Extending Tables with Data from over a Million Websites

Oliver Lehmberg, Dominique Ritze, Petar Ristoski,
Kai Eckert, Heiko Paulheim, and Christian Bizer

University of Mannheim, Germany
Data and Web Science Group
{oli,dominique,petar.ristoski,kai,heiko,chris}
@informatik.uni-mannheim.de

Abstract. This Big Data Track submission demonstrates how the BTC 2014 dataset, Microdata annotations from thousands of websites, as well as millions of HTML tables are used to extend local tables with additional columns. Table extension is a useful operation 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 the headquarter 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. The *Mannheim Search Joins Engine* automatically performs such table extension operations based on a large data corpus gathered from over a million websites that publish structured data in various formats. Given a local table, the Mannheim Search Joins Engine searches the corpus for additional data describing the entities of the input table. 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. Our experiments show that the *Mannheim Search Joins Engine* achieves a coverage close to 100% and a precision of around 90% within different application scenarios.

1 Application Example

Assume a marketing manager who wants to classify the customers of a company according to different properties of the countries in which the customers are located in order to select those that should be targeted by a marketing campaign. While the data about the customers can be found in the company's internal data sources, further background information about the customers' countries is not. Relevant data about countries could for instance include their population, 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 a 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 the relevant data sources that are available on the Web. The *Mannheim Search Joins Engine (MSJ Engine)* supports the manager in reaching his goal by automating the data search and data integration tasks, leaving him his core task.

2 Description

In the following, we first introduce the data corpus that is used by the MSJ Engine (Section 2.1). Afterwards, we explain the general architecture of the engine and provide details about the methods that we employ for data pre-processing, data search, and data consolidation (Section 2.2). Section 2.3 describes how the engine is used via its web interface. Section 2.4 presents the evaluation results using local tables covering books, cities, companies, countries, drugs, films, songs, and soccer players.

2.1 Data Corpus

For our experiments, we use data which is published on the Web either as Linked Data, as Microdata annotations, or as HTML tables. Altogether, the data originates from over a million different websites. We do not gather the data ourselves, but rely on data from the following four datasets:

Billion Triples Challenge 2014 Dataset. As source of Linked Data, we use the Billion Triple Challenge 2014 dataset[1]. It contains around 4 billion RDF quads that were crawled from 47 000 websites in the period between February and June 2014. Please note that we consider a website to be a pay-level domain (PLDs).

WebDataCommons Microdata Dataset. This dataset[2] contains Microdata annotations that were extracted from the 2 billion HTML pages contained in the 2013 version of the CommonCrawl web corpus[3]. The dataset consists of 8 billion RDF quads which were extracted from 463 000 different websites (PLDs). According to Meusel et al. [4], the data strongly relies on the *schema.org* vocabulary and primarily describes products, people, organizations, and events.

WebDataCommons HTML Tables Dataset. Further, we include is the biggest, non-commercial corpus of HTML tables that is available to the public.[4] These tables have been extracted from the 2012 version of the CommonCrawl web corpus which contains 3.3 billion HTML pages. Out of the 11.2 billion HTML tables contained in the crawl, 147.6 million were classified as (quasi-) relational tables containing structured data using similar methods as the ones proposed by Wang et al. [7]. As all our evaluation queries will be formulated in English, we only use the subset of the tables originating from (mostly) English language domains (*com*, *org*, *net*, *eu*, and *uk*). This subset contains 35.7 million tables which originate from 1.5 million different pay-level domains (PLDs).

WikiTables Dataset. In addition to the HTML tables corpus mentioned above, we also use a smaller corpus containing 1.35 million tables that have been extracted from Wikipedia [1]. Although this corpus is a rather small, it contains very valuable data about entities of common interest.

