Matching HTML Tables to DBpedia

Dominique Ritze
Data and Web Science Group,
University of Mannheim,
Mannheim, Germany
dominique@informatik.uni-mannheim.de

Oliver Lehmberg
Data and Web Science Group,
University of Mannheim,
Mannheim, Germany
oli@informatik.uni-mannheim.de

Christian Bizer
Data and Web Science Group,
University of Mannheim,
Mannheim, Germany
chris@informatik.uni-mannheim.de

ABSTRACT

Millions of HTML tables containing structured data can be found on the Web. With their wide coverage, these tables are potentially very useful for filling missing values and extending cross-domain knowledge bases such as DBpedia, YAGO, or the Google Knowledge Graph. As a prerequisite for being able to use table data for knowledge base extension, the HTML tables need to be matched with the knowledge base, meaning that correspondences between table rows/columns and entities/schema elements of the knowledge base need to be found. This paper presents the T2D gold standard for measuring and comparing the performance of HTML table to knowledge base matching systems. T2D consists of 8 700 schema-level and 26 100 entity-level correspondences between the WebDataCommons Web Tables Corpus and the DBpedia knowledge base. In contrast related work on HTML table to knowledge base matching, the Web Tables Corpus (147 million tables), the knowledge base, as well as the gold standard are publicly available. The gold standard is used afterward to evaluate the performance of T2K Match, an iterative matching method which combines schema and instance matching. T2K Match is designed for the use case of matching large quantities of mostly small and narrow HTML tables against large cross-domain knowledge bases. The evaluation using the T2D gold standard shows that T2K Match discovers table-to-class correspondences with a precision of 94%, row-to-entity correspondences with a precision of 90%, and column-to-property correspondences with a precision of 77%.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE—*Learning, Knowledge acquisition*

General Terms

Algorithms, Experimentation

Keywords

html tables, tables matching, knowledge base extension

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Wims 2015 Limassol, Cyprus
Copyright 2015 ACM X-XXXXX-XX-X/XX/XX ...$15.00.

1. INTRODUCTION

Cross-domain knowledge bases (KBs) like DBpedia [8], YAGO [15] or the Google Knowledge Graph [13] play an increasingly important role in web search, question answering, natural language processing, as well as data mining. Ideally, they are as complete and sound as possible.

The huge amount of relational HTML tables [2] that is available on the Web suggests itself for filling missing values in cross-domain KBs ("Slot-filling"). Finding missing values in multiple HTML tables opens up the possibility to apply data fusion methods which assess the trustworthiness of each source and determine the most likely fact in cases of contradicting values [4, 1]. The slot filling process consists of two parts: first, HTML tables containing missing information need to be identified by matching the tables with the KB and second, all candidate values need to be fused to obtain the value that is most likely correct. In this paper, we focus on the first step of the process, but also estimate the utility for filling missing values in DBpedia.

We first present the T2D gold standard for measuring the performance of HTML table to KB matching systems. The gold standard was manually created and consists of 8 700 schema-level and 26 100 entity-level correspondences between the WebDataCommons (WDC) Web Tables Corpus[1] and the DBpedia knowledge base. All parts of the gold standard are publicly available and can thus be used to compare matching systems. Afterward, we present T2K Match, an iterative matching approach which combines schema and entity matching and is designed for the use case of matching large quantities of HTML tables against a cross-domain KB. By solving both matching problems iteratively, the algorithm can benefit from entity-correspondences to find schema-correspondences and vice versa.

The paper is structured as follows: First, we introduce the characteristics of the HTML table corpus (Section 2) and present the T2D gold standard (Section 3). Section 4 describes T2K Match which is evaluated in Section 5. In Section 6, we discuss related work. Finally, we summarize our results and show directions for future work (Section 7).

2. THE WDC WEB TABLES CORPUS

The WDC Web Tables Corpus[2] is the largest non-commercial corpus of (quasi-)relational HTML tables that is available to the public. It has been extracted from the 2012 version of

[1] `http://www.webdatacommons.org/webtables/`
[2] `http://webdatacommons.org/webtables/`

the CommonCrawl Web corpus[3] which consists of 3.5 billion HTML pages originating from 43M different websites (PLDs). Altogether, the 2012 version of the corpus contains over 11 billion HTML tables. For building the WDC Web Tables Corpus, several heuristics were applied to distinguish between relational and layout tables, e.g. all tables containing other tables were excluded. Further, a classifier exploiting layout and context features (similar to the features proposed by Wang et al. [17]) was used to detect relational tables. After both steps, 147.6 million tables were classified as relational tables. This corresponds to 1.3% of the complete corpus which is inline with the results of Cafarella et al. [2] that 1.1% of all HTML tables in a Google crawl contained relational data.

