Matching Web Tables with Knowledge Base Entities: From Entity Lookups to Entity Embeddings

Vasilis Efthymiou*[1], Oktie Hassanzadeh[2], Mariano Rodriguez-Muro[2], and Vassilis Christophides[3]

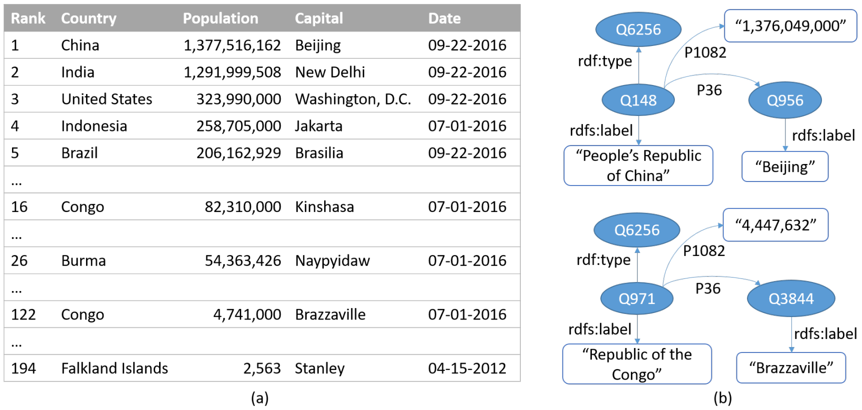
[1] ICS-FORTH & University of Crete, Greece
[2] IBM Research, USA
[3] INRIA Paris-Rocquencourt, France

Abstract. Web tables constitute valuable sources of information for various applications, ranging from Web search to Knowledge Base (KB) augmentation. An underlying common requirement is to annotate the rows of Web tables with semantically rich descriptions of entities published in Web KBs. In this paper, we evaluate three unsupervised annotation methods: (a) a *lookup-based* method which relies on the minimal entity context provided in Web tables to discover correspondences to the KB, (b) a *semantic embeddings* method that exploits a vectorial representation of the rich entity context in a KB to identify the most relevant subset of entities in the Web table, and (c) an *ontology matching* method, which exploits schematic and instance information of entities available both in a KB and a Web table. Our experimental evaluation is conducted using two existing benchmark data sets in addition to a new large-scale benchmark created using Wikipedia tables. Our results show that: 1) our novel lookup-based method outperforms state-of-the-art lookup-based methods, 2) the semantic embeddings method outperforms lookup-based methods in one benchmark data set, and 3) the lack of a rich schema in Web tables can limit the ability of ontology matching tools in performing high-quality table annotation. As a result, we propose a hybrid method that significantly outperforms individual methods on all the benchmarks.

1 Introduction

A large amount of data is published on the World Wide Web as structured data, embedded in HTML pages. A study by Cafarella et al. [9] estimated that Google's index of English documents contains 154 million high-quality relational tables, which constitute a valuable source of facts about real-world entities (e.g., persons, places, products). On the other hand, a great variety of real-world entities are described on the Web as Linked Data. Only the English version of DBpedia [8] describes 6.2M entities using 1.1B triples, including 1.6M persons, 800K places, 480K works (e.g., films, music albums), 267K organizations, 293K species, and 5K diseases.

* Work done while at IBM Research.



Rank	Country	Population	Capital	Date
1	China	1,377,516,162	Beijing	09-22-2016
2	India	1,291,999,508	New Delhi	09-22-2016
3	United States	323,990,000	Washington, D.C.	09-22-2016
4	Indonesia	258,705,000	Jakarta	07-01-2016
5	Brazil	206,162,929	Brasilia	09-22-2016
…				
16	Congo	82,310,000	Kinshasa	07-01-2016
…				
26	Burma	54,363,426	Naypyidaw	07-01-2016
…				
122	Congo	4,741,000	Brazzaville	07-01-2016
…				
194	Falkland Islands	2,563	Stanley	04-15-2012

Fig. 1. (a) An example of a Web table describing countries ranked by population (b) parts of two of those countries' descriptions from Wikidata.

In this paper, we study the problem of interpreting the rows of Web tables and matching them to semantically rich descriptions of entities published in Web KBs. *Web table annotation* [20, 23] (or interpretation [39, 25]) is a prerequisite for a number of applications, such as Web table search [6, 35] or KB augmentation [29, 13, 11, 37, 38, 30]. We focus only on the evaluation of instance-level matching (table rows to KB entities) and leave the evaluation of schema-level matching (table columns to KB properties) outside the scope of this work.