[1] http://km.aifb.kit.edu/projects/btc-2014/

[2] http://webdatacommons.org/structureddata/index.html

[3] http://commoncrawl.org/

[4] http://webdatacommons.org/webtables/

Table 1: Datasets Statistics

Datasets Statistics										
Dataset	# tables	# triples	# cols.	avg. cols.	min. cols.	max. cols.	# rows	avg. rows	min. rows	max. rows
BTC	76K	634M	592K	7.78	2	5,465	18M	244	1	1.2M
Microdata	96K	250M	388K	4.02	1	62	76M	785.96	1	630K
Web	35.7M	2.3B	125M	3.49	3	713	699M	19.52	5	36K
Wiki	1.35M	220M	7.5M	5.34	0	2,349	16M	10.97	0	5K
Preprocessed Datasets Statistics										
BTC	16K	555M	285K	18.5	3	3,942	11M	716	5	340K
Microdata	36K	150M	187K	5.27	3	62	38M	1,056.02	5	50K
Web	35.7M	2.3B	125M	3.49	3	713	699M	19.52	5	36K
Wiki	585K	60M	3.7M	6.3	3	1,000	12M	19.38	5	5K

The MSJ Engine uses tables as internal data model. We thus convert the Microdata and BTC datasets into tables by applying the same procedure that is used by "DBpedia as Tables"[5]. In summary, we first split the datasets by pay-level domain. Non-English or adult content domains are excluded. Afterwards, we create a separate table for each *rdfs:Class* or *owl:Class* and add a column to this table for each RDF predicate that is used by instances of this class.

Furthermore, the tables that we use in our system must comply with the following two conditions: Each table has to contain an *entity label column* (see Section 2.2) and at least 5 rows and 3 columns. Table 1 gives an overview of the size of the data corpora before and after applying these conditions. The second column indicates the number of tables and 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.

2.2 MSJ Engine

Search Joins (SJ) are an approach to extend a local table (called query table) with additional columns from a corpus of other data tables [2]. Given a query table T_q, a SJ performs three operations: a search operation s, a multijoin operation m, and a consolidation operation c. The resulting table R is then computed as a concatenation of the three operators:

$$R = c(m(1, s_{T_q,a}(T))) \tag{1}$$

where T is a set of tables, a is an optional, user-specified parameter for selecting which columns to include. Figure 1 shows how the MSJ Engine implements these SJ in three steps: Table Indexing, Table Search and Data Consolidation.

Table Indexing: The corpus of tables that should be searched (T) is preprocessed: all cell values are normalized, i.e. tokenized, lower cased, values in brackets and stop words

[5] `http://wiki.dbpedia.org/DBpediaAsTables`

are removed. Then, the data type of each table column is identified based on the cell values in the column. Additionally, the engine uses around 200 rules for detecting units of measurements, which are then converted to the corresponding base unit, e.g. *8 sq. km.* will be converted to $8M$ *square meter*. For each table, the engine uses a heuristic to identify the *entity label column* of the tables: If a table contains an *rdfs:label* column, this column will be chosen as entity label column. Otherwise, the column of type string with the highest number of unique values[6] is the entity label column. In cases where two or more columns contain equally high numbers of unique values, the left-most column of those is the entity label column. With this simple approach, an accuracy of about 83% can be achieved [5]. In the final step of the preparation, the entity labels and column headers of each table are stored in a Lucene index[7].

