The Mannheim Search Join Engine

Oliver Lehmberg*, Dominique Ritze*, Petar Ristoski*, Robert Meusel†, Heiko Paulheim*, Christian Bizer‡

*Data and Web Science Group, University of Mannheim, B6, 26, 68159 Mannheim, Germany

Abstract

A Search Join is a join operation which extends a user-provided table with additional attributes based on a large corpus of heterogeneous data originating from the Web or corporate intranets. Search Joins are useful within a wide range of application scenarios: Imagine you are an analyst having a local table describing companies and you want to extend this table with attributes containing the headquarters, turnover, and revenue of each company. Or imagine you are a film enthusiast and want to extend a table describing films with attributes like director, genre, and release date of each film. This article presents the Mannheim Search Join Engine which automatically performs such table extension operations based on a large corpus of Web data. Given a local table, the Mannheim Search Join Engine searches the corpus for additional data describing the entities contained in the input table. The discovered data is then joined with the local table and is consolidated using schema matching and data fusion techniques. As result, the user is presented with an extended table and given the opportunity to examine the provenance of the added data. We evaluate the Mannheim Search Join Engine using heterogeneous data originating from over one million different websites. The data corpus consists of HTML tables, as well as Linked Data and Microdata annotations which are converted into tabular form. Our experiments show that the Mannheim Search Join Engine achieves a coverage close to 100% and a precision of around 90% for the tasks of extending tables describing cities, companies, countries, drugs, books, films, and songs.

Keywords: Table Extension, Data Search, Search Joins, Web Tables, Microdata, Linked Data

1. Introduction

Imagine you are a marketing manager who wants to group the customers of a company according to different properties of the countries in which the customers are located in order to select customers that should be targeted by a marketing campaign. While the data about customers can be found in the company’s internal data sources, further background information about the countries may not be stored there. Relevant data about countries could for instance include population, area, GDP, or human development index. Today, the manager needs to manually search and integrate data about each country using search engines such as Google, access the small set of online databases he knows about, or copy-and-paste values from Wikipedia. Manually searching for data is cumbersome and the manager will likely miss a large fraction of all relevant data sources that are available on the Web.

This article presents the Mannheim Search Join Engine (MSJ Engine) which supports the manager in reaching his goal by automating the data search and integration tasks. Given a local table, the MSJ Engine searches a heterogeneous data corpus for additional data describing the entities contained in the input table. The search operation does not assume any external knowledge about correspondences on schema or instance level. The discovered data are then joined with the local table and their content is consolidated using schema matching and data fusion methods. As result, the user is presented with an extended table and given the opportunity to examine the provenance of the added data. We evaluate the engine using a data corpus consisting of 36 million tables originating from over one million different websites. The data corpus contains HTML tables as well as Linked Data [1] and Microdata annotations [2] which are converted into tabular form. In contrast to existing research on table extension by Google [3] and Microsoft [4], our data corpus as well as
the source code of the MSJ Engine are publicly available.

The article is structured as follows: Section 2 gives an overview of the architecture of the MSJ Engine and describes the methods that are employed for data normalization, data search, and data consolidation. Section 3 describes the data corpus that was used to evaluate the engine and presents the results of our table extension experiments. Section 4 compares the results of the MSJ Engine with related work, while Section 5 outlines directions for future work.

2. Architecture

A Search Join is a join operation which extends a local, user-provided table (called query table) with additional attributes based on a large corpus of heterogeneous tabular data [5]. Search Joins can be described as a concatenation of three operations: a search operation \( s \), a multi-join operation \( m \), and a consolidation operation \( c \):

\[
R = c(m(T_q, s_{T_a}(T_c)))
\]

where \( R \) is the result table, \( T_q \) is the query table, \( T_c \) is a corpus of heterogeneous tables, and \( a \) is an optional, user-specified parameter that allows the search to be constrained to a specific attribute (constrained query), e.g. only search for information about the population of countries. If the parameter \( a \) is not specified, all discovered attributes will be added to the result table (unconstrained query).

Figure 1 gives an overview of the functionality of the MSJ Engine. The functionality can be divided into three areas: Table Indexing, Table Search and Data Consolidation. In the following, we will describe each area in detail.