Example 1. Figures 1(a) and (b) contain the descriptions of countries, as they can be found in a Web table and in Wikidata [4]. The header row (in gray color), gives the property names of the described entities. Each row in the table describes a real-world entity (e.g., the second row describes China), and each column contains the value of the corresponding property, e.g., ("Population", "1,377,516,162"), ("Capital", Beijing)[4]. Graph-based descriptions of the same entities are available in the KB, e.g., China is described by node Q148, which is of type country (node Q6256) and has a label "People's Republic of China". Entity Q148 (China) is related with Q956 (Beijing) by the property P36 (capital).

There are several key challenges in Web table annotation:

1. The *types* of the entities described in a table are *not known in advance*, and the entities described may correspond to more than one type in the target KB. Most of the entities described in the table of Figure 1(a), can uniquely be matched to an entity of type country in Wikidata. However, there are some exceptions. For example, "China" is also the name of a city in Japan (Q932423) and a city in Texas (Q288864)[5]. Also, "Falkland Islands" is of type "British overseas territory" and not "country" according to Wikidata.

[4] This model is only applicable to horizontal relational tables, leaving out vertical tables such as Wikipedia infoboxes. Turning vertical tables to horizontal, identifying sub-tables, grouped columns, etc. are challenges beyond the scope of this work.

[5] Note that although the column header may indicate the right type for the column contents, the majority of tables on the Web have missing or obscure headers [6].

2. Which *columns* should be used to check for *correspondences may differ* from one table row to another. In our example, column "Country" can be used to uniquely identify the names of the entities described in the table. However, some of the values are not unique: e.g., "Congo" in rows 16 and 122, corresponds to two neighbor countries, namely the Democratic Republic of the Congo, and the Republic of the Congo. To successfully match entities, we need to compare descriptions using a *variable* subset of columns/properties per entity (e.g., "Country" only is enough for most rows, but for Congo, "Country" and "Capital" are required).
3. The names of an entity described both in a Web table and in a KB, may significantly differ. This implies that high string similarity of entity labels does not provide sufficient matching evidence and additional information, such as relations to other entities, might be needed as well. For example, the entity described in row 26 of Figure 1(a) has name "Burma" (the old name of the country), which is different from label "Myanmar" used in Wikidata. However, those descriptions can be matched, based on their capital Naypyidaw, which has the same name both in Wikidata and the table.

Clearly, the quality of the entity mapping process depends on the richness of the context (e.g., types, names, relationships) exploited to establish the mappings between Web tables and KBs. In this work, we benchmark three alternative unsupervised approaches for matching entities whose contextual information may vary from poor (in Web tables) to rich (in KBs).

First, we examine a *lookup-based* method, which exploits the columns of the Web tables recognized as entity names. It essentially detects correspondences using the *minimal contextual information* available in Web tables, which is then refined (based on frequently occurring terms in entity descriptions) or enriched (by exploiting relationships with other entities) with respect to the context of entities available in the KB. In the opposite direction, we can exploit a *semantic embeddings* method that exploits a vectorial representation of the rich entity context in a KB to identify the most relevant subset of entities in the Web table. In-between, we explore an *ontology matching* approach, which exploits schematic and instance information of entities available both in a KB and a Web table

In summary, the contributions of our work are as follows:

- We *experimentally evaluate* the effectiveness of different Web table annotation methods on gold standards exhibiting different data characteristics (varying number of rows and columns, the existence of related entities, etc.).
- We provide a *new Web table annotation gold standard*, which is the largest in the literature (by 3 orders of magnitude in the number of tables and 2 orders of magnitude in the number of rows and provided matches), while it contains the greatest diversity on Web table sizes in both rows and columns. We show that this gold standard is more challenging than other gold standards used in this field, due to its structure, diversity, and size.
- We introduce a *novel lookup-based method* that exploits entity relations and frequent words in the values of entity descriptions, outperforming the accuracy of state-of-the-art lookup method by up to 15% in F-score.

– We propose a *hybrid method* for Web table annotation, which outperforms existing methods on all benchmarks, by up to 16% in F-score, while it is able to discover up to 14% more annotations than the individual methods it is composed of.

Outline. In what follows, we first discuss the scope of our study and position our work in the literature (Section 2). In Section 3, we introduce the three classes of annotation methods. In Section 4, we present our experimental evaluation using existing gold standards, as well as our Wikipedia-based gold standard, and finally, we conclude our paper in Section 5.