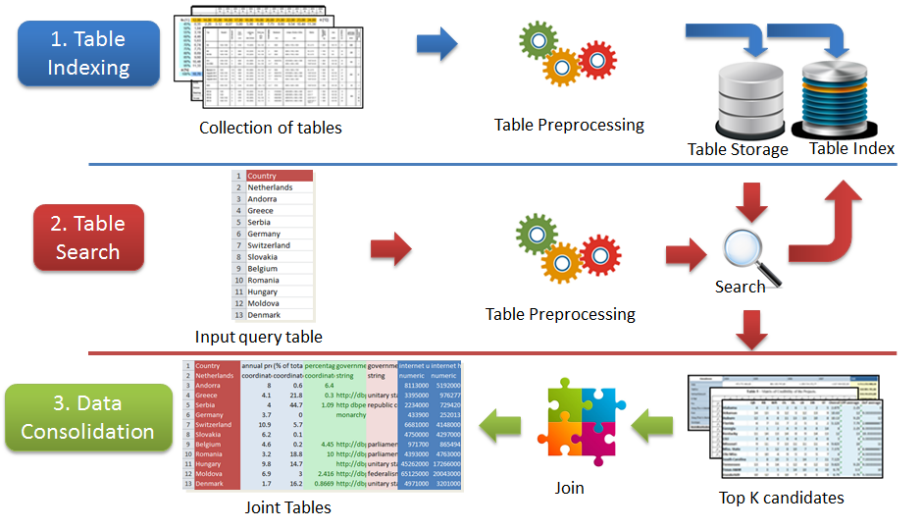


Fig. 1: Search Joins Process Overview

Table Search: The query table T_q is preprocessed in the same manner. Then, the search operator s is applied and tries to find matching entity labels in the previously indexed tables. For deciding whether an entity label value from a table matches an entity label value from the query table, two different methods are available: exact entity label matching, and similar entity label matching using the FastJoin matcher [6].[8] All tables with at least one overlapping entity label are returned and ranked by the number of overlapping entity labels and their resp. similarity. The search can either be unconstrained to find all kinds of columns, or constrained to specific columns, e.g. only columns containing information about the population of countries.

Data Consolidation: After the search is completed, the tables are joined by applying the multijoin operation m, which performs a series of left outer joins between the query

[6] At least 60% of the values must be unique

[7] `http://lucene.apache.org/`

[8] We use the FastJoin parameters $\delta = 0.8$, $\tau = 0.5$, and accept matches with a confidence of at least 0.5.

table and the tables returned by the search. Afterwards, the consolidation operation c combines columns that represent the same property. The type of method applied depends on whether a constrained or unconstrained query is executed.

A *constrained query* considers only columns matching a header specified by the user. The heuristic we apply here is to accept all column headers that contain the given header. For example, if the user queries for "GDP", columns with header "GDP total" and "GDP (US$)" are also matched. After the filtering, the remaining columns are consolidated to a single column using a majority vote for each entity label.

An *unconstrained query* returns all columns without filtering to get as much information as possible. Thus, we only merge columns which contain overlapping information, determined by a combination of label- and instance-based schema matching techniques. For the actual merging of the values, the user can choose from different conflict resolution strategies [3]. Within our experiments, we use voting for resolving conflicts between string values and the median function for combining numeric values.

2.3 Web Access

The functionalities of the MSJ Engine can be easily integrated in any application via its API. For demonstration purposes, we provide a web interface[9] that allows users to explore our system online. Users can run queries against the complete corpus with the possibility for custom configuration of the MSJ Engine. Please note that running a query can take several minutes due to the size of the data corpus. Furthermore, the website contains the pre-computed results of the queries that we use for the evaluation.

2.4 Evaluation

Table 2 gives an overview of the different query tables that we used for evaluating the engine. The *#Rows* column shows the size of the tables. The *Properties* column contains the target properties that we use for constrained queries. In order not to over-simplify the task, we removed DBpedia and Freebase from our corpus for the evaluation. Note that we only use the top $1\,000$ tables returned by the search operation.

Results for constrained queries. For constrained queries, the MSJ Engine only joins columns to the query table with headers containing the specified property name. As result, one final column with consolidated values is returned. We manually evaluated the results with respect to precision (percentage of correct values) and coverage (percentage of entity labels for which we find a value). All *film* properties are evaluated against

[9] `http://searchjoins.webdatacommons.org/`

[10] http://www.bbc.co.uk/arts/bigread/top100.shtml

[11] http://www.citymayors.com/features/largest_cities1.html

[12] http://archive.fortune.com/magazines/fortune/globalmostadmired/top50/

[13] http://www.polgeonow.com/2011/04/how-many-countries-are-there-in-world.html

[14] http://www.rxlist.com/script/main/art.asp?articlekey=79509

[15] http://www.listchallenges.com/empire-magazines-500-greatest-films-of-all-time