2.1. Table Indexing

The MSJ Engine uses simple entity-attribute tables as internal data model. Each table describes a set of entities using a set of single-valued attributes. Each attribute has a header which we assume to be some surface form of the attribute’s semantic intention. We distinguish the following attribute data types: nominal values, numbers with or without unit of measurement, timestamps, and geo-coordinates. As many Web and intranet data sources provide natural language labels for the entities that they describe, we require each table to contain a subject attribute containing the entity name, e.g. Lady Gaga, U.S.A. or United States. This approach to treat entity names as pseudo-keys is also used by the related work from Google [3] and Microsoft [4].

Before a table from the data corpus \( T_c \) is indexed, the value of each cell is normalized, i.e., tokenized, lowercased, values in brackets and stop words are removed.

In order to be indexed, a table needs to fulfill two conditions: (1) it must contain at least three attributes and describe at least five entities, (2) it must contain a subject attribute. Applying these filtering conditions ensures a minimal quality of the remaining tables.

We apply the following heuristic in order to identify the subject attribute of a table: If a table contains an \texttt{rdfs:label} attribute or some other attribute having a header containing the string \texttt{name}, this attribute will be chosen as subject attribute. Otherwise, the string attribute with the highest number of unique values is chosen as subject attribute. In cases where two or more attributes contain equally high numbers of unique values, the left-most attribute is chosen. Furthermore, we only consider attributes with at least 60% unique values.

To identify attribute headers, we use the following heuristic: We assume that the attribute headers are in the first non-empty row of the Web table that contains at least 80% non-empty cells. We present an evaluation of both pre-processing heuristics in Section 3.2.

The data type of each table attribute is identified based on its values. First, the data type of each value of the attribute is detected by using about 100 manually defined regular expressions which are able to detect the data types number (with or without unit of measurement), timestamp and geo-coordinates. Additionally, the algorithm uses around 200 manually generated rules for converting units of measurements to the corresponding base unit (metric system), e.g. 8 sq. mi. will be converted to 20.72M square meter. After the data type of each value is detected, the final attribute data type is decided using majority voting.

As final step of the indexing procedure, the normalized subject values and attribute headers of each table are stored in a Lucene index\(^1\), which is used later by the search operation.

2.2. Table Search

The query table \( T_q \) is preprocessed in the same manner as the tables of the data corpus \( T_c \). Then, the search operator \( s \) is applied and tries to find matching subject values in the previously indexed tables \( T_c \). For deciding whether a subject value from a table matches a subject value from the query table, two different methods

\(^1\)http://lucene.apache.org/
are available: exact subject value matching, and similar subject value matching.

To find similar subject values, we first query the Lucene index for token-based matches which means that at least one token from the subject value of the query table must exactly match a token of the subject value of the indexed table $T_i$ in $T_c$. Afterwards, we use the FastJoin matcher [6] to calculate the final similarity between the two subject values. The FastJoin matcher calculates so called “fuzzy token matching based similarity”, which extends token-based similarity functions (Jaccard similarity in this case) with allowing fuzzy matching between two tokens. Especially for data from Web tables, allowing matches with a typo or slightly different spellings is reasonable.

After the search is completed, for each of the Web tables where at least one matching subject value was found, the table relevance score is computed. The relevance score $r_{T(i)}$ for a Web table $T_i$ is computed as the average Lucene similarity between the matched subject values from the query table and the Web table, using Eq. 2:

$$r_{T(i)} = \frac{1}{|T(q)|} \sum l(q, k, t, k)$$

where $l(q, k, t, k)$ is the Lucene similarity score of the matched subject values, and $|T(q)|$ is the cardinality of the query table. As the final step of the search operation, all tables are ordered by their relevance score. Only the top-k tables, as specified by the user, are selected and passed on to the next operation.

### 2.3. Data Consolidation

After the search is completed, the tables are joined by applying the multi-join operation $m$, which performs a series of left outer joins between the query table and the tables returned by the search using the subject value matches identified before. Afterwards, the consolidation operation $c$ combines attributes that represent the same property. The applied consolidation method depends on whether a constrained or unconstrained query is executed.

