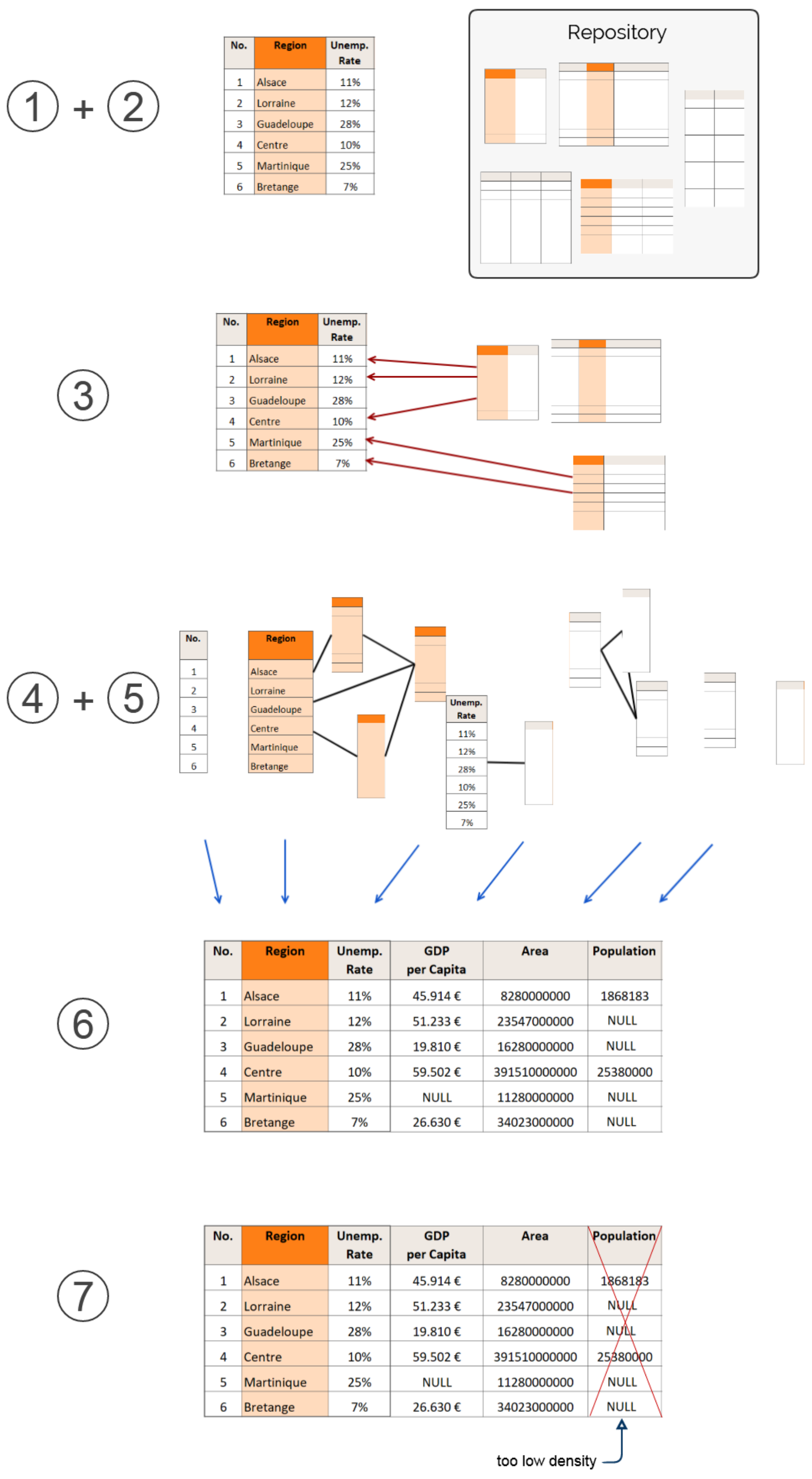
On a high level, the unconstrained table extension searches through the specified repository for tables that contain information about the entities in the submitted table. It then aligns the data from the found tables according to the entities and fuses columns that describe the same attribute.

In concrete, the unconstrained table extension algorithm runs through the following steps – see Figure 3.1:

- Step1: Automatically detect the subject column of the submitted table (using the subject column detection function from the WInte.r library).
- Step2: Using the SubjectColumnIndex (see section 3.2.3), find tables in the repository with similar subject columns.
- Step3: Match entities of the found tables to the entities of the submitted table with the instance matching algorithm described in 3.2.6. Remove tables with no matches.
- Step4: Find schema matches between all columns (those of the submitted table and the found tables). This is done with the schema matching algorithm described in 3.2.7.
- Step5: Cluster the columns according to their similarities.
- Step6: Fuse the clusters by a weighted voting algorithm.
- Step7: If some of the new columns of the extended table have a density below 0.6 (i.e. more than 40% of the values are null), remove them.

The user may add any of the following optional parameters to the http request to change some details of the unconstrained table extension algorithm:

- keyColumnIndex
  In step1, the subject column of the submitted table is automatically detected. This parameter gives the option of specifying the subject column index instead.



No.	Region	Unemp. Rate
1	Alsace	11%
2	Lorraine	12%
3	Guadeloupe	28%
4	Centre	10%
5	Martinique	25%
6	Bretange	7%

No.	Region	Unemp. Rate	GDP per Capita	Area	Population
1	Alsace	11%	45.914 €	8280000000	1868183
2	Lorraine	12%	51.233 €	23547000000	NULL
3	Guadeloupe	28%	19.810 €	16280000000	NULL
4	Centre	10%	59.502 €	391510000000	25380000
5	Martinique	25%	NULL	11280000000	NULL
6	Bretange	7%	26.630 €	34023000000	NULL

No.	Region	Unemp. Rate	GDP per Capita	Area	~~Population~~
1	Alsace	11%	45.914 €	8280000000	~~1868183~~
2	Lorraine	12%	51.233 €	23547000000	~~NULL~~
3	Guadeloupe	28%	19.810 €	16280000000	~~NULL~~
4	Centre	10%	59.502 €	391510000000	~~25380000~~
5	Martinique	25%	NULL	11280000000	~~NULL~~
6	Bretange	7%	26.630 €	34023000000	~~NULL~~

Figure 3.1: Diagram of the execution steps in the unconstrained table extension

- minimumKeyColumnSimilarity

  In step2 the repository is searched for tables with similar subject columns. As a default a minimal cosine tf-idf similarity of 0.6 required, this value can however be changed by this parameter.

- maximalNumberOfTables

  Instead of limiting the number of tables that are found in step2 with the minimumKeyColumnSimilarity, it can be limited by specifying the maximum number of tables that can be found this way.

- minimumInstanceSimilarity

  In step3 instance matching occurs. With this parameter the minimum similarity for an instance correspondence can be changed from the default of 0.7. See section 3.2.6 for more details.

- minimumDensity

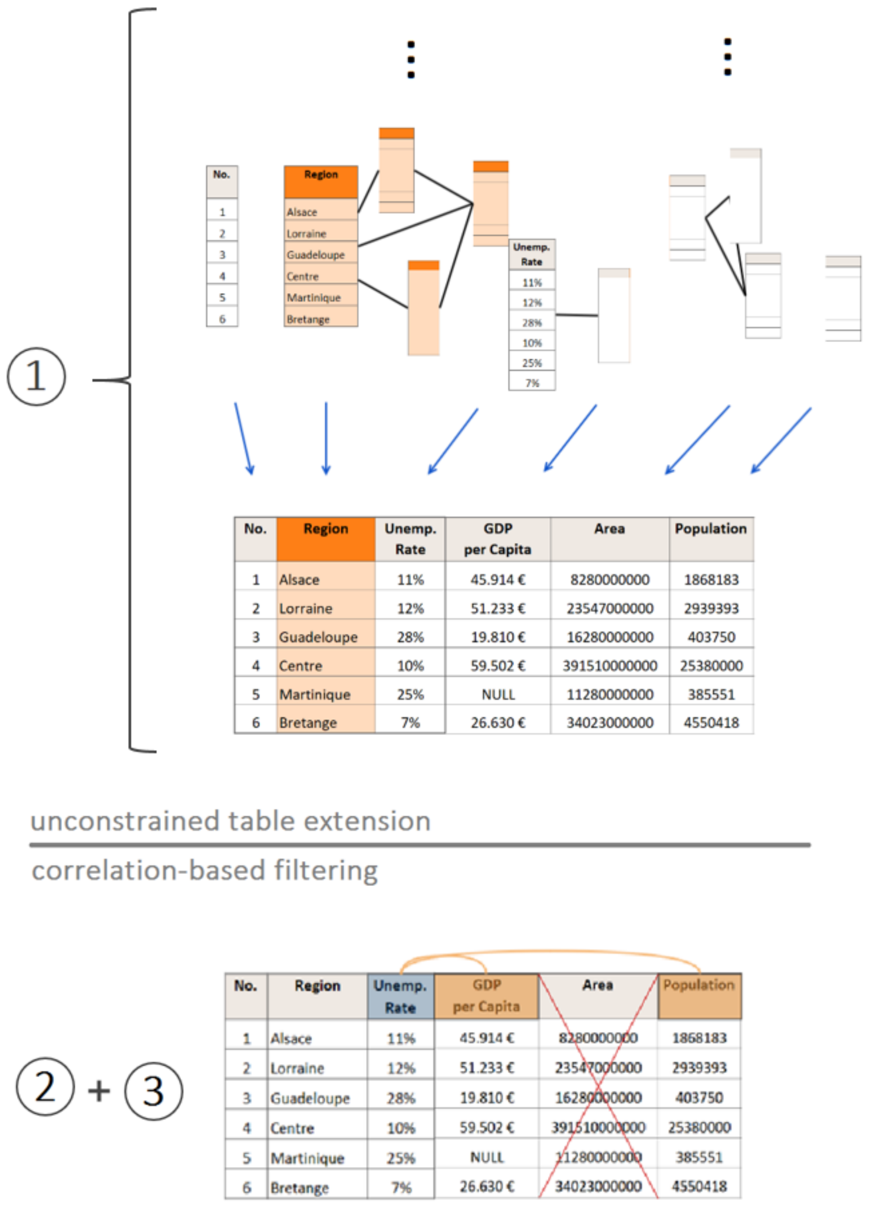
  In step7 the extended table might contain very many columns, some of them very poorly populated. Therefore, as a final step, columns with a density less than 0.6 are removed from the extended table. This parameter allows you to change the threshold to another value.

The default values for these parameters were determined through specific evaluations and large-scale optimization – see section 5.6 and Appendix E.

3.2.5 Correlation-based table extension

Instead of extending a table with as many columns as possible, the correlation-based table extension only extends the submitted tables with columns that correlate with a user-specified attribute of the submitted table – the ‘correlation attribute’.

To run the correlation-based table extension, the user sends a http request to the REST-API of the web service. The http request must contain the to-be-extended table, the name of the correlation attribute and the name of the repository from which the new data should be extracted. After 0-10 seconds the response message will arrive containing the extended table.



No.	Region	Unemp. Rate	GDP per Capita	Area	Population
1	Alsace	11%	45.914 €	8280000000	1868183
2	Lorraine	12%	51.233 €	23547000000	2939393
3	Guadeloupe	28%	19.810 €	16280000000	403750
4	Centre	10%	59.502 €	391510000000	25380000
5	Martinique	25%	NULL	11280000000	385551
6	Bretange	7%	26.630 €	34023000000	4550418

No.	Region	Unemp. Rate	GDP per Capita	~~Area~~	Population
1	Alsace	11%	45.914 €	~~8280000000~~	1868183
2	Lorraine	12%	51.233 €	~~23547000000~~	2939393
3	Guadeloupe	28%	19.810 €	~~16280000000~~	403750
4	Centre	10%	59.502 €	~~391510000000~~	25380000
5	Martinique	25%	NULL	~~11280000000~~	385551
6	Bretange	7%	26.630 €	~~34023000000~~	4550418

Figure 3.2: Diagram of the correlation-based table extension

The correlation-based table extension algorithm has three steps – see Figure 3.2:

- Step1: Run unconstrained table extension on the submitted table.

- Step2: Calculate the correlation between the submitted correlation attribute and the new columns.
- Step3: Remove those new columns whose absolute correlation to the correlation attribute is less than 0.4.

In Step2 different correlation metrics are calculated for different variable combinations. The correlation metrics used are:

- Pearson correlation coefficient – for two numeric variables.
- Scaled p-value from an analysis of variance (ANOVA) – for a numeric and a categorical variable.
- Cramer's V – for two categorical variables.

When calling the correlation-based table extension, the user may add the same optional parameters to the http request as for the unconstrained table extension: keyColumnIndex, minimumKeyColumnSimilarity, maximalNumberOfTables, minimumInstanceSimilarity, minimumDensity. For the correlation-based table extension, there is the additional optional parameter minimumCorrelation. This allows the user to change the minimal acceptable correlation for new columns from 0.4 to any other value – see Step3.

3.2.6 Instance matching

In step3 of the unconstrained table extension (see section 3.2.4), instance matching occurs. As the correctness of the table extension algorithms depends strongly on good instance matching, a custom instance matching algorithm was developed and evaluated (the details about the evaluation are given in Appendix E.3).

The goal of the instance matching algorithm is to accurately identify instance correspondences. These are two rows from two different tables that both describe the same real-world entity. E.g. identifying that the row describing 'Ireland' in the one table corresponds to the row describing ‘Éire’ in the other table.

To find the instance correspondences between two tables, the instance matching algorithm calculates a similarity score for each row combination between these two tables. Given these similarities, the Hungarian algorithm (Kuhn, 1955) is applied to find the optimal set of correspondences (considering that a row from one table can only correspond to one row from the other table). Finally, instance correspondences with a similarity less than 0.7 are rejected.

This similarity score between any two rows is the average of the following two similarity scores:

- entity name similarity
  this is the fuzzy-jaccard string similarity (Levandowsky Winter, 1971) calculated on the subject column values from the two rows.

- the similarity of the remaining values.
  This similarity is the percentage of column values from the shorter row, that have a matching value in the other row.
  To calculate this, the following is done: for every non-subject-column-value in the shorter row, calculate similarities to the non-subject-column-values of the other row; if any of the similarities is above 0.8 it is considered a matching value. The similarity of the remaining column values is the percentage of values from the shorter row for which a corresponding value was found in the longer row.

It is also important to note, that in the above algorithm data-type-specific similarity values are used. This means, when comparing two numeric values, the similarity is the ratio of the two numbers, for other data type combinations, the similarity is the fuzzy-jaccard string similarity.

3.2.7 Schema matching

Schema matching is done in step4 of the unconstrained table extension (see section 3.2.4). The correctness of the table extension algorithm depends strongly on the correct schema correspondences being determined. Therefore, a custom schema matching algorithm was developed and evaluated (the details of the evaluations can be found in Appendix E.4).

The schema matching algorithm tries to identify schema correspondences between two tables i.e. columns from the two tables that describe the same attribute. E.g. it tries to identify that the ‘Population’-column of the one table corresponds to the ‘number of inhabitants’-columns of the other table.

When the schema matching is executed (in step 4 of the unconstrained search), the instance correspondences between the tables are already known – from the instance matching in step 3. The schema matching algorithm uses this knowledge about the instance correspondences to more accurately find the schema correspondences. This algorithm is therefore called ‘instance-based schema matching’.

To find the schema correspondences between two tables, the algorithm calculates a similarity score for each pair of columns. Next, the Hungarian algorithm (Kuhn, 1955) is used to find those column-pairs that have the maximum similarity (considering, that a column from one table may only match one column from the other table). To count as schema correspondence these pairs are required to additionally have a similarity above 0.8.

The column similarity is the weighted sum of the column-header similarity (weight: 0.2) and the column-value similarity (weight: 0.8). The column-header similarity is simply the fuzzy-jaccard string similarity between the two headers.
The column-value similarity between any two columns is calculated as follows: From the two columns you extract pairs of values – one for each of the previously found in-

stance correspondences. If for example the two columns do correspond to each other, then the pairs are pairs of exactly corresponding values (E.g. the population of Ireland and the number of inhabitants in Éire). If a pair has a similarity of over 0.8, it is considered a match. The similarity between any two columns is the fraction of pairs that were found to match.

3.3 Design of the Evaluations

As mentioned in the introduction to this chapter, the requirements for a viable table extension system are: *High usability*, *High availability*, *Wide applicability*, *Fast execution time* and *Correctness*. There will be a separate evaluation for each of these requirements.
The most attention is however given to the evaluation of the correctness, as this evaluation had also been performed on the existing table extension systems and will allow a direct comparison of the new table extension system with the existing table extension systems.

To run the evaluations, the table extension system is run as a web service with a public url. The operations (e.g. unconstrained table extension) can be executed from anywhere with an internet connection by making the appropriate API call. There are API calls for table extension, creating a repository, uploading tables to a repository and many more – the full list is given in Appendix B.

The base urls for the API calls are:

- http://ds4dm.informatik.uni-mannheim.de
  Public version of the system. It has been continuously running and accessible from the 19th April 2018. When in this dissertation 'the table extension system' is mentioned, this is the system that is referred to.

- http://ds4dm-experimental.informatik.uni-mannheim.de

Experimental version of the system. Here new versions of the algorithm were tested. It was subject to frequent changes and outages.

3.3.1 High usability

According to ISO 9241, usability is "the effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments".

According to Virzi (1992) it is essential for evaluating the usability of a software system to have evaluators that were not involved with the design or implementation of the system, as they don't have prior understanding of the system. This dissertation was fortunate to have four such evaluators, who provided continuous evaluation of the usability throughout the development of the system: Dr. Edwin Yaqub[4], Dr. Fabian Temme[5], Philipp Schlunder[6] and Prof. Dr. Christian Bizer[7]. The first three are employees of the company RapidMiner GmbH[8] that used the new table extension system to develop an extension to their data analysis platform – more information on this collaboration is given in Appendix D.

For the evaluation of the usability, the evaluation methodology presented in Jakob Nielsen's 1994 paper is used. This method started the "discount usability engineering" movement[9] and is still commonly used today[10].

The main idea of this methodology is that instead of getting a large group of people to evaluate the product at the end of the development, small groups of users (between 3 and 5) evaluate all changes to the product throughout the development. These evaluators are supposed to be representative users, performing representative

[4]https://www.linkedin.com/in/edwin-yaqub-55775324/

[5]https://www.linkedin.com/in/thomas-fabian-temme-4a9413124/

[6]https://www.linkedin.com/in/philipp-schlunder-8a3406a5/

[7]https://www.linkedin.com/in/chrisbizer/

[8]https://rapidminer.com/

[9]https://en.wikipedia.org/wiki/Jakob_Nielsen_(usability_consultant)

[10]https://en.wikipedia.org/wiki/Nielsen_Norman_Group

tasks.

The usability evaluation was set up as follows: the four evaluators Edwin Yaqub, Fabian Temme, Philipp Schlunder and Christian Bizer are experienced IT- and data-professionals, which is representative for the target users of the table extension system.

They evaluated the usability after every change to the system, doing the representative tasks (mainly table extension and repository creation). The evaluations with Christian Bizer were on-site evaluations. Most of the evaluations with the other users were asynchronous remote evaluations (Andreasen, Nielsen, Schrøder & Stage, 2007), whereby an on-site evaluation took place during a workshop on the 30th May 2018.