3. THE T2D GOLD STANDARD

The T2D Gold Standard[4] was developed with several goals in mind. We required a realistic ratio of mappable and unmappable HTML tables as many relational HTML tables contain data that cannot be matched with DBpedia. Further, characteristics like sizes, coverage of various topics, and diverse matching behavior, e.g. the number of attributes that can be matched to DBpedia properties should be balanced.The gold standard consists of two parts:

The Schema-level Gold Standard contains 1 748 tables of which 762 can be matched with DBpedia classes and 7 983 columns which correspond to DBpedia properties. The tables were randomly selected from the relational tables in the WDC corpus.

The Entity-level Gold Standard contains a subset of 233 tables from the schema-level gold standard. In total, it includes 26 124 row-to-entity correspondences to DBpedia resources, 2 523 rows were marked as not matchable. To find annotations on entity-level, the tables need to include at least one correspondence to a DBpedia resource.

All correspondences were created manually by a team of two human annotators, requiring altogether about six person weeks. The distribution of the attribute data types in the schema-level gold standard is similar to the distribution of data types in the whole corpus, i.e. 68% strings, 27% numeric. To give an overview on the topics, we grouped the tables by their corresponding super class in DBpedia, resulting in seven categories. For example, the category "Organization" contains tables about companies, universities and political parties. Figure 1 shows the number of tables and instance correspondences per category for the entity-level gold standard.

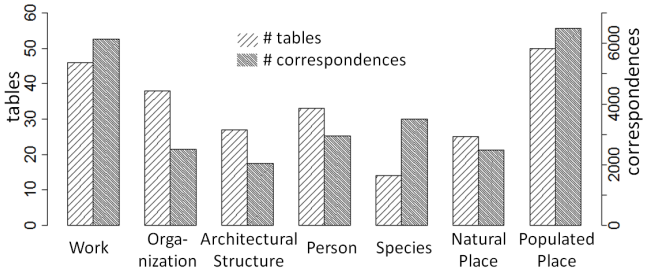

Figure 1: Number of class and entity correspondences per category.

[3] http://commoncrawl.org/

[4] Project page with download: http://webdatacommons.org/webtables/goldstandard.html

The average number of entity correspondences for tables in the entity-level gold standard ranges from 65 (organizations) to 250 (species) per table. The average number of property correspondences ranges from 1.5 (person) to 4.1 (populated place). Overall, the entity-level gold standard contains 1 153 columns whereof 653 are mapped to a DBpedia property (117 distinct properties).

4. T2K MATCH

This Section describes the T2K Match[5], a matching method for iterative instance and schema matching. After describing the data model and preprocessing, we introduce the major steps of the algorithm.

Data Model: As internal data model, we use simple entity-attribute tables. Each table describes a set of entities (rows in HTML tables) having a set of multi-valued attributes (columns). Each attribute has a header which we assume to be some surface form of the attribute's semantic intention. We distinguish between different attribute data types like strings and numerical values. Further, we require that tables contain an attribute with natural language labels (the entity label attribute) for the entities that they describe (e.g. New York, Barak Obama).

Preprocessing: In order to match HTML tables, we first have to clean them, e.g. remove HTML artifacts, special characters and additional whitespaces. Further, value lists are split into individual values, all values are lower-cased and normalized. For the normalization, we use a set of handcrafted transformation rules to resolve abbreviations, e.g. "co." is transformed into "company". Further, we normalize units of measurements using around 200 manually generated conversion rules, e.g. $8mi^2$ is converted to 20.72 million m^2.

Next, we detect the entity label attributes. For HTML tables, we apply the heuristic of taking the string attribute with the highest number of unique values as entity label attribute (in case of a tie, the left-most attribute is used). Attributes with values having on average less than four characters are excluded, e.g. IATA airport codes. In DBpedia, the entity label attribute is the attribute containing the labels (rdfs:label) of the resources. Our strategy for detecting attribute headers is similar, albeit simpler: we assume that the attribute header is the first non-empty entity. In DBpedia, the attribute header corresponds to the property label. An evaluation of the performance of our header and entity label attribute detection is presented in Section 5.