2 Background & Related Work

In this section, we position our work in the literature, with respect to the different tasks on which the Web table annotation problem is decomposed. While there has been some remarkable work on supervised Web table annotation (e.g., [7, 23, 32]), here we focus on unsupervised and scalable methods, which do not require training sets of annotated tables, or any kind of human interaction (e.g., [18]). Our motivation for this focus is our use case in designing a fully unsupervised and generic cloud API, making no assumptions about the input corpus and availability of training data. Our aim is not an exhaustive evaluation of every possible method, such as supervised (e.g., [7, 23, 32]) or less scalable (e.g., [19]) methods.

Interpretation of Web tables. Our goal is to map each Web table row to an entity described in a KB, unlike related works [23, 39, 7] treating individual cells as entities. The attribute values for an entity described in a row, are given by the contents of the cells for each column of the row, following the definition of entity descriptions in the Web of data as sets of attribute-value pairs [10].

Label column detection. The vast majority of Web tables contain a column, whose values serve as the names of the described entities [6]. Rather than supervised learning [6], we rely on a heuristic method: the *label column* is defined as the leftmost column with the maximum number of distinct (non-numeric) values [28]. In other words, the label of an entity (given by the label column) is the most important attribute of an entity (described as a table row).

Lookup. Recent works follow an iterative approach between instance- and schema-level refinements, until convergence. The first step for such refinements is to look up the contents of the label column in a KB index and get a list of first, unrefined candidate matches. For instance, Ritze et al. [28] use the DBpedia lookup service [1], while Zhang [39] uses the Freebase lookup service as baselines. In our experiments, we also use the unrefined results of *DBpedia lookup* as a baseline. In our lookup-based approach, we use our own generic search index over Wikidata entities that we refer to as *FactBase*. Another interesting approach is to use a trained text classifier to extract the entity types from the snippets of Google search results, given the content of the cell which has been inferred to contain the entity name [27].

Relations extraction. Relationships between entities described in the same row of a Web table can be induced by a probabilistic model built from a Web document corpus and natural language processing [30, 6]. Our relationship extraction method is inspired by Venetis et al. [35], which consults an isA database and a relations database to identify binary relations in Web tables. Instead, we exploit the information contained in the target KB, and the unambiguous entity mappings that have been already identified.

Ontology matching and link discovery tools. There is a large body of work on Ontology Matching [33]. LogMap [22] is a logic-based tool for matching semantically rich ontologies. It iteratively explores new correspondences, based on a first list of lexicographically and structurally similar entities and the ontologies' class hierarchies, which are then searched for logical inconsistencies. It has been evaluated as one of the best and most efficient publicly available ontology matching tools [12]. PARIS [34] is an iterative probabilistic tool, that defines a normalized similarity score between the entities described in two ontologies, representing how likely they are to match. This similarity depends on the similarity of their neighbors, and is obtained by first initializing the scores on literal values, and then propagating the updates through a relationship graph, using a fixed point iteration. We have chosen these tools based on their popularity and availability, while we are planning to extend our experiments with other ontology matching tools such as RiMOM-IM [31] and SERIMI [5], which have also shown good results in the recent OAEI [2] benchmarks.

Link discovery tools (e.g., [36, 26]) have a similar goal, but their applicability to our problem is limited as they require linkage rules that are manually-specified, or learned from training data [21].

Entity matching context. T2K [28] annotates Web tables by mapping their columns to DBpedia properties, and their rows to DBpedia entities, associating the whole table with a DBpedia class. The initial candidate instance mappings stem from a lexicographical comparison between the labels used in the table and those of the entities described in DBpedia, which allows a first round of property mapping. The results of property mapping can then be used to refine the instance mappings, and this process continues until convergence. Our lookup-based method uses a similar candidate generation phase, and then exploits entity types, relations and frequent terms in the descriptions of candidate matches to refine or even expand the candidate matches.

TableMiner [39] maps columns to ontology classes and single cells to entities, following a two-phase process. In the first sampling phase, it searches for candidate matches, which are ranked based on similarity computations using the contents of the table, as well as the page title, surrounding paragraphs and table caption. Then, it scans the table row-by-row, until a dynamic confidence value for the type of each column has been reached. In the second phase, it uses the class mappings of the first phase to refine the candidate instance mappings. Although new candidate matches can be provided in the second phase, convergence is usually reached from the first iteration. We use a similar sampling phase to detect the entity types in a table (using the label column).