The results of this evaluation are presented in section 4.3.1.

3.3.2 High availability

Availability is “the probability that an item will operate satisfactorily at a given point in time when used under stated conditions in an ideal support environment” (Cochrane & Hagan, 1998).

As previously mentioned, the new table extension system was continuously running from the 19th April 2018. Occasional outages due to system failure did however occur. These outages were tracked for the five month period from 19th April 2018 to 19th September 2018. The availability of the system is evaluated by analysing these downtimes. Specifically, the following commonly used metrics were calculated:

- Operational Availability (Ao)

$$A_o = \frac{T_m - T_d}{T_m} \qquad (T_m = MissionDuration; T_d = ObservedDownTime)$$

- Mean Time Between Failures (MTBF)

- Mean Time To Recover (MTTR)

The results of this evaluation are presented in section 4.3.2.

3.3.3 Wide applicability

The table extension system guarantees wide applicability by permitting users to create their own repositories for table extension. If a user wants to extend tables in a specific domain, he/she can create a repository with tables in this domain to do so. For instance, a company might upload internal company tables to a repository for extending tables with these internal company data.

To evaluate whether the table extension algorithm has a comparable performance across topics, the evaluation of the correctness (see section 3.3.5) will be performed with different repositories, on tables covering a wide range of topics. The evaluation results are analysed in section 5.3.

3.3.4 Fast execution time

The runtime of the table extension algorithm increases with the size of the repository used for it. To give a worst-case estimate of the execution time, the evaluated table extension operations will be performed with a repository containing over 460 thousand tables – the so-called Web Table Corpus[11].
The tables in the Web Table Corpus were extracted from the Wikipedia pages found by the Common Crawl[12]. The tables in the repository therefore cover all the spectrum of topics you might find on Wikipedia pages.
For the evaluation of the execution times, the execution times for the table extensions of 13 different tables were measured. The results are discussed in section 4.3.4.

[11]`http://webdatacommons.org/webtables/#results-2015`
[12]`https://commoncrawl.org/`

3.3.5 Correctness

This is the most important evaluation, as it is the only evaluation which was also performed for the existing table extension systems – Octopus (Cafarella, Halevy & Khoussainova, 2009), InfoGather (Yakout, Ganjam, Chakrabarti & Chaudhuri, 2012) and the MSJE (Lehmberg et al., 2015). The evaluation of the correctness will therefore allow a direct comparison between the systems.

The methodology used for evaluating the correctness of the new table extension algorithms matches exactly the methodology used in the above papers: Individual tables will be extended using the algorithm; for each extended table, precision and density are calculated by comparing the new column with the truth. The truth is the column with the correct values which were obtained from reliable sources (usually websites). Precision and density are calculated as follows:

$$precision = \frac{TP}{TP + FP} = \frac{\#values\_correctly\_populated}{\#values\_populated} \tag{3.1}$$

$$density = \frac{\#populated\_fields}{\#fields\_to\_be\_populated} \tag{3.2}$$

(TP stands for 'true positives' and FP stands for 'false positives').

In section 4.3.5, all the details of the evaluation are presented, including the tables and repositories that were used, as well as the evaluation results.

3.4 Conclusion

In this chapter the design of both the new table extension system and of the evaluation of this table extension system were presented.

At the beginning of the chapter, the following five key requirements have been identified for the design of the table extension system: *High usability*, *High availability*, *Wide applicability*, *Fast execution time*, *Correctness*.

These five key requirements motivated all design decisions for the new table extension system. To achieve high usability and high availability, the system is run as a web service – different operations can be run by making http requests to its public API. The design of the actual table extension algorithms has been optimized for fast execution and correctness – search indexes improve the execution time and the instance- and schema- matching algorithms have been intricately designed and optimized to guarantee the most correct results possible.

The performance-critical instance- and schema- matching algorithms achieve maximal correctness by using all available knowledge: knowledge about the subject columns, knowledge about instance correspondences, etc. Furthermore, the matching algorithms use data-type-specific similarity measures, compare column headers as well as column values, and use parameters and thresholds which have been optimized for maximal correctness – see Appendix E.3 and E.4.

For the evaluation a holistic approach was chosen with a separate evaluation for each of the key requirements of the system:

- The *usability* is continuously being evaluated by four external users.
- The *availability* is evaluated by tracking and analysing all outages that occur over a 5-month period.
- *Wide applicability* is guaranteed by the architecture which allows for custom repository creation.
- The *execution time* is measured for multiple table extensions.
- The *correctness* is evaluated by running multiple table extensions and comparing the values in the new columns with the true values. From this the evaluation metrics precision and density are calculated.

The next chapter will illustrate how the designed system and the planned evaluations were implemented in practice. It describes how the system was used and what the evaluation results are.

Chapter 4

Implementation and Results

4.1 Introduction

In the previous chapters of this dissertation, the motivation for automatic table extension was discussed (chapter 1), the existing table extension systems were analysed (chapter 2) and a new table extension system along with the evaluations of this table extension system was designed (chapter 3).
This fourth chapter will now highlight how the table extension system was used, and how well it performed in practice.

In the first section, a brief overview of the new table extension system is given. In the second section of this chapter, the evaluation results are presented. As designed in chapter 3, there were five separate evaluations – one for each of the five key requirements: *High usability*, *High availability*, *Wide applicability*, *Fast execution time* and *Correctness*.

4.2 The Table Extension System

The core-functionality of the table extension system is contained in 55 java classes containing 9535 lines of code. The table extension system was implemented and evaluated over a 5-month period in 2018.

The entire code of the table extension system is freely available on GitHub[1]. It is saved in such a way, as to be easily useable by anybody – to setup and run the table extension system on your own computer, you only have to download the code and execute some command line statements. The readme-file[2] of the GitHub repository gives a detailed description of how to setup and run the table extension system on your own computer – see Figure 4.1.

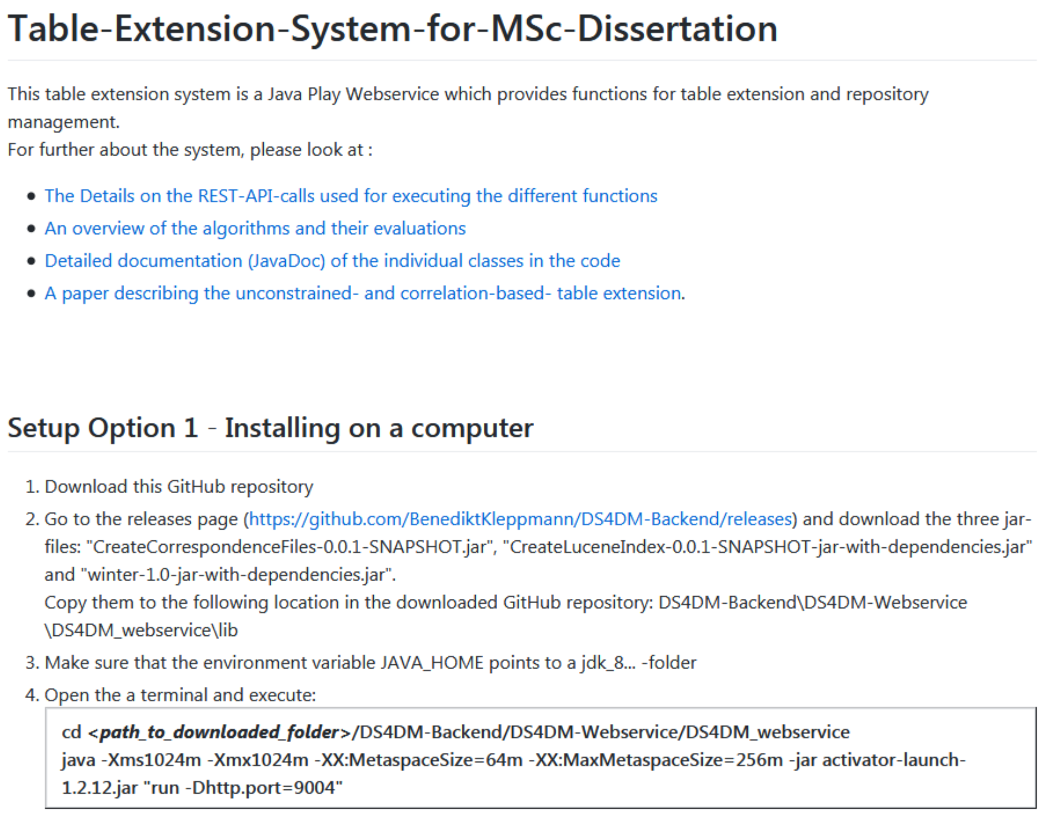


Table-Extension-System-for-MSc-Dissertation

This table extension system is a Java Play Webservice which provides functions for table extension and repository management.
For further about the system, please look at :

- The Details on the REST-API-calls used for executing the different functions
- An overview of the algorithms and their evaluations
- Detailed documentation (JavaDoc) of the individual classes in the code
- A paper describing the unconstrained- and correlation-based- table extension.

Setup Option 1 - Installing on a computer

1. Download this GitHub repository
2. Go to the releases page (https://github.com/BenediktKleppmann/DS4DM-Backend/releases) and download the three jar-files: "CreateCorrespondenceFiles-0.0.1-SNAPSHOT.jar", "CreateLuceneIndex-0.0.1-SNAPSHOT-jar-with-dependencies.jar" and "winter-1.0-jar-with-dependencies.jar".
   Copy them to the following location in the downloaded GitHub repository: DS4DM-Backend\DS4DM-Webservice \DS4DM_webservice\lib
3. Make sure that the environment variable JAVA_HOME points to a jdk_8... -folder
4. Open the a terminal and execute:

```
cd <path_to_downloaded_folder>/DS4DM-Backend/DS4DM-Webservice/DS4DM_webservice
java -Xms1024m -Xmx1024m -XX:MetaspaceSize=64m -XX:MaxMetaspaceSize=256m -jar activator-launch-1.2.12.jar "run -Dhttp.port=9004"
```

Figure 4.1: Screenshot of the readme-file in the GitHub repository

For the new table extension system, extensive documentation was created in order to enable others to use and build upon the work. The documentation is described in

[1] https://github.com/BenediktKleppmann/Table-Extension-System-for-MSc-Dissertation

[2] https://github.com/BenediktKleppmann/Table-Extension-System-for-MSc-Dissertation/blob/master/README.md

more detail in Appendix C. It includes:

- JavaDoc documentation
- Swagger API specification
- A short conference paper about the two table extension algorithms (Kleppmann et al., 2018)

As previously mentioned, the new table extension system was used by the company RapidMiner to build table-extension-operators for their well-known data mining tool RapidMiner Studio. Details on the collaboration with RapidMiner are given in Appendix D.

4.3 Evaluation Results

Five evaluations were performed for the five key requirements of the table extension system: *High usability*, *High availability*, *Wide applicability*, *Fast execution time*, *Correctness*.

The basic design for each of these evaluations was presented in section 3.3. This section describes the practical challenges in implementing them as well as the results.

As previously mentioned, the correctness is the most important evaluation, as results of this evaluation allow the new table extension system to be compared with the existing ones.

4.3.1 High usability

As mentioned in section 3.3.1, there were four external evaluators that evaluated the usability of the new table extension system after every change to the interface. These external evaluators were Dr. Edwin Yaqub, Dr. Fabian Temme, Philipp Schlunder and Prof. Dr. Christian Bizer.

The feedback from the usability evaluations was used to make many improvements to the interface of the table extension system. Specifically the following improvements were triggered by user feedback:

- 27th April 2018

  The swagger-API-documentation of the API-interface to the new table extension system was published to communicate the required format of the API calls more clearly. More information on this documentation is given in Appendix C.2.

- 3rd May 2018

  The format of the extended tables returned by the unconstrained- the correlation-based- table extension was changed. In the JSON string of the http-response, the table format changed from dictionary-of-lists to list-of-lists.

- 4th May 2018

  The way null-values are represented was changed from "null" to "". This change was applied to tables that are being uploaded to a repository, tables that are submitted to any of the table extension operations and tables returned by the table extension operations.

- 25th May 2018

  The additional operation *getUploadStatus* was added to the table extension system. The bulkUploadTables operation now returns an upload id. When the user calls getUploadStatus with this upload id, the progress of the bulk upload operation is returned. (With the bulkUploadTables operation many tables can be uploaded at once. Understandably this might take some time and it is useful for the user to follow the progress.)

- 18th June 2018

  Two new operations were added to the table extension system: getRepositoryNames and getRespositoyStatistics. GetRepositoryNames returns the names of all repositories in the system. GetRespositoyStatistics returns the creation timestamp of and the number of tables in the specified repository.

- 29th July 2018

  Two optional configuration parameters were added to the unconstrained- and

the correlation-based-table extension operations:
minimumKeyColumnSimilarity and minimumInstanceSimilarity.

Evaluation results

The initial system was hard to use, especially as the provided example API calls left room for interpretation. This was soon changed with the swagger-API-documentation, which provides exact and easy to understand guidelines for the format of the API calls. Now it was easy for new, technically-well-versed users to start using the table extension system. Both Philipp Schlunder and Dr. Fabian Temme got familiarized with it in less than 3 hours.

In May some minor changes to the output format were done to adapt it to RapidMiner's format. For other users, this format change has no disadvantages. From here on the easy usability of the system seemed to be established – all changes that were proposed from then on created additional functionality, especially functionality that provides the user with more information or allows more control and flexibility.

When adding additional functionality, attention was paid to striking a balance between fulfilling all the needs of users who already know the system and not making the functionality too complex for newcomers. The result is a system that is usable – it is currently being used by RapidMiner and the University of Mannheim is considering using it in a practical class for their Master in Data Science course (state: 20th October 2018).

4.3.2 High availability

The new table extension system was continuously running on a server for the five month period from 19th April 2018 to 19th September 2018. During this time nine outages occurred – they are listed in Table 4.2. The outages were detected by either the author of this dissertation or the external users, who then informed the author.

Outage-fixed timestamp	Approximate outage time (h)	Outage reason
4/30/2018 15:00	1	Unknown reason
5/20/2018 10:00	60	Unknown reason
5/31/2018 16:00	3	Crashed due to massive table upload
6/27/2018 9:00	16	Unknown reason
7/9/2018 13:00	3	Bug in previous release
7/30/2018 18:00	60	Unknown reason (probably related to release)
7/31/2018 13:00	2	Faulty folder-configuration
9/3/2018 10:00	48	Unknown reason
9/5/2018 14:00	24	Unknown reason

Table 4.2: Outages of the table extension system

These are acceptable availability metrics, especially for a research project. This performance is achieved mainly by using the Java Play web service framework, which keeps on running even if individual table extension requests fail. In section 5.3 the results are analysed in detail.

4.3.3 Wide applicability

As mentioned in section 3.3.3, wide applicability is intrinsically guaranteed by the new table extension system, as users can create custom repositories for any desired application domain.

To evaluate whether the performance of the table extension algorithms remains stable across topics, the evaluation of correctness was performed with different repositories, on tables covering a wide range of topics. These evaluation results are presented in section 4.3.5 and analysed in section 5.4.

4.3.4 Fast execution time

Three of the table extension system's operations are particularly time intensive: uploading large amounts of data to a repository, the unconstrained table extension and the correlation-based table extension. These operations are time intensive, because they require processing very many data tables. The execution time of all three of these operations was evaluated on a machine with 8GB of RAM and a 3.1GHz processor. The results are:

Table upload

Uploading 460 513 data tables to a repository took 28 hours. Whereby the copying of the tables to the repository folder took 69 minutes and the indexing of these tables took 1587 minutes. This corresponds to an upload time of 0.215 seconds per table.

It is interesting to note, that the time for indexing (= entering new records into a search index) increases non-linearly. I.e. the first tables are indexed faster, later tables have to be sorted into the existing index, which takes longer.

Unconstrained table extension and correlation-based table extension

The time the system needs for extending a table was measured for both the unconstrained and the correlation-based table extension. 13 different tables were extended with both the unconstrained table extension algorithm and the correlation-based table extension algorithm. For these extensions the large repository with 460 513 tables from Wikipedia was used. The times that these extension operations took, as well as some additional information about the extended tables, is shown in Table 4.4.

The execution times were measured by a small program, that logged the time, set off the extension operation and logged the time again immediately after its completion.

Table subject	**Number of rows**	**Total execution time for unconstrained search (s)**	**Total execution time for correlation-based search (s)**
mountains	41	3.854990	4.897614
airlines	243	5.695331	6.122556
fells	214	4.429745	5.056925
irish_political_parties	7	4.015009	3.920874
airports	137	4.392987	5.045750
currencies	135	7.887399	8.546373
lakes	29	8.768592	8.997190
animals	153	6.287221	6.804818
sky_scrapers	46	3.380996	3.357822
roman_emperors	81	3.934930	3.912603
hospitals	23	3.580877	4.560518
journals	35	2.491133	3.026299
museums	20	4.908730	6.628930

Table 4.4: Execution times for the two table extension operations

The median execution time for the unconstrained table extension is 4.39 seconds. The median of the correlation-based table extension is 5.05 seconds.

It makes sense that the correlation-based table extension takes a bit longer, because it consists of unconstrained table extension *plus* correlation-based filtering (more information can be found in section 3.2.5).

4.3.5 Correctness

Correctness is a very important requirement for the table extension system. It is also the only requirement, which was also evaluated by the other table extension algorithms: Octopus (Cafarella, Halevy & Khoussainova, 2009), InfoGather (Yakout, Ganjam, Chakrabarti & Chaudhuri, 2012) and the Mannheim Search Join Engine (Lehmberg et al., 2015). In this section the evaluation results are presented. In section 5.6, these results will then be compared to the performance of Octopus, InfoGather and the Mannheim Search Join Engine.

4.3.5.1 Correctness of the unconstrained table extension

Two different evaluations of the unconstrained table extension algorithm were performed. For these evaluations, different repositories of tables were used, and different tables were extended. This was done to get a more comprehensive view of the quality of the results produced by the unconstrained table extension.

4.3.5.1.1 Evaluation with data from Wikipedia The best of the existing table extension algorithms – the Mannheim Search Join Engine (MSJE) – was evaluated by extending 7 different tables and calculating the density and precision of the extended columns – see section 2.3.3.1.
To allow a good comparison between the MSJE and the new table extension system, the evaluation of the new table extension system exactly matches that of the MSJE (Lehmberg et al., 2015): the same tables were extended, the repository contained the same corpus of tables and the same quality metrics were calculated.

Extended tables
Table 4.6 shows the seven tables used for both the evaluation of the MSJE and that of the unconstrained table extension. These seven tables had been extracted from trusted webpages (see column 'Table source' in Table 4.6). Other than the subject column (= the column with the entity names), these tables also have other columns/attributes – shown in the third column of Table 4.6. These other columns

are taken to be the ground truth. In the evaluation, the data values in the new, extended columns are compared with the correct values given by this ground truth columns from the trusted webpages.