We further perform a data type detection for each attribute which is important for choosing the similarity measure. The data type is detected using about 100 manually defined regular expressions. They are able to detect string, numeric value, timestamp and coordinate. The final data type decision for an attribute is based on majority vote.

Figure 2 depicts the mayor steps of the T2K Match algorithm.[5] First, we determine a set of candidate resources from DBpedia to reduce the search space and then calculate value-based similarity scores with the entities from the HTML table. These similarities are used to determine an initial schema-level correspondence set which is in the last step iteratively refined until we obtain the final table-

[5] Project page with examples and download: http://dws.informatik.uni-mannheim.de/en/research/T2K

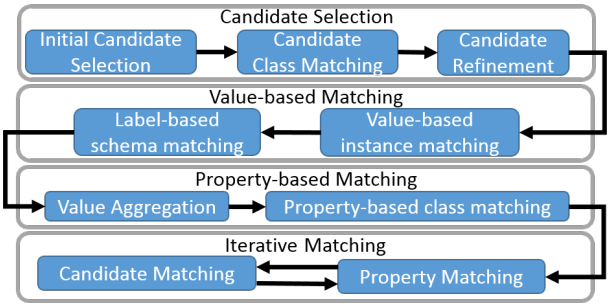

Figure 2: Matching Steps of the Matching Approach

to-class, attribute-to-property and entity-to-resource correspondences. During the whole process, the entity- and schema-level matching mutually influence each other, as one is used to weight the similarities of the other. In the following, we describe the four major steps of the algorithm in more detail:

1. Candidate Selection: The goal of the candidate selection is to determine a set of candidate resources from DBpedia for each entity in the HTML table. First, we search for the entity label in DBpedia. The found candidates are ranked according to a similarity function and the top k candidates are kept. Then, we determine the distribution of DBpedia classes of the best candidate for each entity and choose the most frequent classes as candidates for schema matching. In turn, this is used to refine the candidate resources: All candidates not belonging to a chosen class are removed, and for each entity we perform another search with the selected classes as additional constraint.

2. Value-based Matching: We compute similarities between the values of HTML table entities and candidate resources by applying two blocking strategies: (1) the values of each entity are only compared to the values of its candidates and (2) only values with the same data type are compared. In case of multi-values, we calculate the similarity of all combinations and choose the maximum.

3. Property-based Matching: In this step, we exploit the computed value-based similarities for schema matching by aggregating them per attribute. Such an aggregation is usually referred to as duplicate-based schema matching [11]. Each value of the top candidates votes for a correspondence between the attribute and its property. This vote is weighted by the value-based similarity of the two values and the similarity value from the candidate selection step. Votes from all values are summed up and the attribute property pair with the highest value is chosen.
It follows the intuition that a similar attribute property pair has many similar values on similar entities/candidates. Our computation is purely duplicate-based and does not perform any label matching on the headers since headers in HTML tables do not often have meaningful names. The matching of attributes with properties is further aggregated by summing up all the scores of all property correspondences per class to refine the class ranking. At this point we choose the class with the highest score as final correspondence. Again, all candidates which do not belong to this class (or its super classes) are removed. The idea behind it is that candidates from multiple classes match the entity labels from the HTML table, but it is a strong signal if additionally properties of a class have overlapping values with the entities.

4. Iterative Matching: In the first iteration, the property matching is repeated. This is necessary as the list of candidates has changed and thus the value-based similarity matrix has become smaller. Afterward, the value-based similarities are weighted with the property matching scores. These weighted similarities are then aggregated for each entity/candidate pair and the candidate with the highest score is chosen as correspondence for each entity. To prevent false positives, a minimum similarity threshold is applied. In the following iterations, the entity matching scores are used to weight the value-based similarities for the property matching, and the property matching scores are used as weight for the entity matching. The algorithm terminates when the similarities are not changing.

After the iterative part, we transform the similarities between attributes/properties and entities/candidates into final correspondences by choosing the pair with the highest similarity. Note that while each attribute/entity is only mapped to one property/resource, it is possible that multiple attributes/entities map to the same property/resource, as HTML tables can and often do contain duplicates.