Gold standards. T2D [28] consists of a schema-level gold standard of 1,748 Web tables, manually annotated with class- and property-mappings, as well as an entity-level gold standard of 233 Web tables. Limaye [23] consists of 400 manually annotated Web tables with entity-, class-, and property-level correspondences, where single cells (not rows) are mapped to entities. We have adapted the corrected version of this gold standard [7] to annotate rows with entities, from the annotations of the label column cells. Finally, Bhagavatula et al. [7] use a gold standard extracted from 3,000 Wikipedia tables, using the hyperlinks of cells to Wikipedia pages. In this paper, we introduce a new, instance-level gold standard from 485K Wikipedia tables, the biggest that exists in the literature, in which we use the links in the label column to infer the annotation of a row to a DBpedia entity. Overall, those gold standards exhibit a variety in their sizes, existence of relations, and sparseness, helping us show how these characteristics affect the quality of different annotation methods.

3 Matching Algorithms

In this section, we describe three different individual methods for the problem of Web table annotation, as well as a hybrid solution, built on them.

3.1 Lookup Method

The lookup-based method tries to match the poor information for entities offered by Web tables to the rich information offered for those entities in a KB. In order to search for the closest possible result in the KB to the contents of a Web table, it uses a lookup service on the target KB.

Refined lookup. In this baseline, we keep the type of the top lookup result for each label column cell in a first scan of the table and then store the top-5 most frequent types for each column as acceptable types[6]. Then, we perform a second lookup, but this time, we restrict the results to those of an acceptable type. We select the top result from the refined lookup as the annotation of each row. This method tries to increase the cohesiveness of the results, by filtering lookup results which do not fit well with the rest. As an example, consider that many lookup results are returned for the query “China”, but we only want to restrict our results to those of an acceptable type (e.g., country, populated place).

FactBase lookup. The lookup method that we introduce, identifies and exploits frequent terms in the description of an entity, as well as entity relations. We build on a generic indexing mechanism over a KB with IDs and textual descriptions, and call the generated index *FactBase*. FactBase offers a lookup service, allowing the retrieval of entities with a specific label, or any given attribute-value pair. The pseudocode of FactBase lookup can be found in Algorithm 1.

We perform a first scan of the Web table similar to the refined lookup method (Lines $6-19$). In addition to frequent types, in this method, we also extract the most frequent words, excluding stopwords, used in the values of rdfs:description. Another feature that we extract in the first scan, is the set of binary relations between the entity described in a table row and entities mentioned in the same row, as part of its description (Lines $16-19$). To identify binary relations, we

[6] We assume entities described in the same column to be of the same conceptual type, which can be expressed by different OWL classes, not considering class hierarchies.

Algorithm 1: FactBase lookup.

```
   Data: Table $T$
   Result: Annotated table $T'$
1  $T' \leftarrow T$;
2  allTypes $\leftarrow \emptyset$ ;                                  /* a multiset of types */
3  descriptionTokens $\leftarrow \emptyset$ ;                         /* a multiset of tokens */
   /* samplePhase                                                                        */
4  labelColumn $\leftarrow$ getLabelColumn(T);
5  referenceColumns $\leftarrow$ getReferenceColumns(T);
6  for each row $i$ of $T$ do
7      label $\leftarrow$ $T.i.labelColumn$;
8      results $\leftarrow$ search(label);
9      if results.size $> 0$ then
10         topResult $\leftarrow$ results.get(0);
11         allTypes.addAll(topResult.getTypes());
12         tokens $\leftarrow$ topResult.getDescriptionTokens();
13         descriptionTokens.addAll(tokens);
14         if results.size $= 1$ then
15             annotate($T'.i$, topResult);
16             for each column $j$ of referenceColumns do
17                 $v \leftarrow T.i.j$;
18                 if topResult.containsFact($a$,$v$) then  /* $v$ is the value of a relation $a$ */
19                     candidateRelations.add($j$,$a$);
20 acceptableTypes $\leftarrow$ allTypes.get5MostFrequent();
21 descriptionTokens $\leftarrow$ descriptionTokens.getMostFrequent();
22 for each column $j$ of referenceColumns do
23     relations[$j$] $\leftarrow$ candidateRelations.get($j$).getFirst();
   /* annotation phase                                                                   */
24 for each row $i$ of $T$ do
25     if isAnnotated($T'.i$) then continue;
26     label $\leftarrow$ $T.i.labelColumn$;
27     results $\leftarrow$ search_strict(label, acceptableTypes, descriptionTokens);
28     if results.size $> 0$ then
29         topResult $\leftarrow$ results.get(0);
30         annotate($T'.i$, topResult);
31         continue ;                                                /* go to the next row */
32     for each column $j$ in relations do
33         $r \leftarrow$ relations[$j$];
34         results $\leftarrow$ search_loose(label,$r$,$T.i.j$);
35         if results.size $> 0$ then
36             topResult $\leftarrow$ results.get(0);
37             annotate($T'.i$, topResult);
38             break ;                                               /* go to the next row */
```

build on the observation that when the lookup result is unique, it is in most cases a correct annotation. For the unique results, we further examine if any of their attribute-value pairs have the same value with any of the cells of the current row marked as entity references. If that is the case, then we add the attribute of the attribute-value pair as a candidate binary relation expressed by the column of this cell. After a small number of agreements on the same attribute for a column (5 agreements in our experiments), we use this attribute as the final extracted relation expressed in this column. Finally, we use the unique lookup result for a row as the annotation of this row, skipping the next phase (Lines 15, 25).