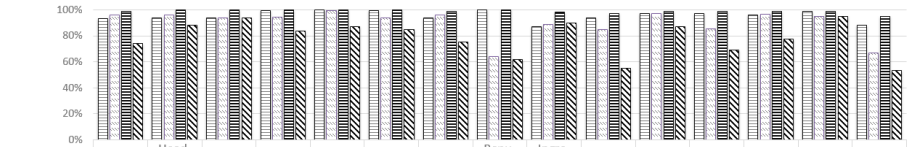
[16] http://www.songlyrics.com/news/top-songs/all-time/

[17] http://www.theguardian.com/football/datablog/2012/dec/24/world-best-footballers-top-100-list

Table 2: Query Tables used for the Evaluation

Class	Description (Source)	# Rows	Properties
book	Britain's best-loved novels[10]	100	author
city	world's largest cities[11]	100	
company	global most admired companies[12]	50	headquarter, industry
country	states with at least partial recognition[13]	201	currency, population area, capital, code
drug	top prescriptions[14]	100	ingredient
film	greatest films of all time[15]	100	cast, director, genre, year
song	top songs of all time[16]	100	artist
soccer player	world's best footballers[17]	100	team

`imdb.com`, for all other classes we use Wikipedia[18]. Figure 2 shows the results for both exact and similarity matching of entity labels.



	Author	Head-quarter	Industry	Area	Capital	Code	Currency	Popu-lation	Ingre-dient	Cast	Director	Genre	Year	Artist	Team
	Book	Company		Country					Drug	Film				Song	Soccer Player
cov (exact)	93%	94%	94%	100%	100%	100%	94%	100%	87%	94%	97%	97%	96%	99%	88%
prec (exact)	96%	96%	94%	95%	100%	94%	96%	64%	89%	85%	97%	86%	97%	95%	67%
cov (similar)	99%	100%	100%	100%	100%	100%	99%	100%	98%	97%	99%	99%	99%	99%	95%
prec (similar)	74%	88%	94%	84%	87%	85%	75%	62%	90%	55%	87%	69%	78%	95%	54%

Fig. 2: Precision and coverage for constrained queries

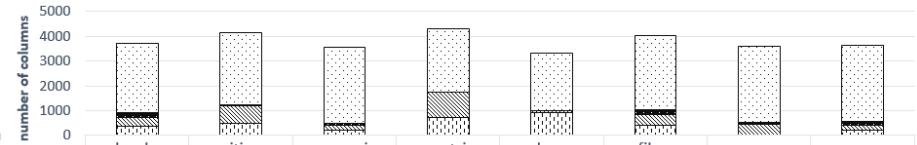
For all queries, the coverage ranges between 88% (exact) resp. 95% (similar) and 100%. The precision ranges between 67% and 100% for exact entity label matching and between 54% and 95% for similar entity label matching. As expected, precision is higher for exact matches and coverage is higher for similar matches. An exception is *drugs*, where precision is higher for similar matches, because exact search returns less values which results in the decision for incorrect values of the majority vote. 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* property 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 properties, a very high precision can be achieved for *area* but not for *population*. We treat a numeric value as a correct match if it does not deviate more than 10% from the reference value. The difference between area and population is explained by their variability over time: the area of a country only changes on rare occasions, while the population is continuously altering, which leads to a number of

[18] Note that we do not use the Wikitables that are included in our corpus.

different values found. Another example is the *team* of *soccer players*, which can also change often and many websites might not have been updated yet.