Extended Table	**Number of entities**	**Columns where the ground truth is known**	**Table source**
Books	100	Author	http://www.bbc.co.uk/arts/bigread/top100.shtml
Companies	50	Headquarter, Industry	http://archive.fortune.com/magazines/top50/
Countries	201	Currency, Population, Area, Capital, Code	http://polgeonow.com/2011/04/how-many-countries-are-there-in-the-world.html
drugs	100	Ingredients	http://rxlist.com/script/main/art.asp
Films	100	Cast, Director, Genre, Year	http://www.listchallenges.com/empire-magazines-500-greatest-films-of-all-time/
Songs	100	Artist	http://www.songlyrics.com/news/top-sons/all-time/

Soccer Players	100	Team	http://www.theguardian.com/football/datablog/world-best-footballers-top-100-list

Table 4.6: Tables used for the evaluation of both the unconstrained table extension and MSJE

Repository

For the evaluation a repository was used which contains the 460 513 tables of the WikiTables Dataset[3]. The WikiTables Dataset had been created by Bhagavatula, Noraset & Downey in 2013 by crawling the Wikipedia pages extracted by the Common Crawl[4]. The algorithm from Bhagavatula, Noraset & Downey automatically identified data tables on these HTML-pages and extracted them. The tables extracted this way cover a very broad range of topics, such as chemicals, populated places, sports teams, animals, etc.

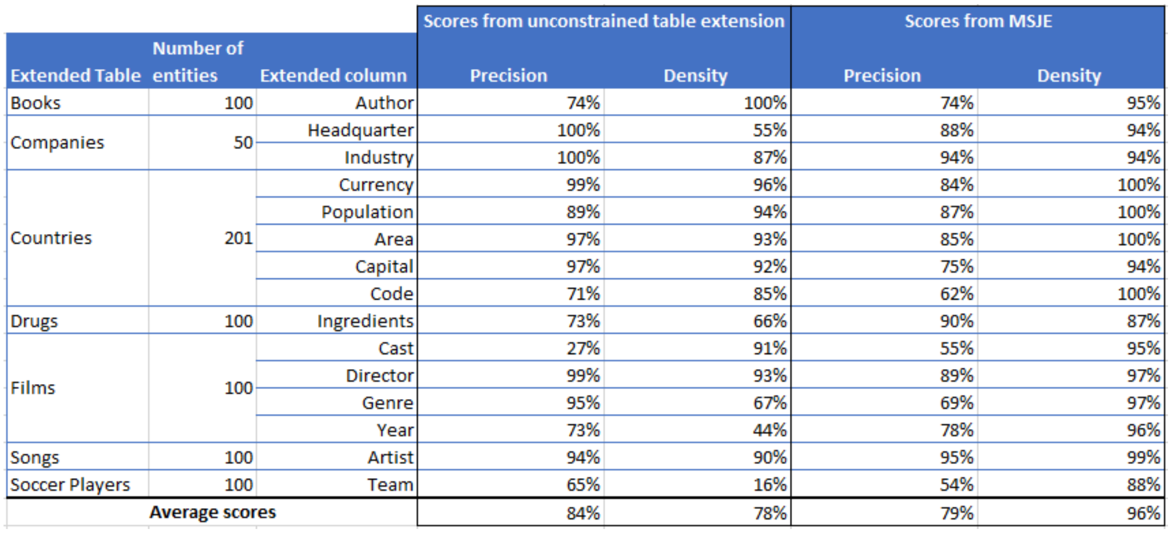
By using this big repository with many tables covering a broad range of topics, both the MSJE and the new table extension system were able to achieve good table extension results.

Quality metrics For both the MSJE and for the unconstrained table extension algorithm, the quality metrics described in section 3.3.5 are used: precision and density.

Evaluation execution There is a small difference between the evaluation of

[3]http://downey-n1.cs.northwestern.edu/public/

[4]https://commoncrawl.org/



Extended Table	Number of entities	Extended column	Scores from unconstrained table extension		Scores from MSJE	
			Precision	Density	Precision	Density
Books	100	Author	74%	100%	74%	95%
Companies	50	Headquarter	100%	55%	88%	94%
		Industry	100%	87%	94%	94%
Countries	201	Currency	99%	96%	84%	100%
		Population	89%	94%	87%	100%
		Area	97%	93%	85%	100%
		Capital	97%	92%	75%	94%
		Code	71%	85%	62%	100%
Drugs	100	Ingredients	73%	66%	90%	87%
Films	100	Cast	27%	91%	55%	95%
		Director	99%	93%	89%	97%
		Genre	95%	67%	69%	97%
		Year	73%	44%	78%	96%
Songs	100	Artist	94%	90%	95%	99%
Soccer Players	100	Team	65%	16%	54%	88%
Average scores			84%	78%	79%	96%

Table 4.7: Comparison of the Evaluation results for the unconstrained table extension algorithm and the MSJE

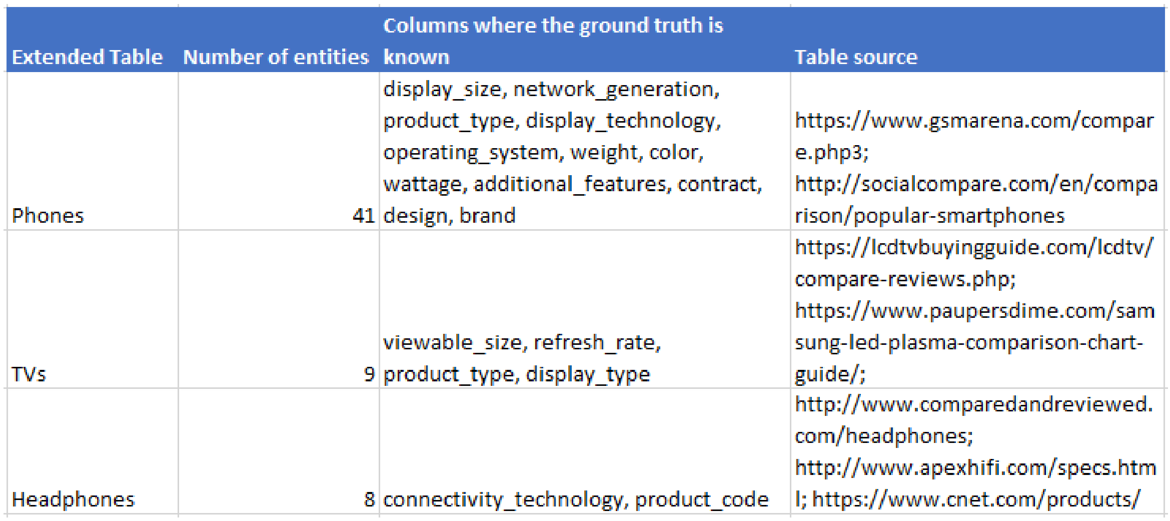
the MSJE and the new table extension system. The MSJE adds columns individually to the table, whereas the new table extension system adds all columns to the table at once. Therefore, the table extension with the MSJE engine was done in multiple table extension steps, while with the new table extension system it was done in only one step. This difference does however not affect the performance of any of the table extension systems. All other details about the evaluation are the same for both table extension systems.

Evaluation results Table 4.7 shows the evaluation results from the unconstrained table extension next to the evaluation results from the MSJE. A detailed discussion and analysis of the results is given in chapter 5.

A detailed analysis of these results is performed in section 5.6.

4.3.5.1.2 Evaluation with Product Data Extended tables

Table 4.8 shows the three tables that were extended for this evaluation. Reading example: the first table used in the evaluation contains the product specifications (display_size, network_generation, product_type, etc.) of 41 smartphones, which had



Extended Table	Number of entities	Columns where the ground truth is known	Table source
Phones	41	display_size, network_generation, product_type, display_technology, operating_system, weight, color, wattage, additional_features, contract, design, brand	https://www.gsmarena.com/compare.php3; http://socialcompare.com/en/comparison/popular-smartphones
TVs	9	viewable_size, refresh_rate, product_type, display_type	https://lcdtvbuyingguide.com/lcdtv/compare-reviews.php; https://www.paupersdime.com/samsung-led-plasma-comparison-chart-guide/;
Headphones	8	connectivity_technology, product_code	http://www.comparedandreviewed.com/headphones; http://www.apexhifi.com/specs.html; https://www.cnet.com/products/

Table 4.8: Tables used for evaluating the unconstrained table extension

been extracted from the websites https://www.gsmarena.com/compare.php3 and http://socialcompare.com/en/comparison/popular-smartphones.

Repository

In previous work by Petrovski, Bryl & Bizer (2014), 19 different webshops had been crawled for product-specification data on headphones, phones and TVs. This data had been extracted and fused into 31 different tables – one table per webshop plus product type (headphones, phones or TVs). For many webshops only one product type is available, that's why there are fewer than 19x3 tables. These 31 tables with product data were uploaded into the repository which was used for running this evaluation.

Quality metrics

As for all other table extension evaluations, the density and precision of the new columns was measured.

Evaluation execution

Extended Table	Number of entities	Extended column	populated values	true_positives	false_positives	Precision	Density
Phones	41	wattage	24	12	12	50%	59%
		network_generation	28	17	11	61%	68%
		product_type	28	20	8	71%	68%
		display_technology	24	20	4	83%	59%
		operating_system	23	13	10	57%	56%
		weight	24	17	7	71%	59%
		color	33	24	9	73%	80%
		display_size	28	22	6	79%	68%
		additional_features	25	6	19	24%	61%
		contract	24	15	9	63%	59%
		design	28	25	3	89%	68%
		brand	36	31	5	86%	88%
TVs	9	viewable_size	6	6	0	100%	67%
		refresh_rate	5	3	2	60%	56%
		product_type	5	3	2	60%	56%
		display_type	5	2	3	40%	56%
Headphones	8	product_code	4	1	3	25%	50%
		connectivity_technology	4	4	0	100%	50%
					Average scores:	64%	63%

Table 4.9: Results from evaluating the unconstrained table extension algorithm with product data

The evaluation with the Product Data was performed in the same way as the evaluation with data from Wikipedia — see section 4.3.5.1.1.

Evaluation results

Table 4.9 shows the results from evaluating the unconstrained table extension algorithm with product data. The precision and density of each of the new, extended columns has been calculated separately to allow for a more detailed analysis (see section 5.6).

A detailed analysis of these results is performed in section 5.6.

4.3.5.2 Correctness of the correlation-based table extension

As previously mentioned, the correlation-based table extension is very similar to the unconstrained table extension, just that instead of extending the column with as many new columns as possible, it extends it only with the columns that correlate with a specified attribute of the original table – the 'correlation attribute'.
The algorithm of the correlation-based table extension has two steps: (1) the unconstrained table extension is run to add all columns to the table; (2) correlation-based filtering is run on this extended table to remove the columns that do not correlate with the correlation attribute.
The quality of data in the new columns added has already been evaluated in the previous section (4.3.5.1), it therefore only remains to evaluate whether the correlation-based table extensions adds the correct new columns to the table i.e. whether the correlation-based filtering works correctly.

The correlation-based filtering removes columns that do not correlate with the correlation attribute i.e. columns whose correlation with the correlation attribute is below the threshold of 0.4. (The threshold of 0.4 was arbitrarily selected as a sensible default, the user may change this value with the input parameter 'minimum-Correlation' - see API documentation[5]). The only error that can occur with the correlation-based filtering is that it identifies the wrong columns as correlating:
Truly correlating columns are those columns that correlate in the ground truth. The correlation-based filtering however does not receive the ground truth tables as input, but the imperfectly extended table from the unconstrained table extension. In this imperfectly extended table individual column values are missing or wrong, possibly causing the correlation for that column to be wrongly identified.

The quality of the correlation-based filtering was evaluated by comparing the correlations found in the imperfectly extended tables to the correlations in the ground

[5]`http://web.informatik.uni-mannheim.de/ds4dm/API-definition.html#/search/correlationBasedSearch`

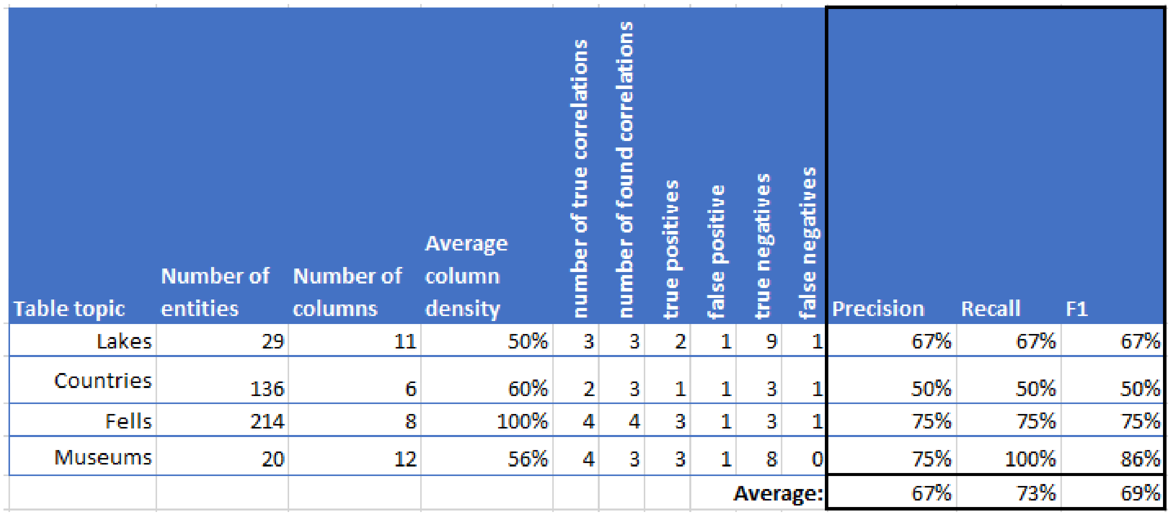


Table topic	Number of entities	Number of columns	Average column density	number of true correlations	number of found correlations	true positives	false positive	true negatives	false negatives	Precision	Recall	F1
Lakes	29	11	50%	3	3	2	1	9	1	67%	67%	67%
Countries	136	6	60%	2	3	1	1	3	1	50%	50%	50%
Fells	214	8	100%	4	4	3	1	3	1	75%	75%	75%
Museums	20	12	56%	4	3	3	1	8	0	75%	100%	86%
									Average:	67%	73%	69%

Table 4.10: Evaluation results of correlation-based filtering

truth. Table 4.10 shows the precision- and recall- scores resulting from this evaluation. As can be seen in Table 4.10, this evaluation was performed on only four tables. These four tables (all from WikiTables) are the only evaluation tables which had a sufficient number of numeric columns. Numeric columns are important, because the correlation-based filtering was changed to only look for correlations between numeric columns after it was discovered, that correlating categorical variables are mostly surface forms and contain no useful information for the user.
The ground truth for these tables was created by fusing several tables from the T2D corpus (more information on the T2D corpus is given in Appendix E.2). The imperfect extended tables were created by extending the subject column of the ground truth tables with an unconstrained table extension which used the WikiTables as repository.

Section 5.6 contains a detailed analysis of these results.

4.4 Conclusion

This chapter contains a description of the concrete outputs of this dissertation – the table extension system and the evaluation results. For the dissertation, a big emphasis was put on the produced work being useful to other researchers and data professionals – this means that not only the evaluation results are informative and comprehensive, but also that the implemented table extension system is easily usable by other researchers and data professionals. The two sections in this chapter describe how these two goals were achieved.

In the first section (4.2) there is a brief description of the new table extension system that was developed to evaluate the new table extension algorithms. The table extension system has been made available on GitHub and has been extensively documented – see Appendix C. This makes it easy for other people to use the system. RapidMiner for instance used the new table extension system to implement an extension to their popular data mining tool RapidMiner Studio. This extension developed by RapidMiner makes the functionality of the table extension system available from within RapidMiner Studio – more information on this is given in Appendix D.

In the second section (4.3), the evaluation results are described. For each of the five key requirements (*High usability*, *High availability*, *Wide applicability*, *Fast execution time* and *Correctness*) a separate evaluation was performed following the evaluation methodologies developed in chapter 3. These evaluations had the following results:

- High usability
  High usability was achieved after multiple iterations of the interface and documentation with the assistance of external evaluators from RapidMiner.
- High availability
  From observing the outages occurring over a 5-month-period of operation, an operational availability Ao of 0.941 was calculated.
- Wide applicability
  The table extension system achieves a wide applicability by allowing users to

create their own repositories with tables on arbitrary domains.

- Fast execution time

  The execution time for uploading tables to a repository was measured to be 0.215 seconds per table. The median execution time for the unconstrained table extension was measured to be 4.39 seconds and for the correlation-based table extension 5.05 seconds.

- Correctness

  Both the correctness of the unconstrained table extension and the correlation-based filtering were evaluated.

  Two different evaluations of the unconstrained table extension were performed in order to get the best possible understanding of its correctness. For the first, the data and methodology completely matched the evaluation of the Mannheim Search Join Engine (Lehmberg et al., 2015); here an average precision and density of 78% and 96% were achieved. The second evaluation was performed with product data and achieved an average precision and density of 64% and 63%.

  The evaluation of the correlation-based filtering resulted in a precision of 67%, recall of 73% and F1 score of 69%.

Overall these are very satisfactory results.

In the next chapter, these evaluation results will be discussed and compared to the results of other systems. There will also be a detailed analysis to determine and evaluate the source of errors produced by the table extension algorithms.

Chapter 5

Analysis, Evaluation and Discussion

5.1 Introduction

In the previous chapter, the evaluation results were presented. In this chapter the evaluation results will be analysed and compared to the results from other table extension algorithms to determine, whether the new table extension algorithms are indeed a viable alternative to the existing table extension algorithms.

This dissertation set out with the goal of proving or disproving the hypothesis that the new table extension algorithms are a viable alternative to the existing table extension algorithms. In order to be a viable alternative, it has to be useful and have a performance comparable to the existing systems. The five key requirements for viability were identified at the beginning of chapter 3. They are: *High usability*, *High availability*, *Wide applicability*, *Fast execution time* and *Correctness.* These key requirements have been used both in the design and the evaluation of the new table extension system. Each of these five key requirements has to be fulfilled i.e. the evaluation results for each of these key requirements has to be sufficiently good.

This chapter contains a section for each of the key requirements. In these sections the performance of the new table extension algorithms with respect to these key requirements are analysed and discussed in detail.

5.2 High Usability

For the evaluation of the usability, the method presented in Jakob Nielsen's influential 1994 paper was used. This methodology does not give a numeric threshold for 'high usability' (Generally, most usability evaluations only rely on qualitative metrics. Quantitative metrics are only used for A/B testing or the usability evaluations of modelling languages). Jakob Nielsen's methodology does however state that the users should find it efficient, learnable, memorable, hard-to-make-errors-with and satisfactory (i.e. pleasant to use).

As shown in section 4.3.1., these criteria are fulfilled for the new table extension system – the user interface (here: API) was modified until all evaluators found it efficient, hard-to-make-errors-with and satisfactory. It was later shown to be also very learnable by new evaluators and memorable for the existing evaluators, when used in combination with the API documentation.
Unfortunately, the usability of the new table extension system cannot be compared to any of the existing table extension systems as the usability had not been evaluated for them.

5.3 High Availability

In Section 4.3.2., the operational availability Ao measured to be 0.941.