After the first scan, many rows are now annotated with a unique lookup result[7]. For the rest of the rows, either many results were returned, i.e., a more fine-grained lookup is needed (disambiguation), or no results were returned, so

[7] We assume that some of the results will be unique, but this is not a requirement. If it holds, it only speeds up the process and helps in identifying binary relations.

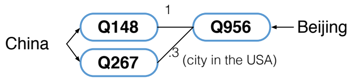


Fig. 2. Disambiguation graph for two terms. China candidates are the country or the city. Edges stand for the topic similarity score between the candidate objects.

a more coarse-grained lookup is needed. For the first case, we perform a new, refined lookup (*search_strict*), restricting the results to those of an acceptable type, having one of the most frequent tokens in their rdfs:description values, if applicable (Lines $28 - 31$). For the second case, we perform a new, looser lookup (*search_loose*) in the labels, allowing a big margin of edit distance (Levenshtein), restricting the results to have in their facts one of the binary relations that we have extracted (Lines $32 - 38$). Allowing a big margin of edit distance offers a tolerance to typos, nicknames, abbreviations, etc, while relating to the same, third entity, in the same way (i.e., using the same relationship) is a positive evidence for two entities to match [15]. The final annotation is the lookup result with the most similar label.

For example, in Figure 1, if many results for the query term "China" are returned, we keep those with an acceptable type (e.g., country, populated place) and having the most frequent words (e.g., "country", "state") in their description. If no results are returned for the query "China", then we perform a new query, restricting the results to only those whose capital is called "Beijing", even if their label is not exactly "China", but something as close to that as possible.

3.2 Entity Embeddings Method

The approach we now describe is a variation of a linking approach for text disambiguation. We considered this approach promising for table annotation because its core hypothesis is compatible with the task. The technique is an instance of a family of techniques called *global disambiguation* techniques, which assume that the entities that appear in sentences or paragraph tend to form *coherent sets* with respect to the topic being discussed in the text. For example, consider two terms, *China* and *Beijing*, as shown in Figure 2. There are two candidates for China, either the city or the country. However, a global approach would annotate "China" with the country (Q148) because it has a stronger connection to the city of Beijing (Q956). This assumption also applies to entities described in tables, where columns are usually strongly typed, and hence coherent at least with respect to types and topics.

We base our work on the global disambiguation technique used in the DoSeR framework [40], where similarity between entities is computed as the *cosine distance* between their vector representations. These vectors, called *embeddings*, are a continuous-space representation of the entities in the target KB (e.g., DBpedia), that capture the structure of the neighborhood of each node. In DoSeR, embeddings are computed using *word2vec* [24], an embedding algorithm for text that is known for its performance and scalability in computing embeddings for words. While graphs are clearly different than text, e.g., they have no clear start or end, DoSeR uses a novel approach to apply word2vec that has shown good results in terms of scalability and performance of the resulting embeddings. We will now describe this approach and the way we apply it for table disambiguation, which we divide in two stages, *off-line* and a *on-line*.

During the *off-line* stage, we first create a *surface form index* that maps each entity e of the KB to a set of known names for the entity $m(e)$. It is collected from properties in the KB that are known to contain *common names* for entities, e.g., rdfs:label, skos:altLabel, etc. In the case of DBpedia, we also use the property dbo:wikiPageWikiLinkText, which contains the anchor text used in Wikipedia to refer to the wiki page of e. Second, we compute *entity embeddings* using word2vec as follows: given the target KB, we generate a text document d by performing a random walk over the neighborhood of each entity in the KB and at each step of the walk, we append the visited node URIs to d. The resulting text document d is now used as input to word2vec which produces the embeddings for all words (node URIs) in the corpus in the standard way.