In addition to the basic algorithm, we implement several extensions that improve the result by specifically addressing the idiosyncrasies of the data that we are matching.

Surface Form Handling: As the values in an HTML table do not need to be same as the labels of resources from DBpedia, we search a surface form catalog for each value from the HTML table. If we find a resource having the value as surface form, we add the label of this resource as a value to the HTML table. The list handling of our algorithm will then take care of the rest. If multiple resources are found, we either (1) take the one with the highest score if it is much better than the second best score or (2) we take the top three otherwise. The catalog was created from anchor-texts of intra-Wikipedia links, Wikipedia article titles and disambiguation pages. The score is the TF-IDF score with all surface forms of a certain resource forming a document.

Redirects: A similar strategy as for the surface forms is applied for redirects in DBpedia. If an HTML table has a value that exactly matches the label of a resource that redirects to another resource, the label of the redirect target is added as a value to the HTML table.

Kurtosis Filter: HTML tables tend to have attributes containing row numbers or rankings. Such attributes cannot be mapped to DBpedia as they are only meaningful in the context of HTML tables. As distributions of such attribute values usually follow a normal distribution, we filter them out by ignoring numeric attributes with low kurtosis value.

5. EVALUATION

This Section presents the evaluation of T2K Match as well as our entity label attribute and header detection heuristics. First, we describe the subset of DBpedia we match with and the parameter setting used for the experiments. Afterward, we describe how the T2D gold standard was used to evaluate T2K Match and compare the algorithm with two baseline approaches. Finally, we discuss the results.

All experiments have been performed on a Linux machine

with 8 cores and 130GB RAM. To improve the performance, we keep DBpedia in-memory and use a Lucene[6] index for the candidate selection step. The pruning as well as the candidate and type blocking in combination with sparse similarity matrices improve the performance further.

Reference KB: For the experiments, we use a subset of DBpedia (version 2014) as reference KB. This subset consists of all classes, properties and resources from DBpedia that are frequently used and hence are good candidates for the ultimate goal of our use case. Only classes and properties from ontology namespace[7] are considered. We use all classes as long as they have at least 1 000 entities. To exclude too specific classes, we only consider classes up to the fourth level of the class hierarchy. As the properties are not necessarily used only for the classes they define as domain, we include a property for a class if at least 5% of the resources belonging to that class have a triple using this property. The result of this selection process is a DBpedia subset covering 94 classes, 1 393 properties and about 3M resources.

Parameter Setting: In our algorithm, about 20 parameters are included, e.g. different similarity functions for the data types, the top k parameters for candidates or weights for properties. We implemented a genetic algorithm to find the best possible parameter values. To avoid a bias in the evaluation, we apply split validation. Therefore, we divide the entity-level gold standard into two equally sized, stratified sets: an optimization set and an evaluation set. The optimization is run on the optimization set to determine the best parameter setting. Then, we run our algorithm on the evaluation set to see the performance on data that was not optimized for. As final setting, we use Jaccard similarity for the candidate selection with an initial k of 50 and a refinement k of 100. The value-based similarity functions are Generalized Levenshtein for strings, deviation similarity (according to Rinser et al. [11]) for numerical values and a similarity for dates that computes the deviation of the years. The weight for the entity label attribute is set to five, which indicates that the entity label is quite important. The whole configuration can be found on the project website.

5.1 Entity Label Attribute and Header Detection Results

It is crucial for the matching process that entity label attributes and headers are correctly detected. If we choose the wrong entity label attribute, we will select the wrong candidates which leads to false class/property correspondences. As we do not use the attribute labels yet, an incorrectly detected header is not as critical as a false entity label attribute. The entity label attribute is correctly detected in 97%. Incorrectly classified entity label attributes are for example multiple attributes containing the label in different languages. 93% of the headers are correctly detected. The main problem with header detection is that we always choose a header, even if no header is present in the HTML table.

5.2 Matching Results

The results of T2K Match on the entity-level gold standard are shown in Table 1. While F1 (opt.) indicates the F1-Score that was achieved on the optimization set, all other results indicate the performance on the evaluation data. All values are micro averaged to get a realistic valuation.