What value of operational availability is considered an acceptable value varies significantly between applications – a heart rate monitor in a hospital requires a significantly higher availability than a student's master thesis. Due to this problematic, no guidelines for what constitutes an acceptable availability are stated in any of the fields that talk about software availability, such as ITIL (Zeng, 2008), reliability engineering (Zio, 2009) and Six Sigma (Bigio, Edgeman & Ferleman, 2004). Also, no comparison with the availability of the other table extension systems can be made, as none of them measured the availability of their system.

Overall, the availability was completely satisfactory for the use case. The users – in particular: RapidMiner – didn't experience major difficulties due to the system not being available.

Detailed analysis

It is also interesting to consider the reasons why the system failures/outages occurred. The most common known reason for system failures (3 out of 9) was bugs in the software system that had been introduced through the release of a new version of the table extension system. The second most common reason was for the system crashing due to being overloaded (1 out of 9). Unfortunately, for most of the system failures (5 out of 9), the reason for the failure could not be identified. In these cases, the failure might also have been caused by the system being overloaded, or it might have been caused by the server crashing or restarting due to the operating system or another program on the server.

There are several ways in which the availability could be increased:

- Monitoring system
  The Mean Time To Recover (MTTR) was 24.11 hours, which is fairly high. This made a significant negative impact on the operational availability. The reason the MTTR was so high, is that often the system failure remained unnoticed for some time – on the 20th May and 30th June 2018, the system failure remained unnoticed for almost 3 days!
  A monitoring system that would automatically send an email or text message when system failure occurred, would greatly reduce the Mean Time To Recover and thus greatly improve the availability.

- Backup redundancy

  An even bigger step towards improving the operational availability could be done by setting up a second server with the same table extension system. In this case a load balancer would be used to direct requests to either of the two table extension systems. If one system fails, all requests will be directed to the other system and no outage will occur. The probability that both table extension systems fail simultaneously is very low, therefore the operational availability would be greatly improved. However, this setup is also more complicated. For instance, the servers would need access to the same repositories.

5.4 Wide Applicability

As mentioned in section 3.3.3, wide applicability is intrinsically guaranteed by the new table extension system, as users can create custom repositories for any desired application domain.

The examples used to evaluate correctness are from widely different subject areas and demonstrate this wide applicability. However, they also show that the correctness of the results depends on the repository – and this is the responsibility of the user. In section 5.6 the factors influencing the correctness of the new table extension are analysed in detail.

5.5 Fast Execution Time

The execution time for the table extension operations is the most critical, as these are the most frequently used operations. For the unconstrained table extension, the median execution time was observed be 4.39 seconds (with individual times ranging between 2.5 and 8.8 seconds). For the correlation-based table extension it was 5.05 seconds (with individual times ranging between 3.0 and 9.0 seconds) – see Table 5.1. As previously mentioned, the correlation-based table extension takes slightly longer, because it combines the unconstrained table extension operation with additional correlation-based filtering. The additional time for the correlation-based table extension is therefore the time that the correlation-based filtering takes (note that due to load variations some results seem to be inverted).

4.39 and 5.05 seconds seem like a long time if you consider the research published by Google, which shows that 53% of mobile website visitors will leave the page if it does not load within three seconds[1]. Or that a webpage will have a significant advantage over competing pages, if its response time is 250 milliseconds faster[2].

Automatic table extension is however not a comparable task to loading a website. Users of this service are aware that they are saving many hours, even days, of manual work by using this innovative service. Also, no faster alternatives are known to exist – unfortunately, the execution time was not evaluated for the existing table extension systems. In the usability evaluations (see 4.3.2 and 5.2.1), the evaluators perceived the execution time as perfectly acceptable.

Detailed analysis

It is interesting to analyse the reasons for the variation in the execution times for extending different tables. To analyse this question in detail, the unconstrained table extension operation was modified to save logs between every step of the execution; logging the current time as well as the current state of the system (specifically how many tables are being processed). Table 5.1 shows the execution times for the different extended tables, as well as the number of tables being processed in the time intensive steps 3 (instance matching) and step 4 (schema matching). The instance- and schema- matching algorithms are computationally complex, as they make very many comparisons between individual data-points in order to accurately

[1]`https://www.marketingdive.com/news/google-53-of-mobile-users-abandon-sites-that-take-over-3-seconds-to-load/426070/`

[2]`https://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html`

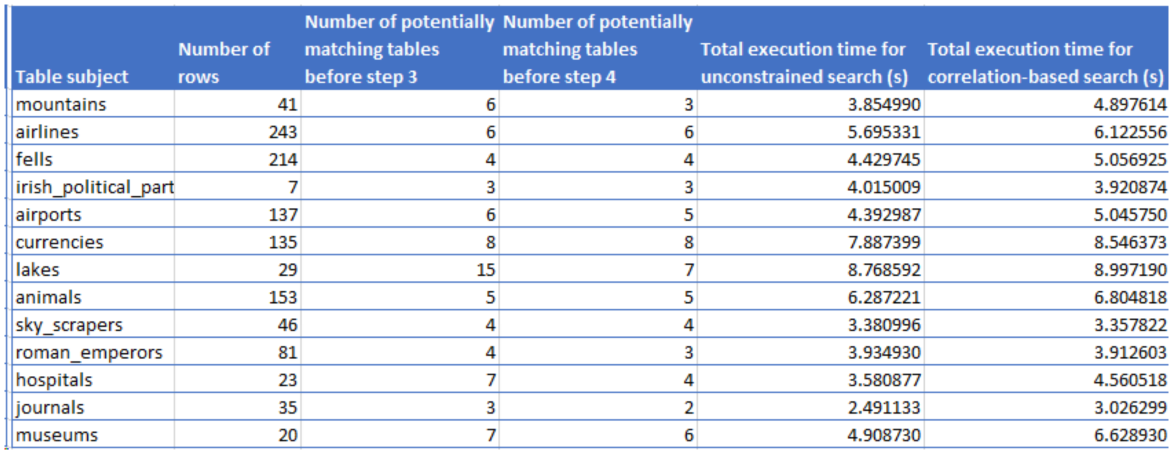


Table subject	Number of rows	Number of potentially matching tables before step 3	Number of potentially matching tables before step 4	Total execution time for unconstrained search (s)	Total execution time for correlation-based search (s)
mountains	41	6	3	3.854990	4.897614
airlines	243	6	6	5.695331	6.122556
fells	214	4	4	4.429745	5.056925
irish_political_part	7	3	3	4.015009	3.920874
airports	137	6	5	4.392987	5.045750
currencies	135	8	8	7.887399	8.546373
lakes	29	15	7	8.768592	8.997190
animals	153	5	5	6.287221	6.804818
sky_scrapers	46	4	4	3.380996	3.357822
roman_emperors	81	4	3	3.934930	3.912603
hospitals	23	7	4	3.580877	4.560518
journals	35	3	2	2.491133	3.026299
museums	20	7	6	4.908730	6.628930

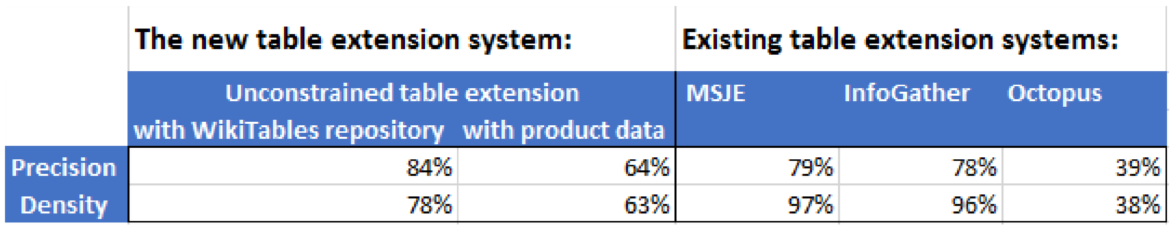
Table 5.1: Execution times for unconstrained and correlation-based table extension

detect instance and schema correspondences between tables. For an overview of the whole table extension algorithm, please refer to section 3.2.4.

Table 5.1 shows a strong correlation between the number of potentially matching tables being processed in steps 3 & 4 and the overall execution time (Pearson correlation of 0.79 and 0.88, respectively). A linear regression model with these two columns as explanatory variables explains 83% of the variation of the execution times for the unconstrained table extension – this is statistically highly significant with a p-value for the model of 0.01%. Due to the correlation between the numbers of tables processed in steps 3 and 4 (r=0.71), the separation of these two effects is difficult. In contrast, the number of rows has no significant effect on the execution times.

The number of tables processed in step 3 depends on how many tables in the repository were found in step 2 to have a similar subject column to the to-be-extended table. In step 2 more tables are found if the repository contains many tables on this particular subject or if the names in the subject column of the to-be-extended table are so general, that it matches many unrelated subject columns.

In step 3 then, the number of potentially matching tables is reduced, as tables that have no instance matches to the to-be-extended table are removed. This excludes



	The new table extension system:		Existing table extension systems:		
	Unconstrained table extension		MSJE	InfoGather	Octopus
	with WikiTables repository	with product data			
Precision	84%	64%	79%	78%	39%
Density	78%	63%	97%	96%	38%

Table 5.2: Correctness scores for all table extension systems

the tables that had been falsely detected in step 2.

5.6 Correctness

Table 5.2 shows the correctness score obtained from the evaluation of the unconstrained table extension (see section 4.3.5.), as well as the correctness scores obtained by the all the existing table extension systems: MSJE (Lehmberg et al., 2015), InfoGather (Yakout, Ganjam, Chakrabarti & Chaudhuri, 2012) and Octopus (Cafarella, Halevy & Khoussainova, 2009).

Two evaluations were performed on the new unconstrained table extension algorithm: the first evaluation with the WikiTables repository had exactly the same setup as MSJE's evaluation (using the same ground truth tables and the same repository with 460 513 tables from Wikipedia – see section 4.3.5.1.1), the second evaluation (with product data) uses a repository with product-specification tables to extend product data tables (see section 4.3.5.1.2.). Looking at Table 5.2, it is obvious, that better results were achieved in the first evaluation.

The main reason why the second evaluation with the product data performs worse is that the repository used for this evaluation only contains 31 product-data tables; the WikiTables repository instead contains of 460 513 tables. This makes a very significant difference to the performance – the redundancy in the WikiTables repository allows for column values to be populated even if individual values in the repository were missed and for wrong values to be removed when fused with correct values.

The second reason for the discrepancy between the two evaluations is that the product-data evaluation was a small evaluation in a specific topic and therefore very sensitive to the specific form and quality of the data. In depth analysis (debugging) resulted in the insight that the names of the different products are sometimes so similar, that the table extension algorithm confuses them. E.g. “samsung galaxy s6” has a greater string similarity to “samsung un60js7000f” (a TV) than to “galaxy s6”.

In conclusion, the correctness very much depends on the quality of the data in the repository. It is however important to note, that given a repository with high-quality data, the table extension performs very well independent of the domain of the tables in the repository.

In the further discussions of the unconstrained table extension algorithm, the result from the first evaluation will be used, as it is the more representative – a big repository was used and many different tables were extended.

Table 5.2 shows that the first evaluation of the unconstrained table extension has a better precision than all the existing table extension systems. On the other hand, the density is lower than that of the MSJE and InfoGather. It can however be argued, that the density is less important than the precision: it is more important for users to be able to trust the values in the new table columns than for the new table columns to be completely populated.

Thus, there is justification in the claim that the new unconstrained table extension algorithm has a comparable, if not even better, performance than the existing table extension algorithms.

The correlation-based table extension produces exactly the same tables as the unconstrained table extension, with the only difference being that some of the new columns have been removed by the correlation-based filtering. As the correlation-based filtering is not biased towards removing especially correct or incorrectly populated columns, the columns added by the correlation-based table extension also roughly have a precision and density of 84% and 78%, respectively. The correlation-based filtering itself has a precision of 67% and recall of 73%. The overall result of the correlation-based table extension is therefore comparably useful as the result of the unconstrained table extension.

Detailed analysis

Figure 5.1 shows a diagram of the steps in the unconstrained table extension algorithm. The most critical steps of this algorithm are the steps 2, 3 and 4. Difficult and complex procedures are executed in these steps that try to gain insights from the very heterogeneous tables that the algorithm has to work with.

It is essential for the overall correctness of the unconstrained table extension algorithm, that the insights obtained in the steps 2, 3 and 4 are as correct as possible. Therefore, individual evaluations of each of these steps have been performed – the details of these evaluations are described in Appendix E. The results of these individual evaluations of the important algorithm-parts are the following:

- Step 2, finding of tables with similar subject columns:
  Pair completeness = 35%; reduction ratio = 99.994%.
  Pair completeness is the percentage of tables that were found to be potentially matching by the blocking step (step 2), that are indeed true matches. Reduction ratio is the percentage of tables that are no longer considered potentially matching after the blocking step. Please refer to Appendix E.1 for more information.

- Step 3, instance matching:
  Precision = 0.949; Recall = 0.624; F1 = 0.753.

- Step 4, schema matching:

Precision = 0.803; Recall = 0.712; F1 = 0.755.

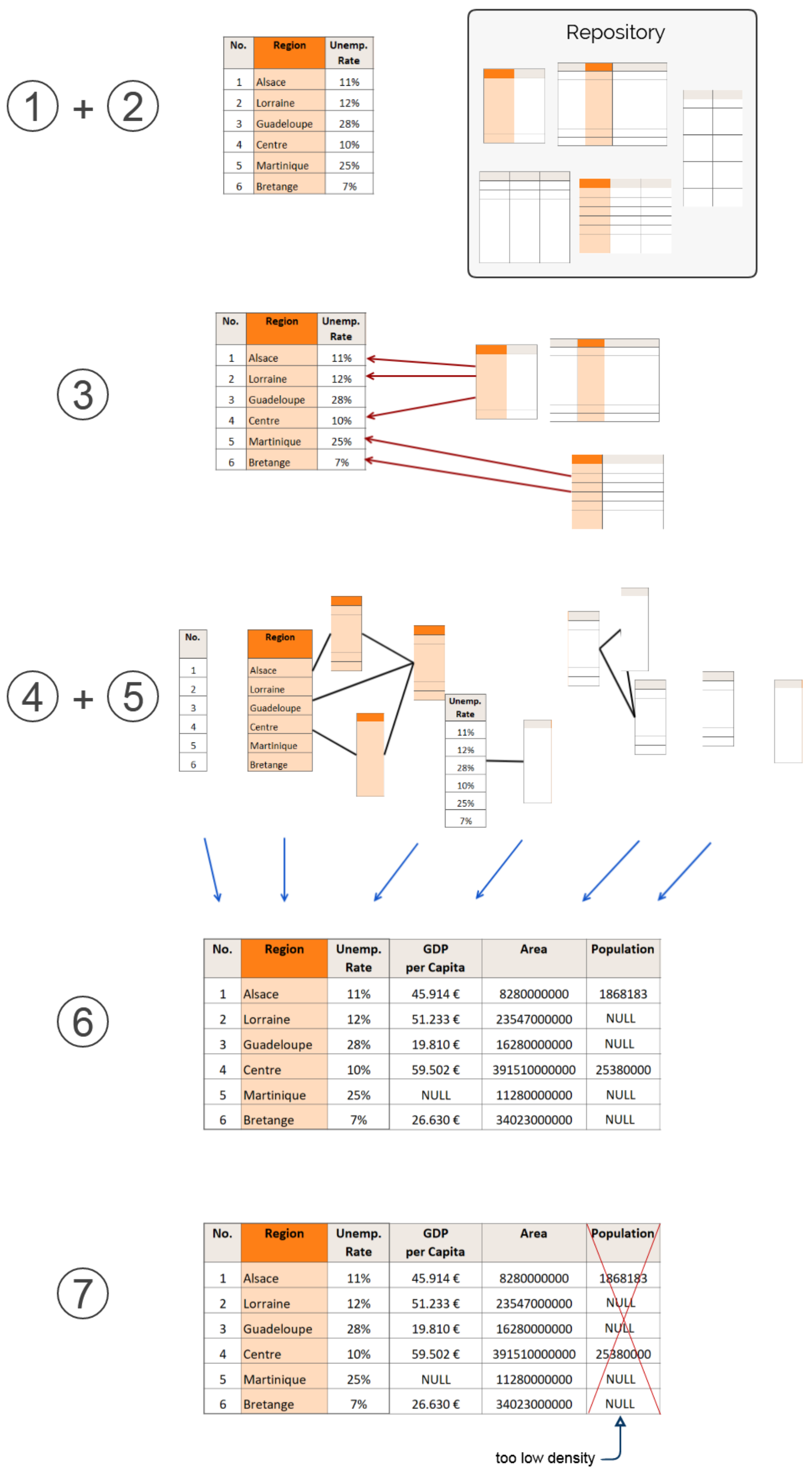
These results mean the following:
In step 2 of the unconstrained table extension algorithm, an average of 25 potentially matching tables are retrieved from the repository, of which only about 35% are truly matching tables. Fortunately, many of the non-matching tables are filtered out in step 3, because they have no instance correspondences with the to-be-extended table: in step 3, 62% of the truly existing instance correspondences between the potentially matching tables and the to-be-extended table are found; the correspondences found are very reliable – 95% of them are correct; therefore, in step 3 the number of wrongly matching tables can be reduced from 65% to 25%.

In step 4, the schema matches between all the tables (the potentially matching tables and the to-be-extended table) are identified. Unfortunately, only 80% of the identified schema matches are correct, leading to some wrong columns in the schema clusters produced in step 5. Also, only 71% of the existing schema matches were found, leading to too many schema clusters – columns that belong to a cluster are not joined to this cluster and instead form a new cluster.

Finally, in steps 6 & 7 the results are improved a bit. In step 6, the clusters are fused into a single column. The similarity-based voting mechanism used here helps to improve the result, as it makes it more likely that the values from the correct columns are chosen (for more information refer to section 3.2.4.). In step 7, columns with a density less than 60% are removed.

In the extended tables that are returned by the unconstrained table extension, the new columns have an average density of 78% and a precision 84% i.e. 84% of the column values are correct.

In-depth analysis of individual runs of the table execution algorithm (debugging) have confirmed that the errors in the final output are the result of small errors



No.	Region	Unemp. Rate
1	Alsace	11%
2	Lorraine	12%
3	Guadeloupe	28%
4	Centre	10%
5	Martinique	25%
6	Bretange	7%

No.	Region	Unemp. Rate	GDP per Capita	Area	Population
1	Alsace	11%	45.914 €	8280000000	1868183
2	Lorraine	12%	51.233 €	23547000000	NULL
3	Guadeloupe	28%	19.810 €	16280000000	NULL
4	Centre	10%	59.502 €	391510000000	25380000
5	Martinique	25%	NULL	11280000000	NULL
6	Bretange	7%	26.630 €	34023000000	NULL

No.	Region	Unemp. Rate	GDP per Capita	Area	~~Population~~
1	Alsace	11%	45.914 €	8280000000	~~1868183~~
2	Lorraine	12%	51.233 €	23547000000	~~NULL~~
3	Guadeloupe	28%	19.810 €	16280000000	~~NULL~~
4	Centre	10%	59.502 €	391510000000	~~25380000~~
5	Martinique	25%	NULL	11280000000	~~NULL~~
6	Bretange	7%	26.630 €	34023000000	~~NULL~~

Figure 5.1: Diagram of the execution steps in the unconstrained table extension

			Scores from unconstrained table extension	
Extended Table	Number of entities	Extended column	Precision	Density
Books	100	Author	74%	100%
Companies	50	Headquarter	100%	55%
		Industry	100%	87%
Countries	201	Currency	99%	96%
		Population	89%	94%
		Area	97%	93%
		Capital	97%	92%
		Code	71%	85%
Drugs	100	Ingredients	73%	66%
Films	100	Cast	27%	91%
		Director	99%	93%
		Genre	95%	67%
		Year	73%	44%
Songs	100	Artist	94%	90%
Soccer Players	100	Team	65%	16%
Average scores			84%	78%

Table 5.3: Precision and density scores for individual table extension operations

in many steps of execution. These small errors are due to the huge difficulty of the problem: finding the right tables from amongst half a million heterogeneous tables and joining tables that have completely different formats and structures and imperfect data quality.