A constrained query considers only attributes matching a header specified by the user. The heuristic we apply here is to accept all attribute headers that contain the given header. For example, if the user queries for “GDP”, attributes with header “Total GDP” and “GDP (US$)” are also matched. After the filtering, the remaining attributes are consolidated to a single column. To do so, related values for each subject value are clustered first, based on similarity measures as proposed by Rinser et al. [7] for different data types. For numeric values, the measure takes the deviation into account while the Jaccard measure on n-grams is applied for strings. Additionally, we set the similarity of two strings to 1.0 if one of the values is contained in the other.
one. As an example, the values “Federal Republic of Germany” and “Germany” should be in the same cluster. The final value is then chosen by first selecting the cluster with the highest number of elements and then determining the most frequent value within this cluster (majority vote). Picking only the majority value, without clustering the values first, can lead to a final value which does not represent the majority, e.g. due to small numeric derivations or spelling errors. For example, the values “kuala lumpu,” “kuala lumpua”, and “kuala lumpur” are different spellings for the same city name but each of the names counted separately may not occur more often than the name “Ipoh”, which may lead to the wrong city being chosen as Malaysian capital.

The goal of unconstrained queries is to return as many different attributes as possible describing the entities of the query table. Thus, we only merge attributes which contain overlapping information (have the same semantic intention). In order to determine such corresponding attributes, we apply a combination of label- and instance-based schema matching techniques: First, attributes are matched based on their values for each subject. As in the case of constrained queries, we use a similar set of similarity measures as the one proposed by Rinser et al.. For matching attribute headers, we employ a matching operator that relies on background knowledge from DBpedia, YAGO, and Wordnet, i.e., we first identify matching concepts for each attribute header, then we exploit the IS-A relations between the concepts to calculate their similarity [8]. If no matching concepts are found, the operator falls back to string similarity of the attribute header using Levenshtein distance. Finally, the scores from the instance- and label-based operators are combined to determine which attributes can be merged. The user of the MSJ Engine can set the similarity threshold for two attributes to be merged. For the actual merging of the values, the user can choose between different conflict resolution strategies [9]. Within our experiments, we use voting for resolving conflicts between string values and the median function for combining numeric values. Additionally, the user can set a column density threshold which defines the minimum number of non-null values of an attribute after merging for including the attribute into the result table.

3. Evaluation

We evaluate the MSJ Engine using data from over one million different websites\(^2\). In the following, we will first give an overview of the evaluation data. Section 3.2 reports the results of evaluating the accuracy of the employed subject attribute and attribute header detection algorithms. Section 3.3 presents the results of experiments using the MSJ Engine to extend local tables describing countries, cities, companies, books, films, songs, drugs, and soccer players.

3.1. Evaluation Data Corpus

The data corpus that we use to evaluate the MSJ Engine consists of data that is published on the Web either as HTML tables, Linked Data [1], or as Micrdata annotations [2]. The corpus was build by combining two datasets that we have extracted from a large Web crawl ourselves, the WebDataCommons HTML Tables Dataset and WebDataCommons Microdata Dataset with two datasets that were gathered by third parties, the Billion Triples Challenge 2014 Dataset (BTC) and the WikiTables Dataset. Below, we give an overview of the content of each dataset and describe how the datasets were generated and combined.

WebDataCommons HTML Tables Dataset\(^3\): This dataset is the largest, non-commercial corpus of relational HTML tables that is available to the public. These tables have been extracted from the 2012 version of the CommonCrawl Web corpus\(^4\) which consists of 3.5 billion HTML pages originating from 43 million PLDs. Initial studies of the data showed that a large fraction of those Web tables is used for layout purpose. In order to identify the small fraction of HTML tables, containing relational data in the CommonCrawl Web corpus, we filtered out HTML tables that contain nested tables. This step, together with the filtering conditions (see Section 2.1) discards around 90% of all the tables in the corpus. Afterwards, in order to clean out further layout tables, we apply a classifier to distinguish between relational and non-relational tables. Our classifier relies on 16 table features: seven layout features, eight content type features and one word group feature. The features are similar to the features proposed by Wang et al. [11]. Out of the 11.2 billion innermost tables contained in the CommonCrawl, our classifier considers 147.6 million tables to be relational (1.3% of all tables, which confirms the finding by Cafarella et al. [10]). We evaluated

\(^2\)We consider a website to be a pay-level domain (PLD).

\(^3\)http://webdatacommons.org/webtables/

\(^4\)http://commoncrawl.org/
our table classification approach on a manually generated gold standard covering 7,350 randomly selected Web pages. The gold standard consists of 77,630 tables out of which 1,011 were annotated as relational tables. The evaluation showed that our approach achieves a precision of 58% and recall of 62%. As all our evaluation queries are formulated in English, we further filtered the corpus of relational tables to only contain tables originating from (mostly) English language top-level domains (com, org, net, eu, and uk). This resulting subset contains 35.7 million tables which originate from 1.5 million different PLDs.