Table 1: T2K Match Evaluation

Task	Precision	Recall	F1	F1 (opt.)
Entities	0.90	0.76	0.82	0.86
Properties	0.77	0.65	0.70	0.73
Classes	0.94	0.94	0.94	0.97

The precision for entity correspondences is quite high with 0.9, which is important for our ultimate goal of extending KBs. The more precise our correspondences are, the less incorrect data we have to deal with in the data fusion step. For property correspondences we achieve a precision of 0.77. This is mainly due to constellations where the pure value-based similarity is misleading. For example, an HTML table attribute "language" (value "German") is more similar to the DBpedia property "demonym" (with value "German") than to the property "language" (with value "German language"). Finally, the class correspondences achieve a precision of 0.94. An example for an incorrectly chosen class is a table that contains different types of persons where our algorithm decides for the majority more-specific person class.

Figure 3 shows the precision for each of the topical categories of the entity-level gold standard.

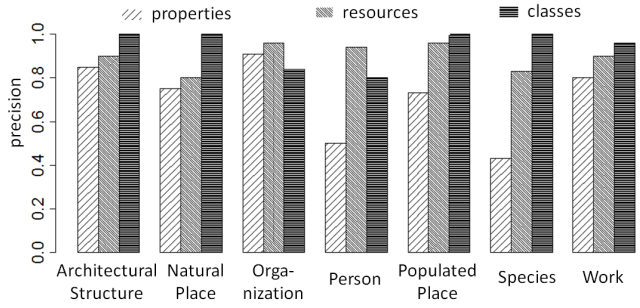

Figure 3: Precision of different categories

The reasons for different category results can be very specific, thus we only present some examples. The low property precision for the person category stems from varying attribute values, e.g. the weight of an athlete might change during his/her career and the birth or death dates of historic persons are often different from source to source.

The better performance for the entity correspondences has also different reasons. In some cases, a high weight on the entity label attribute in combination with a class constraint is sufficient to find the correct correspondences. But, for example, for companies, we can see that the precision for tables where no properties can be mapped is only 0.45 and for tables where we find one property correspondence, it increases to 0.97. Hence, even a single, mappable property provides us with a strong signal that our algorithm makes use of the entity matching task. However, such a signal is not enough for ambiguous entities, e.g. the natural places "lake geneva" is mapped to `Geneva_Lake` instead of `Lake_Geneva`. Another problem is related to our concept of entity label attributes: tables about species often use inconsistent resource naming, sometimes the Latin name otherwise the English name is used as entity label.

5.3 Comparison with Baselines

In order to set our results into context, we compare our entity correspondence results with two baseline approaches.

[6] `http://lucene.apache.org/core/`
[7] `http://dbpedia.org/ontology/`

String matching: Uses the best candidate for each entity from the first search operation of our candidate selection. It serves as a reference value for the improvements achieved by the following steps.
DBpedia Lookup: Queries the DBpedia Lookup service[8] for each key value and choose the first result as correspondence. In addition to relying on string comparison, it considers the link indegrees and thus emulates the strategy to always match the most common sense for each string.

Our algorithm outperforms the baselines concerning precision, recall, as well as F1 score (Table 2). Both approaches which purely base on the entity label are not as precise as our algorithm which indicates that the combination of value- and property based matching is indeed useful.

Table 2: Entity Correspondence Comparison

Approach	Precision	Recall	F1
String Matching	0.53	0.53	0.53
DBpedia Lookup	0.79	0.73	0.76
Our Approach	**0.90**	**0.76**	**0.82**

Challenges: Having a closer look at the results, we were able to identify future challenges. For example, some properties are hard to distinguish, i.e., airports in the United States almost always have the same IATA as FAA code. If we encounter a table that only contains airports from the United States, choosing the correct property is purely random as both will have the same similarity score. Another example stems from the nature of our use case: If the resources in DBpedia that match the entities in the HTML table have no property values, we cannot compute a similarity and will hence not map these columns. The same holds the other way around. If the only matching attribute of an HTML table is the entity label attribute, all we can do is to apply string matching and determine the most frequent class. Also, such tables are not directly useful for our use case as they do not provide us with missing values. However, they could be used for set completion tasks (i.e., adding resources to classes) or for the creation of new properties.