As can be seen in Table 5.3, the correctness varies for the different tables. The main reason for this was found to be the varying quality and quantity of tables in the repository for the different topics. In the cases where there were tables in the repository that exactly aligned with the to-be-extended table, very good results were achieved. In the cases where the tables from the repository had a different format and bad data quality, the new columns were also bad.

5.7 Conclusion

In this chapter the performance of the new table extension system was discussed with respect to each of the five key requirements.
It was shown that all five requirements are fulfilled by the system. The hypothesis that the new table extension algorithms are a viable alternative to the existing table extension systems has therefore been proven.

In addition, the table extension algorithms were analysed in detail to give further insights into the system and answer questions such as: ‘What causes the errors?’, ‘How do the individual steps of the algorithm affect the different performance metrics?’ or, ‘How could the algorithms be improved?’.

The next chapter will go deeper into the possible improvements and additions that could be done to the new table extension system.

Chapter 6

Conclusion

6.1 Research Overview

The amount of freely available data has increased by many orders of magnitude in recent decades. Companies can extract great value from this open data by harnessing it to gain more insights and improve their decision making or by improving their data-based tools. To quote Halevy, Rajaraman & Ordille (2006): "Data integration is crucial in large enterprises that own a multitude of data sources, for progress in large-scale scientific projects, where data sets are being produced independently by multiple researchers, for better cooperation among government agencies, each with their own data sources, and in offering good search quality across the millions of structured data sources on the World-Wide Web."

Finding the right data in data portals and integrating it with the existing company data is however a very time-consuming and arduous task. The use of open data has therefore remained much behind the original expectations (Janssen, Charalabidis & Zuiderwijk, 2012).

In order to address this challenge, automatic table extension algorithms have been developed which automate the process of data finding and data integration. The automatic table extension algorithms add additional columns with additional

information to any table, by finding and matching the required data from a repository containing thousands of open data tables.
The existing table extension algorithms work quite well. However, they require the user to enter keywords for finding additional columns. The problem with this is that the user will not always know the right keywords to use – the quality of the retrieved columns is very dependent on the correct keywords being chosen and the user often does not know about all the columns that could be added to the table.

In this dissertation two new table extension algorithms are presented and evaluated which no longer require the user to enter keywords – the unconstrained- and the correlation-based- table extension. The unconstrained table extension automatically extends a table by adding all possible columns to the table, instead of only adding individual columns as the existing table extension systems do. The correlation-based table extension adds all columns to the table that correlate with a specified attribute/column of the original table.

6.2 Problem Definition

This dissertation investigates the hypothesis that the two new table extension algorithms are a viable alternative to the existing table extension algorithms.
In chapter 3, five key requirements for viable software solutions were identified: *High usability*, *High availability*, *Wide applicability*, *Fast execution time* and *Correctness.*
This dissertation attempts to prove or disprove the hypothesis by evaluating the new table extension algorithms with respect to the five key requirements and by comparing them with the existing table extension algorithms.

6.3 Design/Experimentation, Evaluation & Results

To evaluate the new table extension algorithms, they were implemented in a new table extension system. The evaluations of the new table extension algorithms produced the following results:

- High usability

  The usability of the new table extension system was evaluated after every iteration by four external evaluators. The feedback from the evaluators was used to continuously improve the usability so that towards the end there were no more complaints. The new table extension system was shown to be efficient, learnable, memorable, hard-to-make-errors-with and satisfactory.

- High availability

  The availability of the new table extension system was evaluated over a five month period. During this time, the new table extension system was continuously running and being used. All outages that occurred during the five month period were tracked and documented. From this documentation the operational availability was calculated to be 0.941.

- Wide applicability

  Tables can only be extended if the data for the new columns is in the repository. However, the table extension algorithms may be used with different repositories of tables. A wide applicability of the table extension algorithms is guaranteed, by users being given the possibility to create their own repositories. The table extension algorithms were shown to have a similar performance when used with different repositories.

- Fast execution time

  To evaluate the execution time, many different tables were extended with both of the table extension algorithms. In order to gain an estimate for the maximum execution time, a very large repository with 460 thousand tables was used. The median execution time for the unconstrained table extension was measured to be 4.39 seconds and 5.05 seconds for the correlation-based table extension.

- Correctness

  The evaluation of the correctness was the most comprehensive, because the correctness is the most critical requirement and because it is the only requirement which was measured by the other table extension systems. Two different evaluations of the unconstrained table extension were performed and one of the correlation-based table extension.

  For the first evaluation of the unconstrained table extension, the data and methodology completely matched the evaluation of the Mannheim Search Join Engine (Lehmberg et al., 2015); here an average precision and density of 78% and 96% were achieved. The second evaluation was performed with product data and achieved an average precision and density of 64% and 63%.

  The evaluation of the correlation-based filtering resulted in a precision of 67%, recall of 73% and F1 score of 69%.

In addition to the evaluations, in-depth analyses were performed to answer questions such as 'What causes the errors?', 'How do the individual steps of the algorithm affect the different performance metrics?' or, 'How could the algorithms be improved?'.

6.4 Contributions and Impact

The evaluations showed that the two new table extension algorithms are indeed a viable alternative to the existing table extension algorithms. Creating an entirely new approach to automatic table extension: keyword-free table extension.

The extensive evaluation and analysis of the table extension algorithms provides insight into the strengths and weaknesses of different approaches and suggests possible improvements. This will help future researchers build upon the knowledge gained to build even better table extension algorithms. This dissertation also helps readers to get a deep understanding of the field of data integration from the comprehensive overview and history of this field in chapter 2. Furthermore, they get a detailed understanding of table extension algorithms from the explanation of the existing and the new algorithms.

Overall, automatic table extension is a very exciting and promising field of research. This research and the research of other table extension algorithms shows that good results can be obtained and there are very many interesting problems to be solved.

It is however not only interesting for research. Due to the collaboration between this research project and the company RapidMiner, the new table extension algorithms were integrated into the commercial data analysis tool RapidMiner Studio. This not only proves the commercial applicability of this new technology, but also exposes this at present rarely used technology to a very wide audience.

More exposure was achieved by publishing a paper on the unconstrained- and correlation-based- table extension algorithms[1] at the LWDA2018 conference[2].

6.5 Future Work & Recommendations

The most critical aspect of any table extension algorithm is the correctness of the new columns that are added to the table. There are many different approaches that future research on automatic table extension could take to further improve the correctness of the algorithms.

The correctness of the extended table greatly depends on the data quality of the tables in the repository. The results of the table extension can only be as good as the data that is used for it – if the repository contains wrong data, then inevitably the

[1] `https://dws.informatik.uni-mannheim.de/fileadmin/lehrstuehle/ki/pub/KleppmannBizer-DensityAndCorrelationBasedTableExtension-LWDA2018.pdf`

[2] `http://ceur-ws.org/Vol-2191/`

result is also wrong. The correctness of the table extension will therefore be greatly enhanced by filtering out bad data. For this, different approaches could be taken. For instance, when uploading a table to a repository, the user could specify the degree to which she/he trusts the correctness of the table. This trust metric can then be used for the fusion (step 6 of the unconstrained table extension algorithm – see section 3.2.4): if multiple data points were found for a specific position in a new data column, then the current strategy is to choose the data point that is most similar to the others. For a trust-based fusion, instead the data point from the most trusted table would be chosen.

Instead of requiring the trust of a table to be specified when it's uploaded to a repository, it could be determined through reinforcement learning. One could imagine a tool that displays the fully extended table and allows the user to remove individual values, which the user thinks are wrong. The information of which values were removed is then used to determine the trust of the tables: the algorithm remembers from which tables the removed values came, the trust of these tables is then reduced.

There is another common problem with the tables in repositories that greatly affects the correctness of the table extension: Frequently, column names are not sufficiently specific and understandable. With such tables it is often not possible to find out what information the table contains by just looking at it. For these tables, the text surrounding the table might help to categorize it. This is true for humans, but also for the algorithms. Allowing for keyword-based search on the surrounding text by including it in the search-index could greatly help finding more relevant tables.

Further improvements could be achieved by pre-processing the tables in the repository. In addition to indexing the tables in the repository, schema correspondences between these tables could be calculated. These pre-calculated schema correspondences could be used for indirectly finding matching tables. These are tables that were not found by the initial search using the SubjectColumnIndex, but have many schema correspondences to the tables found by the initial search. By extending the initial set of potentially matching tables with the indirectly matching tables, the algorithm has a greater chance of finding all the relevant data that is in the repository and producing a better result.

Another approach for improving the correctness of table extension algorithms is by making domain-specific table extension algorithms. By specializing on a specific domain, many domain-specific improvements can be made, for instance using domain-specific data types and similarity metrics. Current table extension systems only distinguish between numeric and categorical columns (and sometimes dates), a table extension system focused on product-data could however also distinguish e.g. prices and quality-ratings and use specific similarity metrics for these columns. A domain-specific table extension system could also use custom mappings and dictionaries. For example, a tree of the product-type-hierarchy would greatly help finding the correct product data from the repository.

During the five months the new table extension system was running, the operational availability was measured to be 0.941. This corresponds to a downtime of 20 days per year, which is an acceptable downtime for a master dissertation, but not for a commercial application. As mentioned in section 5.3, there are various ways in which the operational availability of the system could be improved. For instance, a monitoring system that detects outages and sends warning text-messages could assure that outages are detected and fixed earlier, thereby reducing the system's downtime. The number of outages could be reduced drastically by building in redundancy: for example there could be two systems running on two different servers and if one fails, a load balancer could direct the traffic to the other.

The evaluation of the table extension systems could be improved by running the evaluation on more different repositories. Also, evaluating the table extension of more different tables will help produce more reliable performance estimates.

Finally, with small alterations to the table extension algorithms presented in this dissertation, completely new applications could be created:

For instance, a tool for *data verification* could be implemented. Instead of adding new columns to a table, it recreates the existing columns of the table from a repository. Values which are different in the recreated columns are potentially wrong values in the original table: Therefore, the data verification highlights them.

Very similar to the data verification tool is an *auto-fill tool*. This auto-fill tool tries to populate the missing values in any table. This also works by recreating the existing columns and transferring the values from the recreations whenever a value in the original column is missing.

Finally, another completely new tool that could be implemented based on the table extension algorithms is a *keyword-based table generation tool*. This tool is able to generate a completely new table from only some keywords. In the first step it searches for tables with subject-column headers that match the keywords. In the next step it clusters the tables into groups of tables with many instance-matches. The biggest group is likely to contain the desired instances, the tables in this group are then matched and fused into one table.

References

Abiteboul, S., & Duschka, O. M. (1998). Complexity of answering queries using materialized views. Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '98. doi:10.1145/275487.275516

Alexe, B., Ten Cate, B., Kolaitis, P. G., & Tan, W. (2011). Designing and refining schema mappings via data examples. Proceedings of the 2011 international conference on Management of data - SIGMOD '11. doi:10.1145/1989323.1989338

Andreasen, M. S., Nielsen, H. V., Schrøder, S. O., & Stage, J. (2007). What happened to remote usability testing? Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07. doi:10.1145/1240624.1240838

Application guide for ISO 10755, ISO 10756, ISO 10757, ISO 10758 and ISO 10759. (n.d.). doi:10.3403/00457450u

Balakrishnan, S., Halevy, A. Y., Harb, B., Lee, H., Madhavan, J., Rostamizadeh, A., & Shen, W. (2015). Applying webtables in practice. CIDR.

Bernstein, P. A. (2003). Applying Model Management to Classical Meta Data Problems. In Proceedings of the Conference on Innovative Data Systems Research (CIDR).

Bhagavatula, C. S., Noraset, T., & Downey, D. (2013). Methods for exploring and mining tables on wikipedia. Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics, IDEA '13, ACM, New York, NY, USA, 18–26.

Bigio, D., Edgeman, R. L., & Ferleman, T. (2004). Six sigma availability management of information technology in the office of the chief technology officer of Washington, DC. Total Quality Management & Business Excellence, 15(5-6), 679-687.

Braunschweig, K., Eberius, J., Thiele, M. & Lehner, W. (2012). The state of open data: limits of current open data platforms, paper presented at WWW2012 , Lyon, France, 16 - 20 Apr .

Brodie, M. L. (2010). Data Integration at Scale: From Relational Data Integration to Information Ecosystems. 2010 24th IEEE International Conference on Advanced Information Networking and Applications. doi:10.1109/aina.2010.184

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2013). Data complexity of query answering in description logics. Artificial Intelligence, 195, 335-360. doi:10.1016/j.artint.2012.10.003

Catarci, T., & Lenzerini, M. (1993). Representing and using interschema knowledge in cooperative information systems. International Journal of Cooperative Information Systems, 02(04), 375-398. doi:10.1142/s0218215793000174

Chakrabarti, K., Chaudhuri, S., Chen, Z., Ganjam, K., & He, Y. (2016). Data services leveraging bing's data assets. IEEE Data Eng. Bull, 39(3), 15–28.

Cochrane, C. B., & Hagan, G. J. (1998). Glossary of Defense Acquisition Acronyms and Terms. Ninth Edition. doi:10.21236/ada359250

Doan, A., Halevy, A., & Ives, Z. (2012). The Future of Data Integration. Principles of Data Integration, 453-457. doi:10.1016/b978-0-12-416044-6.00019-3

Dong, X. L., & Srivastava, D. (2013). Big data integration. Proceedings of the IEEE International Conference on Data Engineering, 1245–1248.

Elsayed, T., Lin, J., & Oard, D. W. (2008). Pairwise document similarity in large collections with MapReduce. Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies Short Papers - HLT '08. doi:10.3115/1557690.1557767

Fagin, R., Kolaitis, P. G., Popa, L., & Tan, W. (2010). Schema Mapping Evolution Through Composition and Inversion. Schema Matching and Mapping, 191-222. doi:10.1007/978-3-642-16518-4_7

Fagin, R., Kolaitis, P. G., Popa, L., & Tan, W. C. (2004). Composing schema mappings: Second-order dependencies to the rescue. Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '04, 83-94. doi:10.1145/1055558.1055572

Florescu, D., Levy, A., Manolescu, I., & Suciu, D. (1999). Query optimization in the presence of limited access patterns. Proceedings of the 1999 ACM SIGMOD international conference on Management of data - SIGMOD '99. doi:10.1145/304182.304210

Friedman, M., Levy, A., & Millstein, T. (1999). Navigational plans for data integration. In Proceedings of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99), 67– 73.

Gentile, A. L., Ristoski, P., Eckel, S., Ritze, D., & Paulheim, H. (2017). Entity Matching on Web Tables: a Table Embeddings approach for Blocking. In EDBT (pp. 510-513).

Golshan, B., Halevy, A. Y., Mihaila, G. A., & Tan, W. (2017). Data integration: After the teenage years. PODS.

Gupta, R., Halevy, A., Wang, X., Whang, S. E., & Wu, F. (2014). Biperpedia: An ontology for search applications. Proceedings of the VLDB Endowment, 7(7), 505-516. doi:10.14778/2732286.2732288

Gupta, S., & Giri, V. (2018). Data lake ingestion strategies. Practical Enterprise Data Lake Insights, 33-85. doi:10.1007/978-1-4842-3522-5_2

Halevy, A. Y., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., ... Sikka, V. (2005). Enterprise information integration: successes, challenges and controversies. Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05. doi:10.1145/1066157.1066246

Halevy, A. Y., Korn, F., Noy, N. F., Olston, C., Polyzotis, N., Roy, S., & Whang, S. E. (2016). Managing google's data lake: an overview of the goods system. IEEE Data Eng. Bull, 39(3), 5–14.

Halevy, A., Rajaraman, A., & Ordille, J. (2006). Data integration: The teenage years. VLDB, 9-16.

Halevy, A., Rajaraman, A., & Ordille, J. (2006). Data integration: The teenage years. VLDB, 9-16.

Hoover, J. (2017). Open Data in Research. doi:10.14293/s2199-1006.1.sor-edu.clju5m5.v1

Huijboom, N., & Van den Broek, T. (2011). Open data: an international comparison of strategies. European Journal of ePractice, 12, 4-16.

Ives, Z. G., Green, T. J., Karvounarakis, G., Taylor, N. E., Tannen, V., Talukdar, P. P., ... Pereira, F. (2008). The ORCHESTRA Collaborative Data Sharing System. ACM SIGMOD Record, 37(3), 26. doi:10.1145/1462571.1462577

Janssen, M., Charalabidis, Y., & Zuiderwijk, A. (2012). Benefits, Adoption Barriers and Myths of Open Data and Open Government. Information Systems Management, 29(4), 258-268. doi:10.1080/10580530.2012.716740

Kang, K. H., & Kang, J. (2009). How do firms source external knowledge for innovation? Analysing effects of different knowledge sourcing methods. International Journal of Innovation Management, 13(01), 1-17, doi:10.1142/s1363919609002194

Kleppmann, B., Bizer, C., Yaqub, E., Temme, F., Schlunder, P., Arnu, D., & Klingenberg, R. (2018). Density- and Correlation-based Table Extension. Lernen. Wissen. Daten. Analysen. (LWDA 2018).

Kramer, D. (1999). API documentation from source code comments: a case study of Javadoc. Proceedings of the 17th annual international conference on Computer documentation - SIGDOC '99. doi:10.1145/318372.318577

Lehmberg, O., Brinkmann, A., & Bizer, C. (2017). WInte.r - A Web Data Integration Framework. CEUR workshop.

Lenzerini, M. (2002). Data integration: A Theoretical Perspective. Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '02. doi:10.1145/543613.543644

Levandowsky, M., Winter, D. (1971). Distance between sets. Nature, 234(5), 34-35. doi:10.1038/234034a0

Levy, A. Y. (2000). Logic-Based Techniques in Data Integration. Logic-Based Artificial Intelligence, 575-595. doi:10.1007/978-1-4615-1567-$8_2$4

Levy, A. Y., Rajaraman, A., & Ordille, J. J. (1996). Querying Heterogeneous Information Sources Using Source Descriptions. In Proceedings of the International Conference on Very Large Databases (VLDB).

Levy, A. Y., Rajaraman, A., & Ullman, J. D. (1996). Answering queries using limited external query processors. Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '96. doi:10.1145/237661.237716

Madhavan, J., & Halevy, A. Y. (2003). Composing Mappings Among Data Sources. Proceedings 2003 VLDB Conference, 572-583. doi:10.1016/b978-012722442-8/50057-4

Madhavan, J., Jeffery, S., Cohen, S., Dong, X., Ko, D., Yu, C., & Halevy, A. (2007). Web-scale Data Integration: You can only afford to Pay As You Go. CIDR.