WebDataCommons Microdata Dataset\(^5\): The dataset consists of Microdata annotations that we extracted from over 2.2 billion HTML pages contained in the Winter 2013 version of the CommonCrawl Web corpus. The dataset contains 8.7 billion RDF quads originating from over 463,000 different websites. A large part of the Microdata annotations employs the schema.org vocabulary which primarily describe products, people, organizations, news articles, locations and events\(^2\).

Billion Triples Challenge 2014 Dataset\(^12\): We use the Billion Triple Challenge 2014 dataset as a source of Linked Data\(^1\). The dataset contains around 4 billion RDF quads that were crawled from 47,000 PLDs in the period between February and June 2014. The crawler that was used to gather the dataset employed a breadth-first crawling strategy\(^13\) and was seeded with the URI list generated by Schmachtenberg et al.\(^14\).

WikiTables Dataset\(^6\): In addition to the WebDataCommons HTML Tables dataset, we also use a table corpus which has been extracted from Wikipedia pages in the course of the WikiTables project\(^15\). The corpus consists of 1.35 million tables. Although the corpus is rather small in comparison to the WebDataCommons HTML Tables dataset, it contains valuable data about entities of common interest (head entities).

As described in Section 2.1, the MSJ Engine uses entity-attribute tables as internal data model. We convert the Microdata and BTC datasets into tables by applying the same procedure that is used by "DBpedia as Tables"\(^7\): First, we split the datasets by PLD. Than, for each PLD-specific dataset we create a separate table for each rdfs:Class or owl:Class and add an attribute to this table for each RDF predicate that is used by instances of this class. In addition, we resolve multi-valued objects by keeping only the first value.

Table 1 gives an overview of the size of the data corpora before and after applying the filtering conditions described in Section 2.1: (a) our subject attribute detection algorithm must have been able to detect a subject attribute for the table and (b) the table must contain at least three attributes and describe at least five entities.

The second column of Table 1 indicates the number of tables in each datasets. The third column shows the number of triples. For the datasets converted into tables, the second column contains the number of tables after the conversion process. For table data, the third column denotes the number of triples that would be created by applying the reverse conversion process. As we can see, the tables generated from the BTC and Microdata dataset are much larger than the HTML tables, but there are a lot more HTML tables.

3.2. Evaluation of the Table Preprocessing Algorithms

It is important to correctly identify the subject attributes and the attribute headers for the MSJ Engine to deliver good results. We evaluate the performance of both, subject attribute and attribute headers detection, with respect to accuracy. To do so, we selected a random sample of tables from each of the four corpora (see Section 3.1) and manually annotated the correct subject attribute and the row that contains the attribute headers for each table.

The results of the evaluation of the subject attribute detection algorithm, together with the size of each sample, are shown in Table 2. A similar subject attribute detection algorithm is used by Ventis et al.\(^16\). Using a gold standard of 200 tables, they report that their algorithm was able to identify the correct subject attribute in 83% of the tables. The accuracy on our sample of 545 tables is 68%, which can be explained by the different sampling strategy: Ventis et al. used tables that were known to have a subject attribute, while we use random samples from our corpora.

The results of the evaluation of the attribute header detection algorithm are shown in Table 3. The approach for detecting attribute headers only considers headers that are contained in a single row. Therefore, the heuristic will fail on vertical tables\(^17\), on tables that require

\(^5\)http://webdatacommons.org/structureddata/
\(^6\)http://downey-n1.cs.northwestern.edu/public/
\(^7\)http://wiki.dbpedia.org/DBpediaAsTables
### Table 1: Statistics of the used corpora before and after filtering tables without a subject column and tables with less than five rows or three columns.