During the *on-line* stage, we use the embeddings and the surface form index to annotate tables. We consider only columns with text values, and regard each string value as an entity mention e. Then, given a set of entity mentions E, we annotate each entity $e \in E$ as follows. First, we create a *disambiguation graph* where the set of vertices V is the union of all candidates entities $m(e)$ (obtained from the surface form index) for all mentions $e \in E$. For each pair for vertices $v_1, v_2 \in V$ such that the vertices are not candidates for the same mention, i.e., there is no $e \in E$ such that $v_1, v_2 \in m(e)$, we add a weighted directed edge $(v_1, v_2, etp(v_1, v_2))$, where $etp(v_1, v_2)$ is the normalized cosine similarity between the embeddings $emb(v_1)$, $emb(v_2)$ of v_1, v_2, respectively, computed as follows:

$$etp(v_1, v_2) = \frac{cos(emb(v_1), emb(v_2))}{\sum_{k \in V} cos(emb(v_1), emb(k))}.$$

Finally, we create an assignment for each node by applying a weighted PageRank algorithm [40] that allows us to compute the relevance of each node. We use 50 iterations for PageRank and select the nodes with the highest score from the set of candidates for each mention.

3.3 Ontology Matching Method

In this section, we briefly describe an ontology matching framework for annotating Web tables. Our framework provides the required input to any ontology matching tool, resulting in Web table annotations. Our candidate mapping selection enables even the less scalable ontology matching tools to provide annotations to large-scale KBs. For this approach, we require the existence of a header row, since each cell of this row defines the name of a property, and the cell of the label column in the header row defines the name of the table's class. For more details and preliminary results of this method, please refer to our previous work [17].

TBox. The values of an entity description can be literals, i.e., the column property is a datatype property, or references to other entities, i.e., the column property is an object property. We distinguish them by a pre-processing scan of the table and by sampling the data types of each column. To identify columns with entity references, we perform a small number of lookups in FactBase using the first few values from this column, if we have not already assigned it a numeric or date type. If most of those lookups return any result, we mark this column as an object property. The same scan also identifies the label column.

ABox. We perform a second scan, in which we create a new instance of the table class for each row. The label and URI suffix assigned to each instance are determined by the label column cell. The values of this instance for the property of each column, are the cell contents of this row for the respective column.

In Figure 1, the header row defines the class of the table, *Country*, and the properties *Population*, *Capital*, and *Date*, all having the domain *Country*. *Capital* is an object property with range *Capital*. For the first row of the table in Figure 1, we create an instance of the class *Country*, having the label "China", the value 1,377,516,162 for the property *Population*, the value *Beijing* for the property *Capital*, and the value 09-22-2016 for the property *Date*.

Blocking. To enable ontology matching tools that do not scale well be applicable in this framework, and to improve the efficiency of matching tools that do scale, we have applied a pre-processing step of candidate mappings selection, known as *blocking* [10]. Specifically, we retain only the DBpedia instances whose labels have at least one common token with the labels of our ontology's instances.

Finally, we call an ontology matching tool with the table ontology and the DBpedia ontology after blocking, as input, and return the mapping results.

3.4 Hybrid

We introduce two simple hybrid methods, to explore the benefits of combining FactBase lookup and embeddings, in the following way:

Hybrid I. If FactBase lookup provides a mapping for the entity of a row, then this hybrid method keeps this mapping. Otherwise, it uses the annotation provided by the embeddings for this row, if one exists.

Hybrid II. Same as Hybrid I in inverse order, i.e., using the embeddings first, before FactBase lookup.

The motivation is that individual methods handle different aspects of the contextual information that is offered in Web tables. As our experiments show, where one approach fails to perform correct annotations, the other approach often succeeds. This approach can only improve the recall of its first component (i.e., FactBase lookup for Hybrid I and embeddings for Hybrid II), since it returns all the annotations of the first component, plus additional annotations from the second component, if the first fails.

4 Experiments

Settings For our experiments, we use MapReduce for annotating and evaluating multiple tables in parallel, and a key-value store as our index. We do not report run times for each experiment as they depend on the cluster configuration and other settings. Our experiments on smaller datasets take only a few minutes on our cluster of 16 medium-sized nodes, while experiments on larger datasets take several hours to finish. Our *FactBase* index implementation uses a 2016 dump of Wikidata, with entities linked to corresponding DBpedia entities. Hence, FactBase lookup results using gold standards annotated with older versions of DBpedia may slightly underestimate its accuracy. The datasets generated or used are made publicly available [16] along with implementation details: `http://ibm.biz/webtables`.