Mahmoud, H. A., & Aboulnaga, A. (2010). Schema clustering and retrieval for multi-domain pay-as-you-go data integration systems. Proceedings of the 2010 international conference on Management of data - SIGMOD '10.

doi:10.1145/1807167.1807213

McCue, C. (2007). Data Mining and Predictive Analysis: Intelligence Gathering and Crime Analysis, Butterworth-Heinemann, Burlington

Melnik, S., Rahm, E., & Bernstein, P. A. (2003). Rondo: A programming platform for generic model management. Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03. doi:10.1145/872757.872782

Nielsen, J. (1994). Usability inspection methods. Conference companion on Human factors in computing systems - CHI '94. doi:10.1145/259963.260531

Nin Guerrero, J., Muntés Mulero, V., Martínez Bazán, N., & Larriba Pey, J. (2007). On the use of semantic blocking techniques for data cleansing and integration. In 11th International Database Engineering and Applications Symposium (IDEAS'07) (pp. 190-198). IEEE Computer Society.

Petrovski, P., Bryl, V., & Bizer, C. (2014). Learning Regular Expressions for the Extraction of Product Attributes from E-commerce Microdata. Proceedings of Linked Data for Information Extraction Workshop (LD4IE 2014) ISWC 2014, 45–54.

Ritze, D., Lehmberg, O., & Bizer, C. (2015). Matching HTML Tables to DBpedia. Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics - WIMS '15. doi:10.1145/2797115.2797118

Ritze, D., Lehmberg, O., Oulabi, Y., & Bizer, C. (2015). Profiling the Potential of Web Tables for Augmenting Cross-domain Knowledge Bases. Proceedings of the 25th International Conference on World Wide Web - WWW '16.

doi:10.1145/2872427.2883017

Rokach, L., Maimon, O. (n.d.). Clustering Methods. Data Mining and Knowledge Discovery Handbook, 321-352. doi:10.1007/0-387-25465-$x_1$5

Sohan, S. M., Anslow, C., & Maurer, F. (2015). SpyREST: Automated RESTful API Documentation Using an HTTP Proxy Server (N). 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). doi:10.1109/ase.2015.52

Tran, T., Wang, H., & Haase, P. (2009). Hermes: Data Web Search on a Pay-as-You-Go Integration Infrastructure. SSRN Electronic Journal. doi:10.2139/ssrn.3199428

Virzi, R. A. (1992). Refining the Test Phase of Usability Evaluation: How Many Subjects Is Enough? Human Factors: The Journal of the Human Factors and Ergonomics Society, 34(4), 457-468. doi:10.1177/001872089203400407

Wang, Y., & He, Y. (2017). Synthesizing Mapping Relationships Using Table Corpus. Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17. doi:10.1145/3035918.3064010

Xu, L., & Embley, D. (2004). Combining the Best of Global-as-View and Local-as-View for Data Integration. ISTA, 123–136.

Zeng, J. (2008). A case study on applying ITIL availability management best practice. Contemporary Management Research, 4(4).

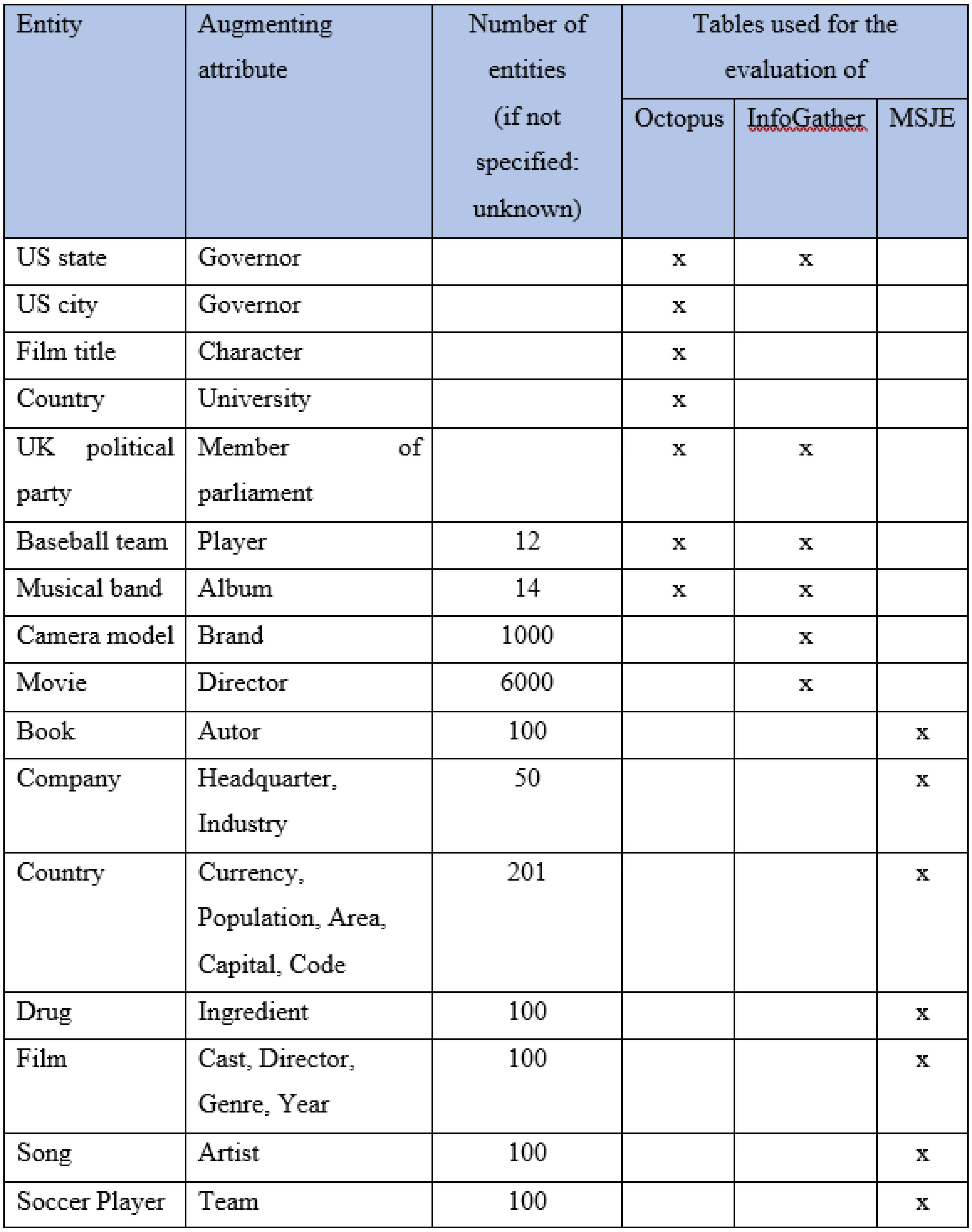
Zhang, S., Zhang, C., & Yang, Q. (2003). Data preparation for data mining. Applied Artificial Intelligence, 17(5-6), 375-381. doi:10.1080/713827180

Zio, E. (2009). Reliability engineering: Old problems and new challenges. Reliability Engineering & System Safety, 94(2), 125-141.

Appendix A

Evaluation tables for existing algorithms

Tables used for the evaluation of the existing keyword-based table extension algorithms: Octopus, InfoGather and the Mannheim Search Join Engine (MSJE).



Entity	Augmenting attribute	Number of entities (if not specified: unknown)	Tables used for the evaluation of		
			Octopus	InfoGather	MSJE
US state	Governor		x	x	
US city	Governor		x		
Film title	Character		x		
Country	University		x		
UK political party	Member of parliament		x	x	
Baseball team	Player	12	x	x	
Musical band	Album	14	x	x	
Camera model	Brand	1000		x	
Movie	Director	6000		x	
Book	Autor	100			x
Company	Headquarter, Industry	50			x
Country	Currency, Population, Area, Capital, Code	201			x
Drug	Ingredient	100			x
Film	Cast, Director, Genre, Year	100			x
Song	Artist	100			x
Soccer Player	Team	100			x

Table A.1: Evaluation tables for existing algorithms

Appendix B

List of API calls

The different API calls that the new table extension system supports:

URL-ending	**Command type**	**description**
/getRepositoryNames	get	This API call will return a list of all repository names.
/getRepositoryStatistics	get	This API call will return some statistics for the specified API.
/getUploadStatus	post	This API call will return the status of the specified bulk upload process.
/suggestAttributes	get	This API call will return a list of suggested attributes.
/createRepository	post	With this API call you can create new repositories. You just need to submit the Repository name in the url parameter ‘repository’.
/uploadTable	post	With this API call you can upload a table to a repository.
/bulkUploadTables	post	With this API call you can upload multiple tables to a repository.

/deleteRepository	post	This API call allows you to delete an entire repository. The repository name has to be specified as url parameter. To delete the repository, the user either has to submit the admin-password in the body of the call, or the IP-adress of the user has to match the IP-adress of the repository creator.
/search	post	For this REST-call you need to submit a table (‘query table’) as well as the name of the additional column you would like to have in this table (‘extension attribute’) – see example below. The web service has several repositories of tables. It searches one of these repositories for tables that contain suitable data for building the extensionAttribute column. The web service then returns the names of the tables as well as the instance- and schema correspondences (which are needed) for building the extensionAttribute column. Which repository of tables is searched has to be specified with the URL parameter repositoryName=xxx .

/extendedSearch	post	This API call works exactly the same as the /search function. The difference is that the web service uses correspondence-based search to enhance the results of the existing keyword-based search.
/unconstrainedSearch	post	In the other search functions, the data for only one extensionAttribute is returned. Here the data for ALL extension attributes is returned. That is, all extensionAttributes where the percentage of fields that can be populated ('density') is above a certain threshold.
/correlationBasedSearch	post	The correlation-based Search then extends the table with multiple new columns, all of which correlate to the correlationAttribute (which you specify).
/fetchTable	get	The search functions don't return the actual table data. Instead, they return the table names and the relevant instance- and schema- matches. To create the extensionAttribute columns, the front end also has to call this fetchTable function for each of the tables it needs the data for.

/fetchTablePOST	post	The search functions don't return the actual table data. Instead, they return the table names and the relevant instance- and schema- matches. To create the extensionAttribute columns, the front end also has to call this fetchTable function for each of the tables it needs the data for.
/getRepositoryNames	get	This API call will return a list of all repository names.
/getRepositoryStatistics	get	This API call will return some statistics for the specified API.
/getUploadStatus	post	This API call will return the status of the specified bulk upload process.
/suggestAttributes	get	This API call will return a list of suggested attributes.
/createRepository	post	With this API call you can create new repositories. You just need to submit the Repository name in the url parameter 'repository'.
/uploadTable	post	With this API call you can upload a table to a repository.
/bulkUploadTables	post	With this API call you can upload multiple tables to a repository.

/deleteRepository	post	This API call allows you to delete an entire repository. The repository name has to be specified as url parameter. To delete the repository, the user either has to submit the admin-password in the body of the call, or the IP-adress of the user has to match the IP-adress of the repository creator.
/search	post	For this REST-call you need to submit a table ('query table') as well as the name of the additional column you would like to have in this table ('extension attribute') – see example below. The web service has several repositories of tables. It searches one of these repositories for tables that contain suitable data for building the extensionAttribute column. The web service then returns the names of the tables as well as the instance- and schema correspondences (which are needed) for building the extensionAttribute column. Which repository of tables is searched has to be specified with the url parameter repositoryName=xxx .

/extendedSearch	post	This API call works exactly the same as the /search function. The difference is that the web service uses correspondence-based search to enhance the results of the existing keyword-based search.
/unconstrainedSearch	post	In the other search functions, the data for only one extensionAttribute is returned. Here the data for ALL extension attributes is returned. That is, all extensionAttributes where the percentage of fields that can be populated ('density') is above a certain threshold.
/correlationBasedSearch	post	The correlation-based Search then extends the table with multiple new columns, all of which correlate to the correlationAttribute (which you specify).
/fetchTable	get	The search functions don't return the actual table data. Instead, they return the table names and the relevant instance- and schema- matches. To create the extensionAttribute columns, the front end also has to call this fetchTable function for each of the tables it needs the data for.

/fetchTablePOST	post	The search functions don't return the actual table data. Instead they return the table names and the relevant instance- and schema- matches. To create the extensionAttribute columns, the front end also has to call this fetchTable function for each of the tables it needs the data for.

Table B.2: List of API calls

Appendix C

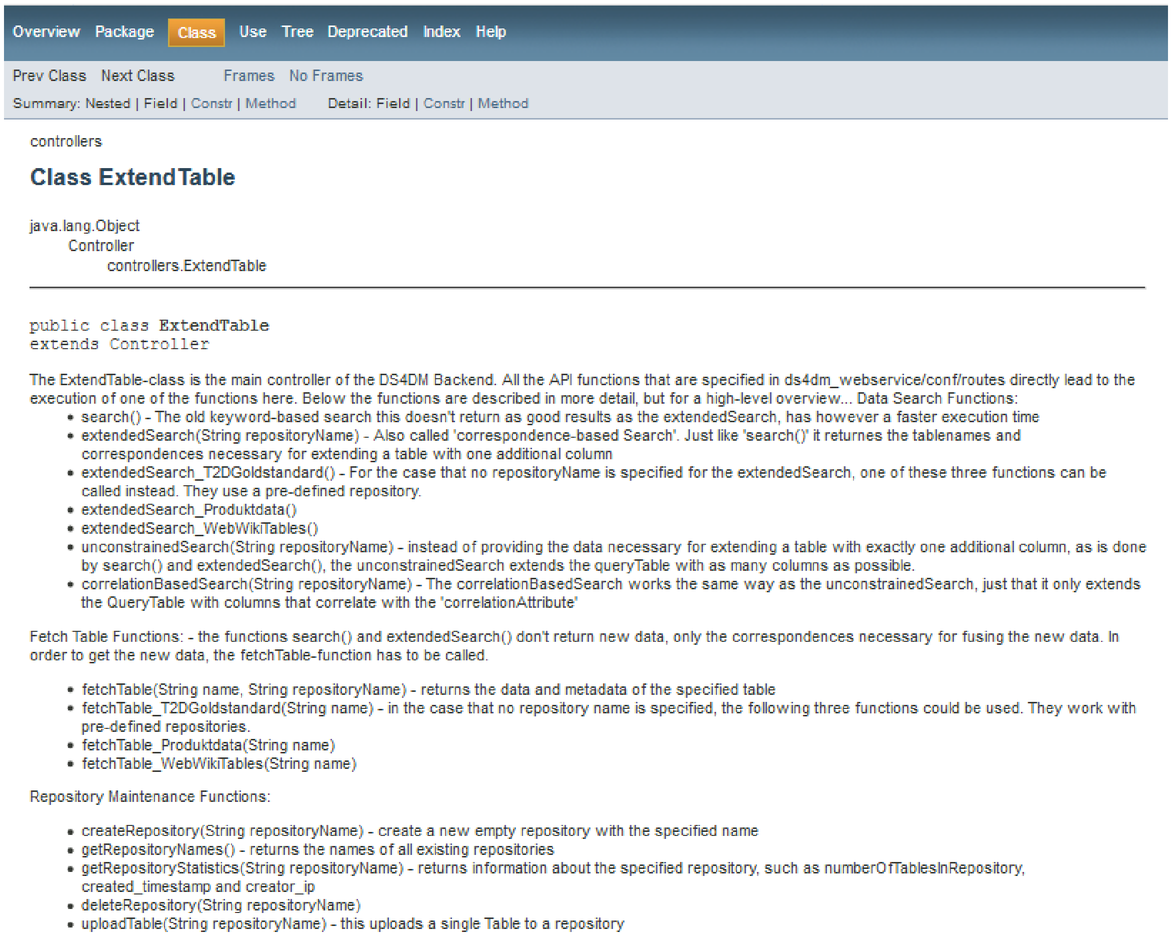
Documentation

Several different documentations have been created. Namely: JavaDoc documentation, Swagger API-definition, a paper that was published in the LWDA conference in 2018. These pieces of documentation are essential for making the system easy to use, allowing other people to understand and build upon the open source code and to making the system more known in the community.

C.1 JavaDoc Documentation

JavaDocs provide a detailed documentation of the source code of the system, they are a common way of documenting Java projects (Kramer, 1999).
The JavaDoc created for the table extension system has been made accessible online[1].
For each of the major java classes of the system one webpage has been created. These webpages give an overview of the respective class – see figure C.1and describe in detail the operation steps done by the individual methods – see figure C.2.
This documentation allows other developers to understand the code very fast (as well as give an overview for future reference).

[1]`http://web.informatik.uni-mannheim.de/ds4dm/Javadoc/index.html`


Overview Package Class Use Tree Deprecated Index Help
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

controllers

Class ExtendTable

java.lang.Object
    Controller
        controllers.ExtendTable

```
public class ExtendTable
extends Controller
```

The ExtendTable-class is the main controller of the DS4DM Backend. All the API functions that are specified in ds4dm_webservice/conf/routes directly lead to the execution of one of the functions here. Below the functions are described in more detail, but for a high-level overview... Data Search Functions:

- search() - The old keyword-based search this doesn't return as good results as the extendedSearch, has however a faster execution time
- extendedSearch(String repositoryName) - Also called 'correspondence-based Search'. Just like 'search()' it returnes the tablenames and correspondences necessary for extending a table with one additional column
- extendedSearch_T2DGoldstandard() - For the case that no repositoryName is specified for the extendedSearch, one of these three functions can be called instead. They use a pre-defined repository.
- extendedSearch_Produktdata()
- extendedSearch_WebWikiTables()
- unconstrainedSearch(String repositoryName) - instead of providing the data necessary for extending a table with exactly one additional column, as is done by search() and extendedSearch(), the unconstrainedSearch extends the queryTable with as many columns as possible.
- correlationBasedSearch(String repositoryName) - The correlationBasedSearch works the same way as the unconstrainedSearch, just that it only extends the QueryTable with columns that correlate with the 'correlationAttribute'

Fetch Table Functions: - the functions search() and extendedSearch() don't return new data, only the correspondences necessary for fusing the new data. In order to get the new data, the fetchTable-function has to be called.