5.4 Slot Filling Potential

In order to get an impression of the potential of HTML tables for slot filling, we calculated the number of facts that we could add to DBpedia from the 233 HTML tables in the entity-level gold standard. For this, we counted all values that are mapped correctly (correct class, property and resource correspondence) and for which DBpedia has no value. Out of all 233 HTML tables, we find 5 178 facts that are not in DBpedia where 2 691 of them are distinct facts (not having the same resource and property). These 2 691 facts concern 2 469 different resources and 52 different properties. It shows that we can already find a lot of values that are not yet in DBpedia by considering this very small subset of the whole corpus. Using the corpus with 35M HTML tables would thus likely result in an enormous number of additional facts that are currently missing in DBpedia.

6. RELATED WORK

A number of approaches have been proposed towards understanding and extracting relational HTML tables from the Web [2, 18]. Recent studies have shown that the knowledge extracted from HTML tables can be useful for applications like table search [16], table extension [20, 3], and KB augmentation [17, 12].

Other Gold Standards: Many of the gold standards about matching HTML tables to KBs mentioned in publications are not publicly available. One well-known gold standard has been created by Limaye et al. [9] and maps tables to YAGO. It consists of a manually created subset with around 400 tables (general HTML tables as well as Wikipedia tables) with 10 930 entity, 747 class and 54 relation correspondences. Since the gold standard of Limaye et al. is not publicly available, Zhang [22] is using the same tables but maps them to Freebase. The resulting gold standard, called Limaye112, covers 112 tables (about 10% HTML tables and 90% Wikipedia tables) with manually annotated colunm and automatically created entity correspondences. Other gold standards have been automatically extracted from IMDB or MusicBrainz [21].

Matching HTML tables: Our work is similar to that of Limaye et al. [9] which uses a probabilistic graphical model comprising several features, trying to learn the best combination with weights. As use case, they describe table annotations on a web search tool designed to complete missing fields in binary relations. Applied on a part of the Limaye gold standard (371 HTML tables), an F1-score of 0.81 for resource, 0.43 for class and 0.52 for relation correspondences is achieved. Similar work in that direction was also done by Mulwad et al. [10], who used the Wikitology KB, and Venetis et al. [16], who used an isA database. Zhang [22] introduces an instance-based schema matching approach to find mappings to resources which are then used to map each of the columns to a class in the KB. Other than our algorithm, this approach maps columns to classes (not properties) and single cells to resources, so no properties are identified and no rows are mapped to entities, which is a crucial step for KB augmentation. On the Limaye112 gold standard, they achieved an F1 score of 0.92 for cell correspondences and 0.63 for column correspondences. Since most of the tables in Limaye112 originate from Wikipedia, these tables are broader and large as well as cleaner as usual HTML tables. Sekhavat et al. [12] describe a probabilistic method that augments an existing KB with facts from HTML tables by levering a web text corpus and natural language patterns associated with relations in the KB. Similarly, Fan et al. [6] propose a two-pronged approach for HTML table matching. They link the columns to classes by using crowd sourcing. Other approaches focus on the creation of KBs or ontologies by exploiting HTML tables. Dong et al. [4] propose an approach for automatically constructing a web-scale probabilistic KB that combines extractions from HTML tables with prior knowledge derived from existing knowledge repositories. Similar, Gupta et al. [7] explore the use of web text and HTML tables combined with query stream data to extract an ontology of binary attributes, called Biperpedia. Further extensive work on automatic KB construction from web sources has been summarized by Weikum et al. [19].

Ontology Alignment: Performing simultaneous instance and schema matching has already been applied in previous work with focus on matching ontologies. The PARIS system by Suchanek et al. [14] combines instance and schema

[8] `http://wiki.dbpedia.org/lookup/`

matching using probabilities. In contrast to our approach, PARIS is designed for matching ontologies with a rich class and relation structure. This is often not the case for HTML tables which can be very noise and usually cover only a few attributes. We run PARIS on our evaluation dataset but this did not produce meaningful results. Duan et al. [5] focus on the scalability of schema matching. They apply their instance-based type matching to determine containment and equivalence relations between two ontologies using locality-sensitive hashing as blocking strategy. In theory, their approach is also applicable for HTML tables, but it is to be expected that it will perform better on larger tables.

7. CONCLUSION

In this paper, we present the first public gold standard T2D for matching HTML tables from a large scale, publicly available web corpus to DBpedia. Further, we presented the T2K Match algorithm that performs reasonably well on this gold standard. Using the results, we were able to explore the potential of HTML tables for the task of filling missing values KBs. Even with the small sample used for evaluation, we produce a fair amount of facts that stem from different sources and are missing in the current version of DBpedia.