<table>
<thead>
<tr>
<th>Corpus</th>
<th># tables</th>
<th># triples</th>
<th># cols.</th>
<th>avg. cols.</th>
<th>min. cols.</th>
<th>max. cols.</th>
<th># rows</th>
<th>avg. rows</th>
<th>min. rows</th>
<th>max. rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tables</td>
<td>147.6M</td>
<td>6.7B</td>
<td>510M</td>
<td>3.49</td>
<td>2</td>
<td>2368</td>
<td>1.8B</td>
<td>12.41</td>
<td>1</td>
<td>70K</td>
</tr>
<tr>
<td>Microdata</td>
<td>96K</td>
<td>250M</td>
<td>388K</td>
<td>4.02</td>
<td>1</td>
<td>62</td>
<td>76M</td>
<td>785.96</td>
<td>1</td>
<td>630K</td>
</tr>
<tr>
<td>BTC</td>
<td>76K</td>
<td>634M</td>
<td>592K</td>
<td>7.78</td>
<td>2</td>
<td>5465</td>
<td>18M</td>
<td>244.00</td>
<td>1</td>
<td>122M</td>
</tr>
<tr>
<td>Wiki Tables</td>
<td>1.35M</td>
<td>220M</td>
<td>7.5M</td>
<td>5.34</td>
<td>0</td>
<td>2349</td>
<td>16M</td>
<td>10.97</td>
<td>0</td>
<td>5K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Preprocessed Corpora Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tables</td>
</tr>
<tr>
<td>Microdata</td>
</tr>
<tr>
<td>BTC</td>
</tr>
<tr>
<td>Wiki Tables</td>
</tr>
</tbody>
</table>

### Table 2: Accuracy of the subject attribute detection algorithm.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Tables</th>
<th># Correct</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tables</td>
<td>545</td>
<td>371</td>
<td>68.0%</td>
</tr>
<tr>
<td>Microdata</td>
<td>221</td>
<td>178</td>
<td>80.5%</td>
</tr>
<tr>
<td>BTC</td>
<td>150</td>
<td>122</td>
<td>81.3%</td>
</tr>
<tr>
<td>Wiki Tables</td>
<td>150</td>
<td>112</td>
<td>74.6%</td>
</tr>
</tbody>
</table>

### Table 3: Accuracy of the header detection algorithm.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Tables</th>
<th># Correct</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tables</td>
<td>545</td>
<td>436</td>
<td>80.0%</td>
</tr>
<tr>
<td>Microdata</td>
<td>221</td>
<td>221</td>
<td>100.0%</td>
</tr>
<tr>
<td>BTC</td>
<td>150</td>
<td>150</td>
<td>100.0%</td>
</tr>
<tr>
<td>Wiki Tables</td>
<td>150</td>
<td>130</td>
<td>86.6%</td>
</tr>
</tbody>
</table>

more sophisticated header unfolding [18], as well as on tables that do not have headers (20% of all tables according to Pimplikar et al. [19]).

### 3.3. Evaluation of the Search Join Operation

For the evaluation of the MSJ Engine, we run several constrained and unconstrained queries covering different topical domains. Table 4 gives an overview of the different query tables that we used for evaluating the engine. The #Rows column shows the number of entities described by each table. The Target Attribute(s) column contains the attributes that we want to add to the tables.

---

**Parameter Values.** For the edit-similarity threshold we chose \( \delta = 0.8 \), and for the fuzzy token similarity threshold we chose \( \tau = 0.5 \). The \( \tau \) parameter was chosen to allow for different levels of brevity in the entity labels (i.e. “coca cola” vs. “the coca cola company”) while ensuring that matches that are too dissimilar are removed.

**Results of the constrained queries.** For constrained queries, the MSJ Engine only joins attributes to the query table with headers containing the specified attribute name. As a result, one final attribute with consolidated values is returned. We run all queries from Table 4 with exact subject matching and similar subject matching (see Section 2). We restrict the engine to only consider the 1 000 top-ranked tables for each query.

We evaluate the result tables with respect to precision and coverage. Coverage is the percentage of the rows of the query table for which a target attribute value is returned. Precision is the percentage of correct values. In order to determine the expected correct values, all film attributes are manually evaluated against IMDB\(^8\). The correct values for all other classes were manually searched on Wikipedia. As both DBpedia [20] and Freebase\(^7\) likely contain the same values as Wikipedia, we removed their data from our corpus to avoid a bias within our evaluation. We treat a numeric value as correct if it does not deviate more than 10% from the reference value.

---

\(^8\) http://www.bbc.co.uk/arts/bigread/top100.shtml
\(^9\) http://www.citymayors.com/features/largest\_cities1.html
\(^10\) http://archive.fortune.com/magazines/fortune/globalmostadmireds/top50/
\(^12\) http://www.txlist.com/script/main/art.asp?articlekey=79509
\(^13\) http://www.listchallenges.com/empire-magazines-500-greatest-films-of-all-time
\(^14\) http://www.songlyrics.com/news/top-songs/all-time/
\(^15\) http://www.theguardian.com/football/datablog/2012/dec/24/world-best-footballers-top-100-list
\(^16\) http://www.imdb.com/
\(^17\) https://www.freebase.com/
Table 4: Query tables and target attributes used for the evaluation.