- fetchTable(String name, String repositoryName) - returns the data and metadata of the specified table
- fetchTable_T2DGoldstandard(String name) - in the case that no repository name is specified, the following three functions could be used. They work with pre-defined repositories.
- fetchTable_Produktdata(String name)
- fetchTable_WebWikiTables(String name)

Repository Maintenance Functions:

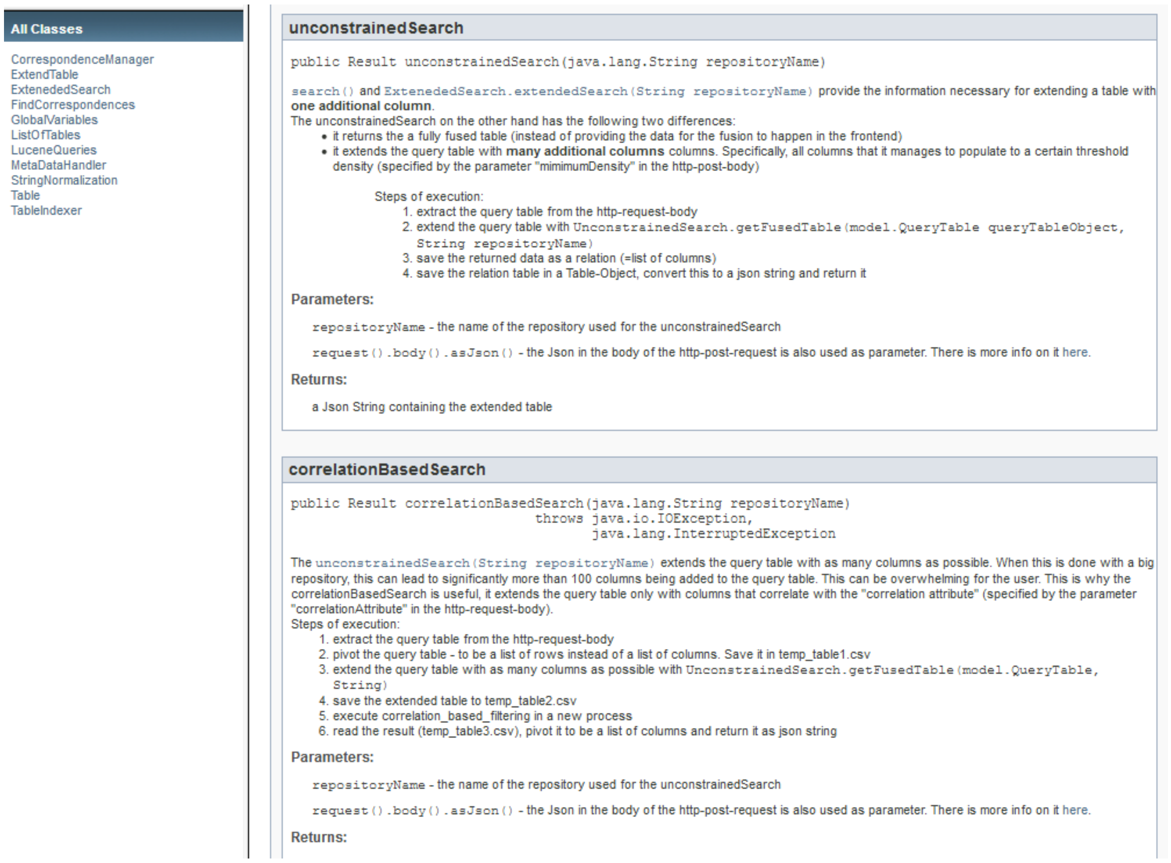
- createRepository(String repositoryName) - create a new empty repository with the specified name
- getRepositoryNames() - returns the names of all existing repositories
- getRepositoryStatistics(String repositoryName) - returns information about the specified repository, such as numberOfTablesInRepository, created_timestamp and creator_ip
- deleteRepository(String repositoryName)
- uploadTable(String repositoryName) - this uploads a single Table to a repository

Figure C.1: Screenshot of the overview of the ExtendTable-class in the JavaDoc

C.2 API Documentation

As mentioned, the table extension system runs as a web service with a REST API. To run different operations, the user must do the appropriate API calls to the system. In order to work correctly, these API calls must follow a certain shape.

In order to effectively communicate the required shape of the API calls to users, good API specification is essential. Various frameworks for creating API specifications/documentation exist (Sohan, Anslow Maurer, 2015), for this project Swagger was chosen.



All Classes

CorrespondenceManager
ExtendTable
ExtenededSearch
FindCorrespondences
GlobalVariables
ListOfTables
LuceneQueries
MetaDataHandler
StringNormalization
Table
TableIndexer

unconstrainedSearch

```
public Result unconstrainedSearch(java.lang.String repositoryName)
```

`search()` and `ExtenededSearch.extendedSearch(String repositoryName)` provide the information necessary for extending a table with **one additional column**.
The unconstrainedSearch on the other hand has the following two differences:

- it returns the a fully fused table (instead of providing the data for the fusion to happen in the frontend)
- it extends the query table with **many additional columns** columns. Specifically, all columns that it manages to populate to a certain threshold density (specified by the parameter "mimimumDensity" in the http-post-body)

Steps of execution:

1. extract the query table from the http-request-body
2. extend the query table with `UnconstrainedSearch.getFusedTable(model.QueryTable queryTableObject, String repositoryName)`
3. save the returned data as a relation (=list of columns)
4. save the relation table in a Table-Object, convert this to a json string and return it

Parameters:

`repositoryName` - the name of the repository used for the unconstrainedSearch

`request().body().asJson()` - the Json in the body of the http-post-request is also used as parameter. There is more info on it here.

Returns:

a Json String containing the extended table

correlationBasedSearch

```
public Result correlationBasedSearch(java.lang.String repositoryName)
                              throws java.io.IOException,
                                     java.lang.InterruptedException
```

The `unconstrainedSearch(String repositoryName)` extends the query table with as many columns as possible. When this is done with a big repository, this can lead to significantly more than 100 columns being added to the query table. This can be overwhelming for the user. This is why the correlationBasedSearch is useful, it extends the query table only with columns that correlate with the "correlation attribute" (specified by the parameter "correlationAttribute" in the http-request-body).
Steps of execution:

1. extract the query table from the http-request-body
2. pivot the query table - to be a list of rows instead of a list of columns. Save it in temp_table1.csv
3. extend the query table with as many columns as possible with `UnconstrainedSearch.getFusedTable(model.QueryTable, String)`
4. save the extended table to temp_table2.csv
5. execute correlation_based_filtering in a new process
6. read the result (temp_table3.csv), pivot it to be a list of columns and return it as json string

Parameters:

`repositoryName` - the name of the repository used for the unconstrainedSearch

`request().body().asJson()` - the Json in the body of the http-post-request is also used as parameter. There is more info on it here.

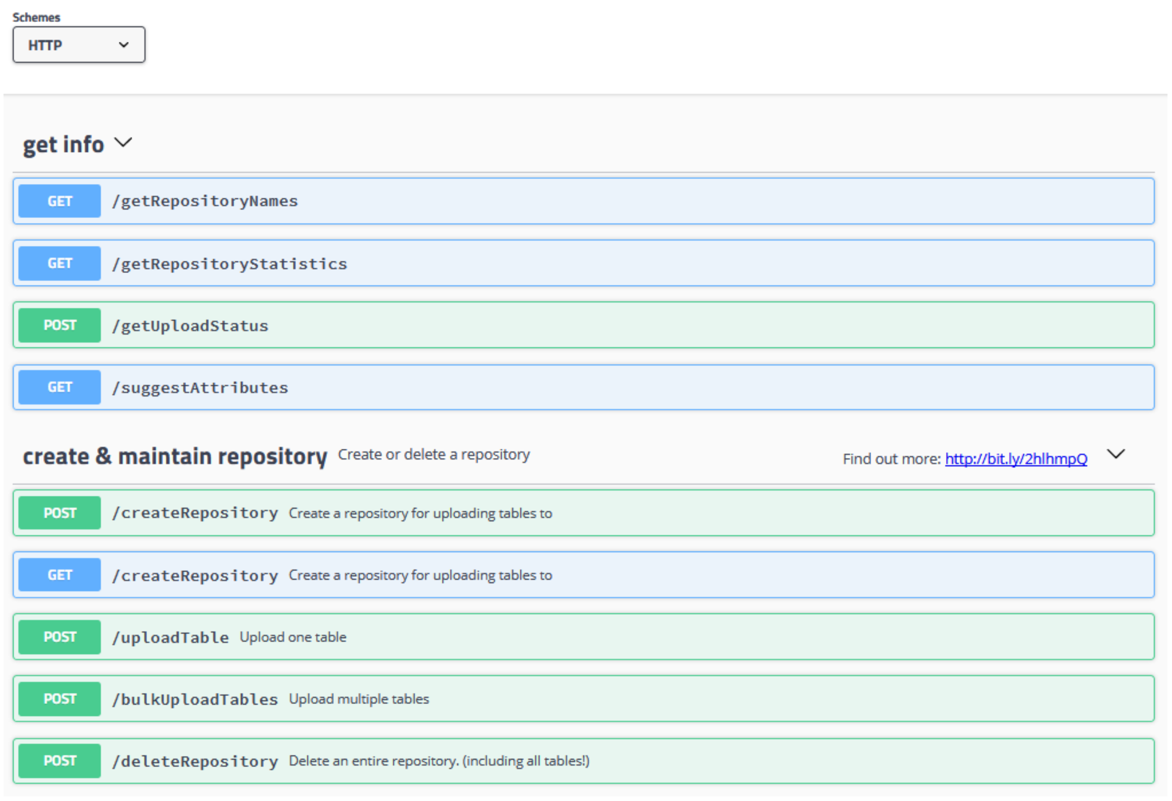
Returns:

Figure C.2: Screenshot of the documentation of some methods in the JavaDoc

The Swagger-API-specification of the table extension system, is also been published as a webpage[2]. It contains an entry for every possible API call – see Figure C.3.

For any of the API calls, the details of this API call can be displayed by clicking on it. E.g. when clicking on the entry for "/unconstrainedSearch" the all the required details about this API call appear below – see Figure C.4. These details include the type of http request (GET or POST), the http-header parameters and the format of the json string in the message body.

[2]`http://web.informatik.uni-mannheim.de/ds4dm/API-definition.html#/`


Schemes
HTTP

get info

GET /getRepositoryNames

GET /getRepositoryStatistics

POST /getUploadStatus

GET /suggestAttributes

create & maintain repository Create or delete a repository — Find out more: http://bit.ly/2hlhmpQ

POST /createRepository Create a repository for uploading tables to

GET /createRepository Create a repository for uploading tables to

POST /uploadTable Upload one table

POST /bulkUploadTables Upload multiple tables

POST /deleteRepository Delete an entire repository. (including all tables!)

Figure C.3: Screenshot of the Swagger-API-specification

C.3 Conference Paper

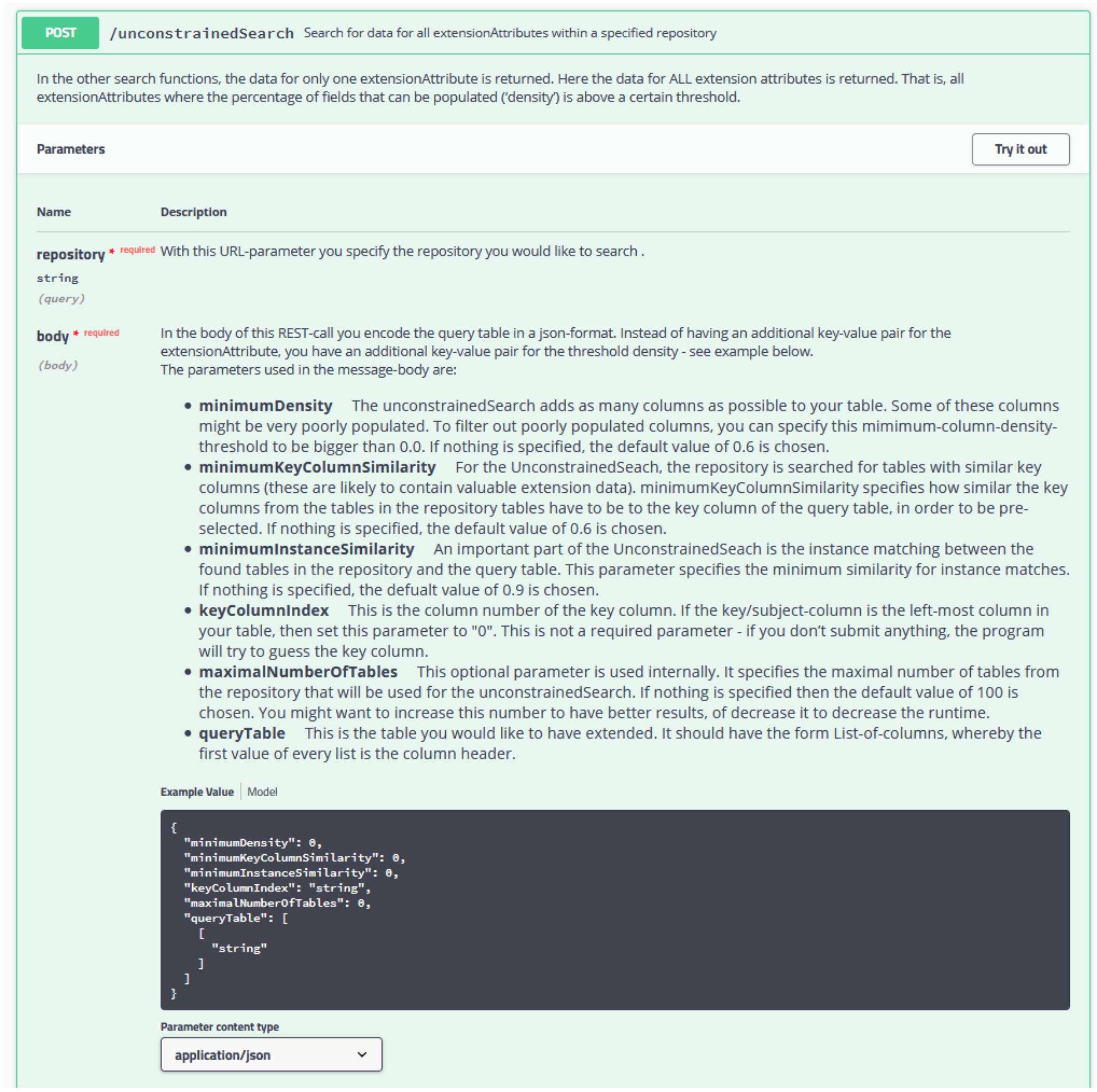
A short paper was submitted to the LWDA 2018 conference[3], where it was accepted. This paper by Kleppmann et al. (2018) is called "Density- and Correlation-based Table Extension". It describes the table extension system and the new algorithms developed for the unconstrained- and correlation-based- table extension, as well as the evaluation of these algorithms. The paper has been posted online and can be downloaded as a pdf-file[4].

The paper was presented at the LWDA 2018 conference on the 23rd August 2018 in Mannheim and was well-received.

This paper did not only serve to spread awareness about the work, but also to advance

[3]https://www.uni-mannheim.de/lwda-2018/

[4]https://dws.informatik.uni-mannheim.de/fileadmin/lehrstuehle/ki/pub/KleppmannBizer-DensityAndCorrelationBasedTableExtension-LWDA2018.pdf



POST /unconstrainedSearch Search for data for all extensionAttributes within a specified repository

In the other search functions, the data for only one extensionAttribute is returned. Here the data for ALL extension attributes is returned. That is, all extensionAttributes where the percentage of fields that can be populated ('density') is above a certain threshold.

Parameters

Name	Description
repository * required string (query)	With this URL-parameter you specify the repository you would like to search .
body * required (body)	In the body of this REST-call you encode the query table in a json-format. Instead of having an additional key-value pair for the extensionAttribute, you have an additional key-value pair for the threshold density - see example below. The parameters used in the message-body are:

- **minimumDensity** The unconstrainedSearch adds as many columns as possible to your table. Some of these columns might be very poorly populated. To filter out poorly populated columns, you can specify this mimimum-column-density-threshold to be bigger than 0.0. If nothing is specified, the default value of 0.6 is chosen.
- **minimumKeyColumnSimilarity** For the UnconstrainedSeach, the repository is searched for tables with similar key columns (these are likely to contain valuable extension data). minimumKeyColumnSimilarity specifies how similar the key columns from the tables in the repository tables have to be to the key column of the query table, in order to be pre-selected. If nothing is specified, the default value of 0.6 is chosen.
- **minimumInstanceSimilarity** An important part of the UnconstrainedSeach is the instance matching between the found tables in the repository and the query table. This parameter specifies the minimum similarity for instance matches. If nothing is specified, the defualt value of 0.9 is chosen.
- **keyColumnIndex** This is the column number of the key column. If the key/subject-column is the left-most column in your table, then set this parameter to "0". This is not a required parameter - if you don't submit anything, the program will try to guess the key column.
- **maximalNumberOfTables** This optional parameter is used internally. It specifies the maximal number of tables from the repository that will be used for the unconstrainedSearch. If nothing is specified then the default value of 100 is chosen. You might want to increase this number to have better results, of decrease it to decrease the runtime.
- **queryTable** This is the table you would like to have extended. It should have the form List-of-columns, whereby the first value of every list is the column header.

Example Value | Model

```
{
  "minimumDensity": 0,
  "minimumKeyColumnSimilarity": 0,
  "minimumInstanceSimilarity": 0,
  "keyColumnIndex": "string",
  "maximalNumberOfTables": 0,
  "queryTable": [
    [
      "string"
    ]
  ]
}
```

Parameter content type: application/json

Figure C.4: Screenshot of the API-specification details of the "/unconstrainedSearch"-call

the state of knowledge in this specific domain.

Appendix D

Collaboration with RapidMiner

According to Virzi (1992) it is essential for evaluating the usability of a software system to have evaluators that were not involved with the design or implementation of the system, as they don't have prior understanding of the system.

This project was fortunate to have several such external evaluators. They not only helped with evaluating the usability, but also the wide applicability (by using the system for different scenarios), the availability of the system (by requiring it to run continuously), and the correctness of the system (qualitative analysis by looking at the results).

The most important external evaluation occurred due to a project from the company RapidMiner GmbH[1]. RapidMiner GmbH is the creator of the popular data mining tool RapidMiner Studio[2]. This tool provides a graphical user interface where users can create data analysis workflows by connecting different operators with arrows – see the screenshot at the top of Figure D.1. There are operators for data access, data transformation and machine learning.