4.1 Datasets

In our experiments, we use three gold standards, whose characteristics are summarized in Table 1. Rows per table show the min, max and average number of rows per table in each gold standard. The same holds for columns per table. For the number of tables featuring entity relations, we applied the relation detection method from *FactBase lookup*. For a measure of cell completeness, we use *structuredness* as defined in [14]. In this context, we compute the percentage of cells in a table that are not empty (or "NULL" or "-"), as the structuredness of a table. Then, the overall structuredness of a gold standard is a weighted sum of each table's structuredness, where the weight of each table is based on its sum of columns and rows, normalized by the total sum of columns and rows in this gold standard. Intuitively, a structuredness value of 1 indicates that no cells are empty and 0 structuredness represents that all cells are empty.

Table 1. Characteristics of the gold standards. All are made publicly available [16].

Name	Tables	Rows	Matches	Rows per table (av.)	Columns per table (av.)	Tables with relations	Structuredness
T2D	233	28,647	26,124	6 - 586 (123)	3 - 14 (4.95)	108 (46%)	0.97
Limaye	296	8,670	5,278	6 - 465 (29)	2 - 6 (3.79)	78 (26%)	0.59
Wikipedia	485,096	7,437,606	4,453,329	2 - 3,505 (15)	1 - 76 (5.58)	24,628 (5%)	0.85

T2D [3] consists of 233 Web tables, manually annotated with instances from the 2014 version of DBpedia. It has the highest average number of rows per table (123), and the highest ratio of tables with relations (46%). It is also the gold standard with the highest structuredness (0.97), meaning that very few cells are empty in this corpus.

The updated version of the *Limaye* gold standard [23] published by Bhagavatula et al. [7], annotates cells with Wikipedia pages. We have replaced Wikipedia annotations with the corresponding entities from the October 2015 version of DBpedia. To make this gold standard applicable to our model, we have kept only one annotation per row, the one assigned to the cell of the label column, in 296 Web tables for which a label column could be detected. This is the gold standard with the lowest number of columns per table on average (3.79), still, one out of four tables (26%) contains entity relations. Due to a big number of empty cells, it presents the lowest structuredness (0.59), while it also contains a big number of empty rows. Missing data have a negative impact in the quality of the annotations for some systems, such as *T2K*. This dataset could not be used in the evaluation of ontology matching methods, as it misses header rows, thus no meaningful property and class names could be created for a table ontology.

Finally, we have created our own *Wikipedia* gold standard, by extracting the hyperlinks of existing Wikipedia tables to Wikipedia pages, which we have replaced with annotations to the corresponding entities from the October 2015 version of DBpedia. Since the header rows in Wikipedia tables are not linked to properties, our gold standard does not contain schema-level mappings. For the needs of our experiments, we only consider one mapping per row to evaluate the different methods, using the annotations for the label column. This gold standard is much more noisy than the other two, as it contains unannotated rows, multi-

Table 2. Results over T2D, Limaye, and Wikipedia Gold Standards.

Method	T2D Gold Standard			Limaye Gold Standard			Wikipedia Gold Standard		
	Re	**Pr**	**F1**	**Re**	**Pr**	**F1**	**Re**	**Pr**	**F1**
DBpedia lookup	0.73	0.79	0.76	0.73	0.79	0.76	-	-	-
DBpedia refined	0.76	0.86	0.81	0.68	0.73	0.71	-	-	-
T2K	0.76	**0.90**	0.82	0.63	0.70	0.66	0.22	**0.70**	0.34
FactBase lookup	**0.78**	0.88	**0.83**	**0.78**	**0.84**	**0.81**	0.50	**0.70**	0.58
Embeddings	0.77	0.86	0.81	0.65	**0.84**	0.73	**0.53**	**0.70**	**0.60**
Blocking	0.71	0.32	0.44	-	-	-	0.39	0.16	0.23
LogMap	0.57	0.89	0.70	-	-	-	0.29	0.34	0.32
PARIS	0.04	0.42	0.07	-	-	-	-	-	-
Hybrid I	**0.83**	0.87	**0.85**	**0.79**	**0.84**	0.81	0.57	0.66	0.61
Hybrid II	0.81	0.85	0.83	**0.79**	**0.84**	**0.82**	**0.60**	0.69	**0.64**

column and multi-row cells, which we split and replicate to avoid empty cells, and cells whose contents are entity labels with additional information for an entity (e.g., the cell "George Washington February 22, 1732 - December 14, 1799 (aged 67)", refers to George Washington), which makes the annotation task more difficult, as such labels are very dissimilar to the corresponding entity labels in a KB. Finally, even if the average number of rows (15) is much smaller than in the other two gold standards, we note that it contains almost 800 tables with more than 1,000 rows and the largest table consists of 3.5K rows. This gold standard contains the lowest ratio of tables with detected relations (only 5%), while it exhibits the highest average number of columns per table (5.58). Its structuredness is high (0.85), thus, only a few cells are empty.