<table>
<thead>
<tr>
<th>Entity Type</th>
<th>Query Table</th>
<th># Rows</th>
<th>Target Attribute(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book</td>
<td>Britain’s best-loved novels³</td>
<td>100</td>
<td>author</td>
</tr>
<tr>
<td>City</td>
<td>world’s largest cities⁹</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td>global most admired companies¹⁰</td>
<td>50</td>
<td>headquarter, industry</td>
</tr>
<tr>
<td>Country</td>
<td>states with at least partial recognition¹¹</td>
<td>201</td>
<td>currency, population</td>
</tr>
<tr>
<td>Drug</td>
<td>top prescriptions¹²</td>
<td>100</td>
<td>ingredient</td>
</tr>
<tr>
<td>Film</td>
<td>greatest films of all time¹³</td>
<td>100</td>
<td>cast, director, genre, year</td>
</tr>
<tr>
<td>Song</td>
<td>top songs of all time¹⁴</td>
<td>100</td>
<td>artist</td>
</tr>
<tr>
<td>Soccer player</td>
<td>world’s best footballers¹⁵</td>
<td>100</td>
<td>team</td>
</tr>
</tbody>
</table>

Figure 2 shows the precision and coverage results for all constrained queries. The table below the figure shows the number of tables from each corpus that were used to answer the query. The complete query results are available on the MSJ Engine website¹⁸.

Using exact subject matching the coverage ranges between 88% and 100%. For similar subject matching we achieve values between 95% and 100%. The precision ranges between 67% and 100% for exact subject matching and between 54% and 95% for similar subject matching. As expected, the precision is higher for exact matches and coverage is higher for similar matches. An exception is drugs, where the precision is higher for similar matches, because exact search returns too few matches to determine the correct values using majority voting. The precision for similar matches is low for films (especially cast and genre) and books as their titles get easily confused with other films, books, computer games, etc. Since the values for the cast attribute usually contain the names of multiple actors, they are marked as correct if all mentioned actors are in the cast of the movie.

Looking at numeric attributes, a very high precision is achieved for the area of countries but not for their population. A plausible explanation for this difference is the time-dependence of the the two attributes: The area of a country only changes on rare occasions, while the population is continuously changing and a large number of different population values is published on the Web. Another example is the team of soccer players which also changes often and websites thus might report outdated values.

Concerning the number of tables per corpus, we see that the majority of tables always comes from the Wikitables and Web tables corpora. Only for answering the query about country codes a larger number of tables from the BTC and Microdata corpora are used.

Comparing exact and similar subject attribute matching, we observe that similar subject attribute matching returns on average 3.4 times more tables.

Results for unconstrained queries. Unconstrained queries add as many attributes as possible to the query table. Evaluating the coverage and precision of each added attribute would be equivalent to running constrained queries for these attributes. Hence we only report the number of attributes that are added to each query table as well as the origin of the added data. For our experiments, we use exact subject matching, use the top 1000 tables and specify a minimum attribute density of 50%. For the merging of attribute, we set the similarity thresholds to 0.8 for string attributes and to 0.4 for numeric attributes. Table 5 shows the total number of attributes that an unconstrained query adds to each query table. The table also contains the maximum number of attributes of tables from the corpus that were combined into a single output attribute after the schema matching step. The last four columns of Table 5 provide the number of input tables from each dataset that were used to construct the result table.

The largest number of tables for each query originates from the Web table corpus. For the other corpora, we can spot some domains that characterise them. For example, Wikitables are frequently found for countries and films, but not for companies or drugs. In contrast, the Microdata corpus has more tables about companies than about countries. Drugs can be found in the BTC corpus, but no soccer players and only few companies.

4. Related Work

Extending tables with additional attributes given a large corpus of tabular data is a relatively new research
area and only a few systems exist that are directly related to the Mannheim Search Join Engine. These systems can be distinguished according to their goal to either extend a user-provided table with a single additional attribute (constrained queries) or with multiple attributes (unconstrained queries). Research that is related but tries to achieve slightly different goals is the work on (1) table search which aims at finding tables in a large data corpus that are relevant to a user query, but does not aim at further integrating data from the relevant tables with a local table, and (2) knowledge base augmentation which aims at extending a knowledge base describing different types of entities with data from the web. In the following, we will first give a short overview of the research on distinguishing HTML tables that contain structured data from tables that are used for layout purposes. Afterwards, we will discuss the existing work on table extension and will then give an overview of the work on table search and knowledge base extension.