RapidMiner is interested in creating new operators for table extension. They

[1]`https://rapidminer.com/`

[2]`https://rapidminer.com/products/studio/`

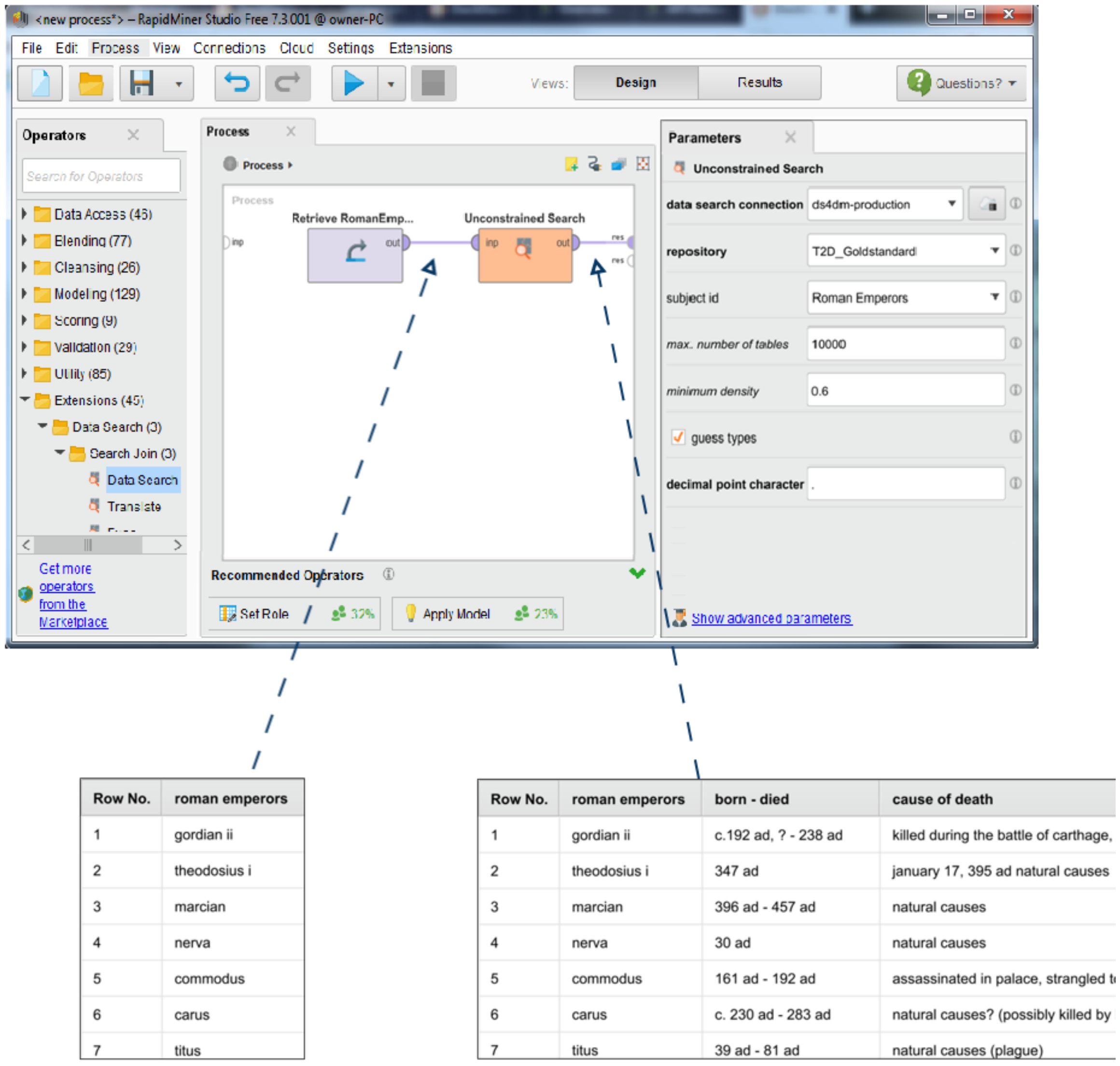
decided to develop prototypes of these operators which are effectively wrappers that call the operators of the new table extension system.

In other words, the table extension system serves as a backend that provides the functionality for the graphical frontend: RapidMiner Studio.
Figure D.1 shows a workflow where a table with the names of Roman emperors is loaded into RapidMiner Studio, this table is then passed into an unconstrained-table-extension operator, which adds additional columns to this table. Finally, the extended table is passed to the standard output on the right.

RapidMiner did not only implement an operator for unconstrained table extension (called 'Unconstrained Search' in Figure D.1). The other operators that were implemented for correlation-based table extension, creating a new repository and uploading tables to the repository. On the left of Figure D.2 you can see the create-repository operator (step1), upload-tables-to-repository operator (step2) and unconstrained-table-extension operator (step3). Figure D.2 tries to illustrate how the RapidMiner operators communicate with the new table extension system. The RapidMiner Studio operators directly makes API calls to the public API of the new table extension system. The table extension system executes the requested operations ('createRepository' in step1, 'uploadTable' in step2 and 'unconstrainedSearch' in step3) and returns the answers (e.g. in step3: the extended table).

RapidMiner has made these table extension operators openly available as the "Data Search for Data Mining" – RapidMiner Extension. This extension can be freely downloaded from the RapidMiner Marketplace – see Figure D.3. Up until the 22nd September 2018 it has been downloaded XX times.
For RapidMiner these prototype table extension operators have been a success. They are currently developing their own table extension backend web service, which is

unconstrained table extension2.png



Parameter	Value
data search connection	ds4dm-production
repository	T2D_Goldstandard
subject id	Roman Emperors
max. number of tables	10000
minimum density	0.6
guess types	✓
decimal point character	.

Row No.	roman emperors
1	gordian ii
2	theodosius i
3	marcian
4	nerva
5	commodus
6	carus
7	titus

Row No.	roman emperors	born - died	cause of death
1	gordian ii	c.192 ad, ? - 238 ad	killed during the battle of carthage,
2	theodosius i	347 ad	january 17, 395 ad natural causes
3	marcian	396 ad - 457 ad	natural causes
4	nerva	30 ad	natural causes
5	commodus	161 ad - 192 ad	assassinated in palace, strangled t
6	carus	c. 230 ad - 283 ad	natural causes? (possibly killed by
7	titus	39 ad - 81 ad	natural causes (plague)

Figure D.1: Diagram showing the tables at various stages of a RapidMiner Studio workflow

heavily based on the system we tested.

Having this collaboration with RapidMiner has been invaluable for this project. They provided great feedback and helped greatly with the evaluations.

extension - various operations.png

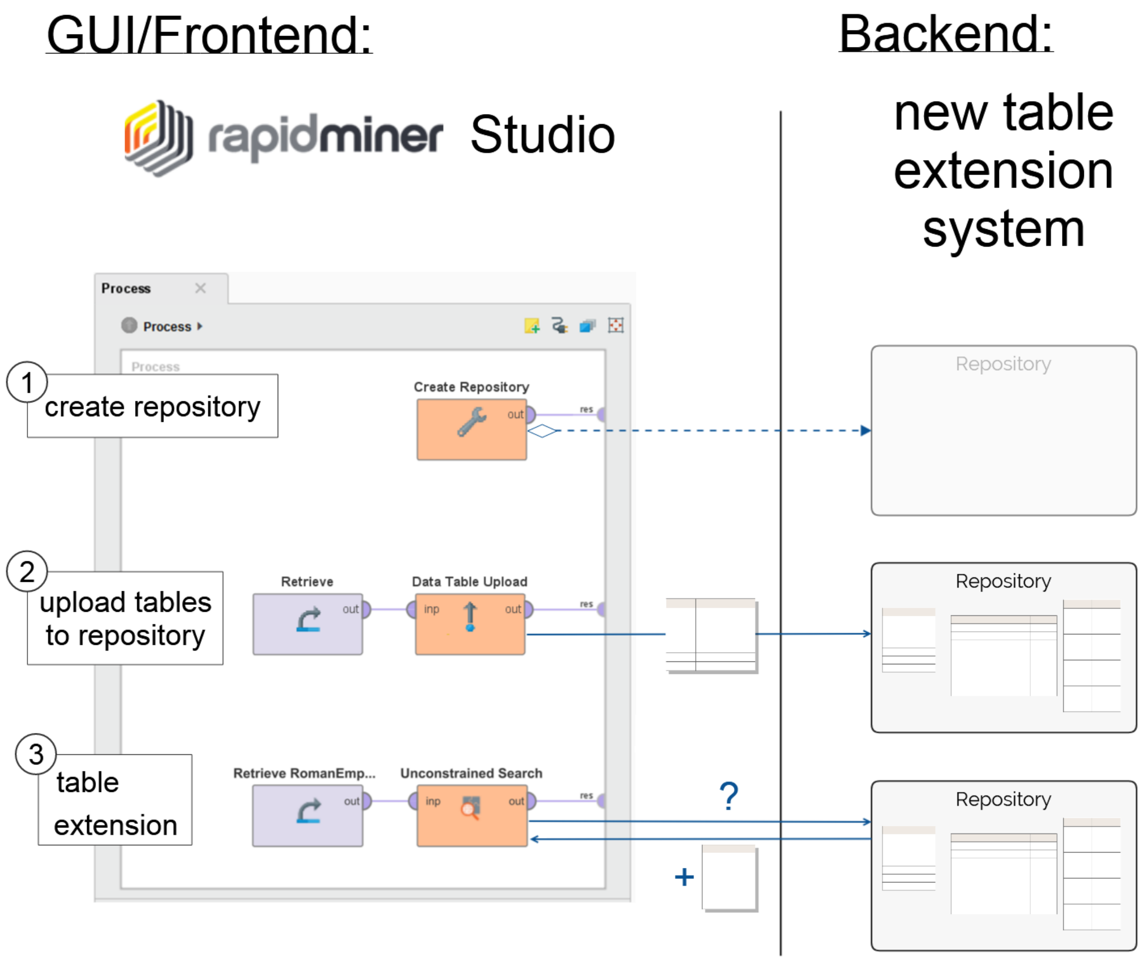
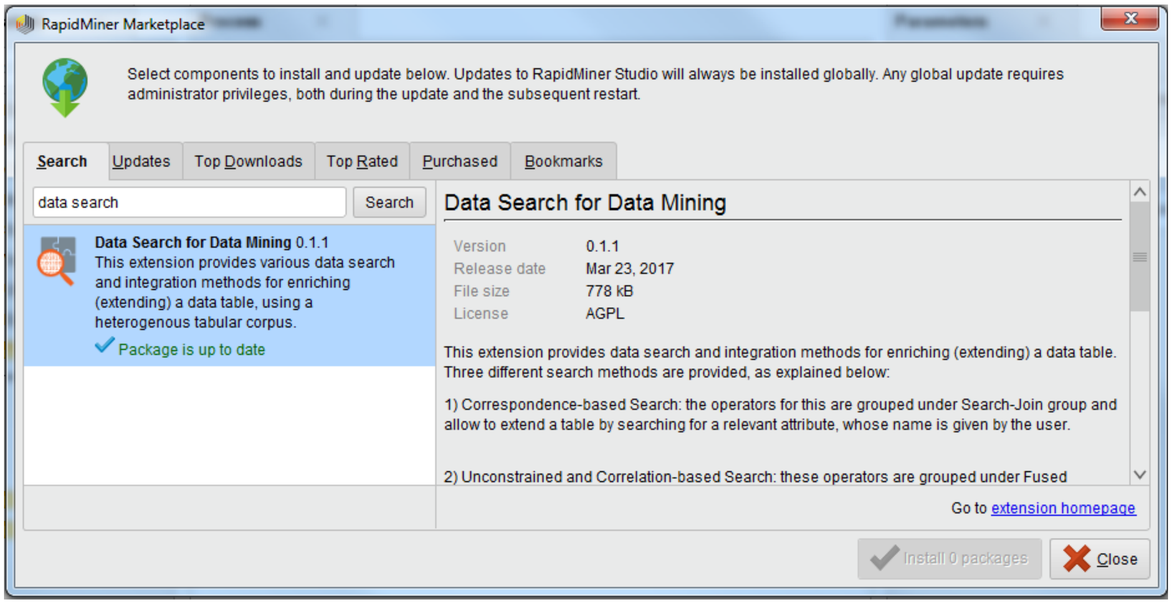


Figure D.2: Visualisation of how various RapidMiner Studio operators communicate with the new table extension system



Select components to install and update below. Updates to RapidMiner Studio will always be installed globally. Any global update requires administrator privileges, both during the update and the subsequent restart.

Data Search for Data Mining 0.1.1
This extension provides various data search and integration methods for enriching (extending) a data table, using a heterogenous tabular corpus.
✓ Package is up to date

Data Search for Data Mining

Version	0.1.1
Release date	Mar 23, 2017
File size	778 kB
License	AGPL

This extension provides data search and integration methods for enriching (extending) a data table. Three different search methods are provided, as explained below:

1) Correspondence-based Search: the operators for this are grouped under Search-Join group and allow to extend a table by searching for a relevant attribute, whose name is given by the user.

2) Unconstrained and Correlation-based Search: these operators are grouped under Fused

Go to extension homepage

Figure D.3: Screenshot of the RapidMiner Marketplace

Appendix E

Evaluations of components

Section 4.3.5 contains the results from evaluating the overall correctness of the table extension algorithms. A lot of insight about the performance of the table extension algorithm can however be gained by additionally evaluating individual parts of or the algorithm. The three parts, that are most critical to the correct execution of the table extension, have been evaluated. They are:

- The finding of tables with similar subject columns to that of the to-be-extended table (step 2)
- Instance matching (step 3)
- Schema matching (step 4)

E.1 Evaluation of the Finding of Tables with Similar Subject Columns

In step 2 of the unconstrained table extension algorithm, the repository is searched for tables that have a similar subject column to the subject column of the to-be-extended table. In practice this is done with a Lucene index:
When the tables were uploaded to the repository, their subject columns were automatically detected and saved in the Lucene search-index called SubjectColumnIndex (see

3.2.2.). Later, in step 2 of the unconstrained table extension the subject column of the to-be-extended table is used as search term to retrieve similar subject columns from the SubjectColumnIndex search index. The tables with these similar subject columns are considered potentially matching tables. In the literature this step of finding potentially matching tables is called "blocking" – see section 2.2.2.4. The general way of evaluating such a blocking procedure is to calculate the following two metrics (Nin Guerrero, Muntés Mulero, Martínez Bazán & Larriba Pey, 2007, Gentile, Ristoski, Eckel, Ritze & Paulheim, 2017):

- Reduction ratio

  This is the percentage of tables that are no longer considered potentially matching after the blocking step.

- Pair completeness

  This is the percentage of tables that were found to be potentially matching by the blocking step, that are indeed true matches.

The reduction ratio and pair completeness were calculated by analysing 13 different queries that were performed on the repository with the 460 513 WikiTables.
In average, 25 tables were returned by step 2 (the search for tables with similar subject-columns). The number of potentially matching tables was thereby reduced from 460 513 to 25 – a reduction ratio of 99.994%.
The returned tables were manually checked to see if they truly match the to-be-extended table (i.e. contain about the same type of entities). In average 35% of the potentially matching tables found in step 2 are truly matching tables. This corresponds to a pair completeness of 35%.

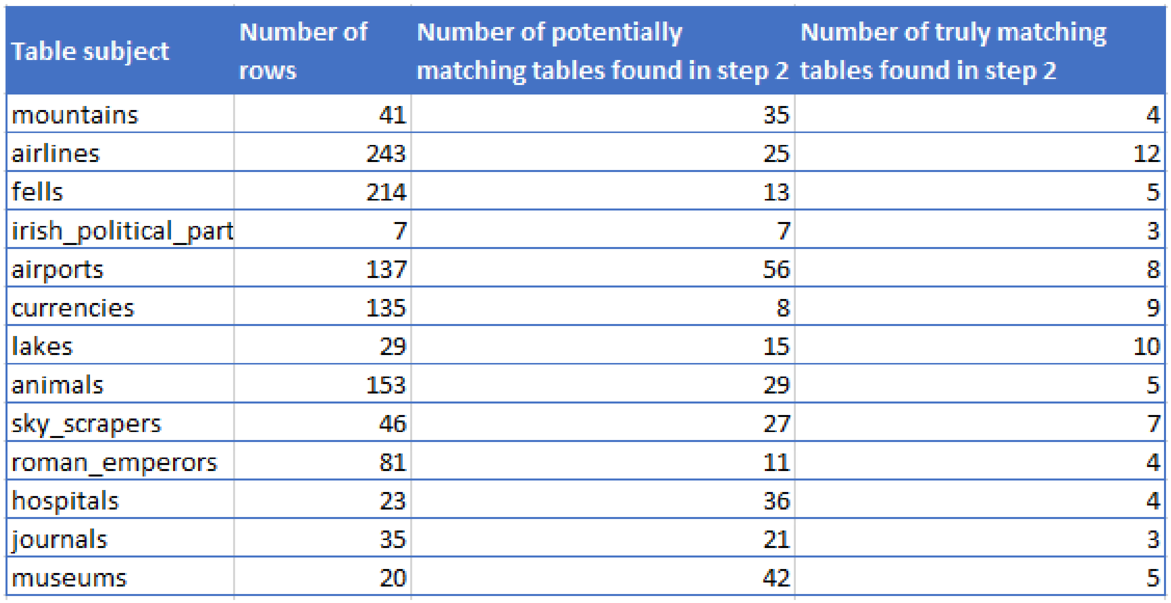


Table subject	Number of rows	Number of potentially matching tables found in step 2	Number of truly matching tables found in step 2
mountains	41	35	4
airlines	243	25	12
fells	214	13	5
irish_political_part	7	7	3
airports	137	56	8
currencies	135	8	9
lakes	29	15	10
animals	153	29	5
sky_scrapers	46	27	7
roman_emperors	81	11	4
hospitals	23	36	4
journals	35	21	3
museums	20	42	5

Table E.1: Number of tables found by the SubjectColumnIndex

E.2 T2D Goldstandard

The following two evaluations used as data the T2D Goldstandard[1]. This is a corpus of tables of 263 tables which was created by Ritze, Lehmberg & Bizer in 2015. The 263 tables cover a wide range of topics such as populated places, organizations, animals, etc.

The schemas (=columns) and instances (=rows) of the 263 tables in the corpus had been manually mapped to the corresponding schemas and instances in dbpedia. From these mappings to dbpedia, correspondences between the tables were deduced – two columns correspond if they were mapped to the same dbpedia instance.

The deduced correspondences are instance- and schema- correspondences. The evaluation of the instance matching makes use of the instance correspondences, the evaluation of the schema matching makes use of the schema correspondences. For more information on these types of correspondences, please refer to section 2.2.2.2.

[1]`http://webdatacommons.org/webtables/goldstandardV2.html`

E.3 Evaluation of Instance Matching

Instance matching is the task of finding instance correspondences between two tables. Instance correspondences are two rows from the two tables that contain data about the same entity – see section 2.2.2.2 for more information.

In section 3.2.6 the instance matching algorithm is described, it uses data-type-specific similarity measures to compare the similarities of both the subject-column and non-subject-column values.

The evaluation was performed with pairs of tables from the T2D Goldstandard. As mentioned above, the true instance correspondences between these tables are already known and used as ground truth.
The instance matching algorithm was evaluated by comparing the instance correspondences it predicted with the true instance correspondences from the ground truth. The evaluation produced the following evaluation scores:

- Precision = 0.949
- Recall = 0.624
- F1 = 0.753

It is interesting to note that during the development of the instance matching algorithm, this evaluation was performed many times in order to optimize the algorithm. Specifically, find the optimum weighting between subject-column similarity and non-subject-column similarity (50%, 50%) and threshold similarity (0.7).

E.4 Evaluation of Schema Matching

Schema matching is the task of finding schema correspondences between two tables. Schema correspondences are two columns from the two tables that describe the same attribute – see section 2.2.2.2 for more information.

In section 3.2.7 the schema matching algorithm is described, it uses all available information to make the best possible prediction: both the column values and the column headers of the columns are compared, the previously found instance matches as the data types of the values are used for identifying the schema correspondences.

The evaluation was performed with pairs of tables from the T2D Goldstandard. As mentioned above, the true schema correspondences between these tables are already known and used as ground truth.
The schema matching algorithm was evaluated by comparing the schema correspondences it predicted with the true schema correspondences from the ground truth. The evaluation produced the following evaluation scores:

- Precision = 0.803
- Recall = 0.712
- F1 = 0.755

The above evaluation of the schema matching algorithm was performed many times in order to optimize several parameters of this algorithm. Specifically, the minimum similarity threshold for schema matches (0.8) as well as the weights of the column-header similarity and the column-value similarity (0.2 and 0.8, respectively).