Results for unconstrained queries. For unconstrained queries, the MSJ Engine adds as many columns as possible to the query table. As the result contains all kinds of properties, it depends on the use case which columns are determined as relevant and useful, thus we do not apply measures like precision and coverage. Instead, we evaluate the amount of data joined to the query table. This evaluation is shown in Figure 3. For each query, the diagram shows the overall number of columns that were added to the table. The numbers show how many columns were added from each dataset.



	books	cities	companies	countries	drugs	films	soccer	songs
Web	2773	2905	3079	2528	2327	2992	3102	3043
micro	209	43	62	5	16	180	59	187
Wiki	342	688	191	1026	87	443	453	198
BTC	380	499	223	737	905	406	0	201

Fig. 3: Number of added columns for unconstrained queries

For nearly all of the queries, we can find meaningful information in each of the four data corpora, but the amount of added columns per corpus differs: For example, Wikitables contain a lot of information about countries and cities, the BTC dataset offers a large amount of information about drugs.

In summary, our experiments show that (1) we can extend local tables with meaningful information coming from over a million of websites, and (2) we achieve high precision and coverage for constrained queries.

3 Challenge Criteria

We believe that the *Mannheim Search Joins Engine* fits the primary goal of the Big Data Track to demonstrate approaches that can work on web-scale using realistic web-quality data. In the following, we will discuss how we meet the specific sub-criteria.

Data Volume Our system combines four large data corpora, including the BTC2014 dataset, the largest non-commercial corpus of HTML tables, and the largest Microdata corpus that is currently available to the public. Altogether, our system indexes 3 billion triples which are structured into 36 million tables.

Data Variety The data corpus is highly heterogeneous concerning the original data formats and the employed schemata as the data originates from over a million different websites. In general, the MSJ Engine can work with any data that can be converted into a tabular representation.

Data Velocity As Lucene supports live index updates our application can in theory handle update streams that would be generated by continuous web crawling. Note that our current implementation does not employ update streams but works with the data corpora mentioned in Section 2.1.

The application should do more than simply store/retrieve large numbers of triples. The key features of our MSJ Engine are the search for and ranking of tables matching a given query table, the subsequent join operation extending the original table and the final data consolidation steps that employ schema matching and data fusion methods in order to merge values for the same property from different sources.

The application or tool(s) should be scalable Currently, the MSJ Engine runs on a single machine. Components critical for the scalability of the MSJ Engine are the index and the query processor. The index infrastructure that we use is Lucene for which distributed implementations are available and could be used to scale the index. Concerning the query processor, two scaling scenarios are possible: On the one hand, if a large user base needs to be served, a simple load balancing between multiple query processors will be sufficient. On the other hand, if query sizes increase, each query can be split into multiple parts, which are then processed on several machines in parallel.

The application should either function in real-time or, if pre-computation is needed, have a real-time realization Running a table extension query against our current corpus on a single machine takes several minutes. Given multiple machines, this the response time could be significantly reduced (see bullet point about scalability). Nevertheless, waiting several minutes is already much shorter than the time it would take to manually search and integrate data from a large set of websites.

References

1. C. S. Bhagavatula, T. Noraset, and D. Downey. Methods for Exploring and Mining Tables on Wikipedia. In *Proc. of the ACM SIGKDD Interactive Data Exploration and Analytics (IDEA)*, 2013.
2. C. Bizer. Search Joins with the Web. In *Proc. of the 17th Int. Conf. on Database Theory (ICDT)*, 2014.
3. J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.
4. R. Meusel, P. Petrovski, and C. Bizer. The WebDataCommons Microdata, RDFa and Microformat Dataset Series. In *Proc. of the 13th Int. Semantic Web Conference (ISWC14)*, 2014.
5. P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering Semantics of Tables on the Web. *Proc. of VLDB Endow.*, 4(9):528–538, 2011.
6. J. Wang, G. Li, and J. Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *Proc. of the 27th Int. Conf. Data Engineering (ICDE)*, pages 458–469, 2011.
7. Y. Wang and J. Hu. Detecting Tables in HTML Documents. In *Proc. of the 5th Int. Workshop on Document Analysis Systems V*, 2002.