**Content Tables:** Early work on distinguishing HTML tables into quasi-relational content tables and other tables that are for instance used for layout purposes includes the approach of Wang and Hu [11]. Their work combines layout features, such as the standard derivation of the number of columns, and content features, such as the average and standard derivation of the length of the cell content as well as the datatype consistency of the cell content. Cafarella et al. [10, 21] employ a two step approach to content table detection which is very similar to the approach used in this work. First, they filter out extremely small tables, tables that are contained in forms as well as tables that are likely calendars using a set of rules. The remaining tables are afterwards classified using a similar set of layout and content features as proposed by Wang and Hu. Cafarella et al. apply their approach to 12.3 billion HTML tables from the Google crawl out of which 154 million (1.1%) are classified as content tables (precision: 0.41, recall:
An approach for classifying tables into a more fine-grained table type taxonomy as well as an analysis of the different types of HTML tables contained in the Bing crawl is presented by Crestan and Pantel [17].

**Constrained Table Extension**: The goal of extending a local table with an additional attribute based on tabular data from the Web was first formulated by Cafarella, Haley, and Khoussainova in [3]. The proposed EX- TEND operation was implemented in the Octopus prototype. Given a local table and the name of the attribute that should be added to the table, Octopus first queries a search engine for web pages containing subject attribute values from the query table as well as the name of the additional attribute. It then extracts all HTML tables from these pages and clusters them according to their schema similarity. The system then picks the cluster with the highest similarity to the query table and uses values from this cluster to fill the extension attribute. Their evaluation shows an average coverage of 33%. They do not report any precision values for the added data.

A second system that implements constrained table extension is Infogather developed by Microsoft Research [4]. Infogather applies sophisticated schema matching techniques to build a correspondence graph between Web tables in pre-processing. In addition to exploiting schema- and instance-level features, the matching techniques also rely on page URLs and the text outside the table on the HTML pages for matching. This correspondence graph is then used at runtime to find relevant tables. The values from these tables are fused using a voting strategy. Infogather is evaluated using 573 million HTML tables extracted from the Bing crawl. The system reached an average precision of 0.79 and an average coverage of 0.97 on the tasks of extending tables describing cameras, movies, music albums, and politicians. An extension of the Infogather system for handling time-dependent attributes with units of measurement (like company revenues or the population of countries) is presented by Zhang and Chakrabarti in [22]. The basic idea of their approach is to propagate sparse time-stamp and unit of measurement information along the correspondence graph to tables not containing such information.

**Unconstrained Table Extension**: Adding as many attributes as possible to a table (unconstrained table extension) is an interesting operation for exploitative tasks like discovering factors that might explain a certain variable, such as the factors that might explain why the inhabitants of one city claim to be happier than the inhabitants of another [23]. Up to our knowledge, only the WikiTables [15] system implements unconstrained table extension operations beside of the MSJ Engine. The system extends tables with data from 1.4 million Wikipedia tables in order to lay the foundation for finding interesting correlation in the data. The additional attributes are ranked according to their semantic relatedness to the query table. The employed relatedness measure leverages the Wikipedia link structure.

**Table Search**: The goal of table search is to rank tables according to their relevancy for a user query. In contrast to table extension, table search does not aim at the further integration of data from the relevant tables with a local table. An early implementation of table search functionality is also found in the Octopus system [3]. Pimplikar and Sarawagi propose an approach to searching tables using attribute headers as keywords [19]. Venetis et al. [16] propose an approach to recovering the semantics of Web tables by matching them to a large IS-A database extracted from the Web. The semantic annotations are afterwards used to rank tables according to user queries. Das Sarma et al. [24] also exploit an IS-A database to find tables that are related to an input table by either providing additional attributes (schema complement) or additional entities (entity complement). Commercial implementations of table search systems include Google Table Search [19] and Microsoft Power BI [20].