Current ideas for further improving T2K Match are to adjust the property weights during the iterative step and to exploit the abstract from DBpedia. Another direction for improvement is to learn property recognizers for DBpedia. By this we specialize on DBpedia as our target KB and should improve on the property matching scores. We plan on experimenting with candidate consistency measures which could for example, compare the values of other properties (independently from whether they can be mapped or not) to find sets of candidates that are more consistent than others.

As the next step towards the goal of our use case, we will run the matching algorithm on the complete WDC Web Tables Corpus and use matching table data to fill missing values in DBpedia. This will include further work on the supervised learning of conflict resolution functions [1].

8. REFERENCES

[1] V. Bryl and C. Bizer. Learning conflict resolution strategies for cross-language Wikipedia data fusion. In *Proc. of the 23rd Int. Conf. on World wide web companion*, pages 1129–1134, 2014.

[2] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: Exploring the Power of Tables on the Web. *Proc. VLDB Endow.*, 1:538–549, 2008.

[3] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding Related Tables. In *Proc. of the Int. Conf. on Management of Data*, pages 817–828, 2012.

[4] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proc. of the 20th SIGKDD*, pages 601–610, 2014.

[5] S. Duan, A. Fokoue, O. Hassanzadeh, A. Kementsietsidis, K. Srinivas, and M. J. Ward. Instance-based matching of large ontologies using locality-sensitive hashing. In *The Semantic Web–ISWC 2012*, pages 49–64. Springer, 2012.

[6] J. Fan, L. Meiyu, O. Beng Chin, T. Wang-Chiew, and M. Zhang. A Hybrid Machine-Crowdsourcing System for Matching Web Tables. In *30th IEEE Int. Conf. on Data Engineering*, pages 976–987, 2014.

[7] R. Gupta, A. Halevy, X. Wang, S. Whang, and F. Wu. Biperpedia: An Ontology for Search Applications. In *Proc. 40th Int'l Conf. on Very Large Data Bases*, 2014.

[8] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.

[9] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. VLDB Endow.*, 3:1338–1347, 2010.

[10] V. Mulwad, T. Finin, Z. Syed, and A. Joshi. Using linked data to interpret tables. In *Proc. of the 1st Int. Workshop on Consuming Linked Data*, 2010.

[11] D. Rinser, D. Lange, and F. Naumann. Cross-Lingual Entity Matching and Infobox Alignment in Wikipedia. *Inf. Syst.*, 38:887–907, 2013.

[12] Y. A. Sekhavat, F. di Paolo, D. Barbosa, and P. Merialdo. Knowledge Base Augmentation using Tabular Data. In *Proc. of the 7th Workshop on Linked Data on the Web*, 2014.

[13] A. Singhal. Introducing the knowledge graph: Things, not string. Blog, 2012. Retrieved March 19, 2015.

[14] F. Suchanek, S. Abiteboul, and P. Senellart. Paris: Probabilistic alignment of Relations, Instances, and Schema. *Proc. VLDB Endowment*, 5:157–168, 2011.

[15] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *Proc. of the 6th Int. World Wide Web conference*, NY, 2007. ACM Press.

[16] 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.*, pages 528–538, 2011.

[17] J. Wang, H. Wang, Z. Wang, and K. Q. Zhu. Understanding Tables on the Web. In *Proc. of the 31st Int. Conf. on Conceptual Modeling*, pages 141–155. Springer-Verlag, 2012.

[18] Y. Wang and J. Hu. Detecting Tables in HTML Documents. In *Proc. of the 5th Int. Workshop on Document Analysis Systems V*, pages 249–260, 2002.

[19] G. Weikum and M. Theobald. From Information to Knowledge: Harvesting Entities and Relationships from Web Sources. In *Proc. 29th Symp. on Principles of Database Systems*, pages 65–76. ACM, 2010.

[20] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *Proc. of the 2012 SIGMOD*, pages 97–108, 2012.

[21] Z. Zhang. Start small, build complete: Effective and efficient semantic table interpretation using tableminer. *Under transparent review: The Semantic Web Journal*, 2014.

[22] Z. Zhang. Towards efficient and effective semantic table interpretation. In *The Semantic Web–ISWC 2014*, pages 487–502. Springer, 2014.