**Knowledge Base Extension**: There is a large body of work on constructing new knowledge bases from Web content as well as to extend existing knowledge bases using Web content [25]. Sekhavat et al. [26] describe a probabilistic method that augments an existing knowledge base with facts from Web tables by leveraging a Web text corpus and natural language patterns associated with relations in the knowledge base. A similar approach is proposed by Fun et al. [27], who develop a two-pronged approach for Web table matching, by linking the attributes to concepts of a knowledge base, and using crowd sourcing to infer the best matches in case of noisy data. Munoz et al. [28] propose that existing knowledge bases, like DBpedia, can be used to semi-automatically extract high-quality facts from tables embedded in Wikipedia articles. Similar, but more general approaches for triplifying tables from the Web have been proposed [29]. Gupta et al. [30] explore the use of Web text and Web tables with combination of query stream data to create an large ontology of binary attributes, called Biperpedia. The Biperpedia contains two orders of magnitude more attributes than Freebase.

---

19. https://research.google.com/tables/
and by this also increases the number of Web tables that match the ontology by more than a factor of 4 compared with Freebase. Dong et al. [31] propose an approach for automatically constructing a Web-scale probabilistic knowledge base that combines data from Web tables with facts that have been extracted from Web texts as well as Microdata annotations. The approach determines the overall quality of a website as well as the effectiveness of the different information extraction techniques and combines both assessments in order to select potentially true values.

5. Conclusion

This article presented the MSJ Engine which extends a local table with additional attributes given a large corpus of Web data. The engine supports constrained and unconstrained queries, with the goal of constrained queries being to extend the local table with a single, user-specified attribute, while the goal of unconstrained queries is to extend the table with as many attributes as possible as a basis for correlation analysis. Our experiments show that the engine is able to produce good results in terms of completeness (coverage close to 100%) and correctness (precision of above 90%) for local tables describing diverse types of entities such as books, cities, companies, countries, drugs, films, songs, and soccer players.

Existing table extension prototypes, such as Octopus [3], Infogather [32], and WikiTables [15], were evaluated using HTML tables from the Web and respectively Wikipedia only. In addition to HTML tables, our data corpus contains tables generated from Linked Data as well as Microdata annotations. Compared to HTML tables, these tables are much larger and thus easier to match to the local table. Using Linked Data and Microdata tables might also increase the consistency of the added data, as each tables covers more entities than HTML tables.

The MSJ Engine demonstrates that given a large corpus of Web data, good results can already be achieved with relatively simple methods, meaning that the corpus contains enough low hanging fruit. In future work we plan to improve the MSJ Engine into the following directions:

Subject Attribute and Header Detection: For subject attribute detection, currently a classification approach similar to the one proposed by Venetis et al. [16] is used. In order to exploit tables with more complex attribute structures, a header detection approach similar to the one proposed by Chen and Cafarella [18] could be employed.

Matching: The matching methods could be improved by using features generated from the page content around HTML tables in addition to the actual table content [32]. We also plan to experiment with matching approaches that more closely combine instance and schema matching [33] and/or use a large cross-domain knowledge base as intermediate schema [30].

Data Fusion: The engine currently does not consider the temporal aspect of data values. Wherever possible, time stamp information should be extracted from the Web content and used by the fusion techniques [22]. An additional simple heuristic to obtain an approximation of data source quality would be to rely on the PageRank of the Web pages from which the data was extracted. More sophisticated data fusion heuristics could aim at in parallel estimating the probability of values being true and the overall quality of data sources [31].

RDF Links and Shared Vocabularies: The providers of Microdata and Linked Data try to ease data integration by using shared vocabularies, such as schema.org or FOAF, and to a certain extend also by publishing correspondences between entity descriptions or schema elements in the form of owl:sameAs and owl:equivalentClass or owl:equivalentProperty links. The Search Join Engine currently does not exploit such integration hints, which we will change in the future.

With the increasing uptake of Semantic Web technologies such as Linked Data [14], Microdata, RDFa, and Microformats [2], the Web is becoming more structured and we believe that it is an interesting challenge to exploit this structure for table extension. There are large corpora of HTML tables, Linked Data, Microdata, RDFa, and Microformats data available to the public (see Section 3.1). Table extension thus does not need to stay a research topic for large Web companies, but every interested researcher is given the chance to work in this area.

More information about the Mannheim Search Join Engine, the source code of the engine, as well as the detailed results of our experiments are found on the project website at http://searchjoins.webdatacommons.org/.

Acknowledgements

The work presented in this paper has been partly funded by the German Research Foundation (DFG) under grant number PA 2373/1-1 (Mine@LOD) and co-funded by the European Commission under grant number ICT-PSP-297274 (DM2E)
References