4.2 Evaluation

In Table 2, we present the experimental results over the three gold standards, with respect to micro-averaged recall, precision, and F-score. The micro-averaged values over a set of tables are acquired by using the sums of true positives, false positives, true negatives and false negatives from each table, as if they were a single test. Different methods are separated by double horizontal lines.

Results over T2D gold standard As the results in Table 2 show, a simple *DBpedia lookup* without any schema-level refinements has very good results, verifying the numbers reported in [28]. Moreover, the *DBpedia lookup refined* is almost as good as state-of-the-art methods. *FactBase lookup* is the overall winner in this gold standard, having a slightly better recall than *T2K* (+2%) and a slightly worse precision (-2%). The *embeddings* are also better than *T2K* with respect to recall (+1%), but worse overall (-1% in F-score), showing almost the same results as the *DBpedia lookup refined* baseline. *The almost perfect structuredness value of T2D provides the ideal conditions for methods that exploit all the columns of a table for their annotations*, such as *T2K*.

The *ontology matching* tools exhibit much worse results, mainly in recall. For a fair comparison between the methods, we have included the results of *blocking*, which the ontology matching tools use as input. It is important to note that the recall of *blocking* is the upper recall threshold that an ontology matching tool can achieve and in this case, it is already lower than the recall of the other methods. Still, the difference between the recall of *blocking* and the recall of *LogMap* [22]

(-14%) and *PARIS* [34] (-73%), is substantial. Most ontology matching tools are not designed to provide mappings between such heterogeneous ontologies with respect to the information richness and diversity they contain, like the ones we produce. The number of attributes (i.e., columns) used to describe entities in a Web table are quite different than the respective information in the ontology of a KB. Efthymiou et al. [15] show that an average entity description in DBpedia uses 11.44 attributes, whereas the number of attributes used in a Web table corpus is typically between 4 and 6 (Table 1). Furthermore, Duan et al. [14] show that unlike Web tables, KBs such as DBpedia are of very low structuredness.

The effectiveness of lookup-based methods heavily relies on the lookup service that is employed. Thus, lookup-based annotations could be used as an evaluation of such services. For example, when substituting *DBpedia lookup* with an unrefined version of FactBase lookup, the results were 0.57 recall, 0.62 precision, and 0.59 F-measure, as opposed to the T2D results for the DBpedia lookup (0.73 recall, 0.79 precision, and 0.76 F-measure). This shows that the DBpedia lookup service is much better than the FactBase lookup service, as FactBase lookup is still in the development phase. This difference is observed, mainly due to the different ranking of the results in those two services. DBpedia lookup service exploits the in-degree of entities in its returned rankings, whereas the lookup service of FactBase only considers the label similarity to the query.

Results over Limaye gold standard As shown in Table 2, *FactBase lookup* outperforms other approaches with a difference of +8% in F-score from the second best technique, the *embeddings*, even if they are both tied at the highest precision. The difference in recall from the second best method *DBpedia lookup* is +5%. As we can see in Table 1, even if this gold standard contains more tables than *T2D*, the number of rows in those tables is significantly lower. Thus, *methods that rely on a sampling phase (e.g., FactBase lookup), or on a set of coherent results (e.g., embeddings), perform worse than in datasets with bigger tables*, which is also the reason why DBpedia lookup performs better than DBpedia refined in this gold standard. Nonetheless, even if this gold standard contains small and sparse tables (0.59 structuredness), there is a decent percentage (26%) of tables with entity relations, which *FactBase lookup* can exploit to achieve a much better performance than *embeddings.* The recall values of *DBpedia lookup* and *DBpedia lookup refined* are close to those of *embeddings*, while the latter show a much better precision. Due to the missing rows in this gold standard, *T2K* may disregard some tables as of low quality. It may also detect a different label column than the one *FactBase lookup* detects. This also explains the worse performance of *T2K* compared to *DBpedia lookup. FactBase lookup* yields a 15% higher F-score than *T2K*, showing that it can better handle tables of low structuredness, i.e., with many missing values.

Results over Wikipedia gold standard As shown in Table 2, the *embeddings* show the best results for the *Wikipedia* gold standard, with *FactBase lookup* following (-4% in F-score) with worse recall (-6%) and equally good precision. Ontology matching is again worse than the other methods, even from the step of blocking (we excluded PARIS from this experiment given its poor performance on T2D). As expected, the results presented in this table are much worse than the other two gold standards, which can be justified by the noisiness of this dataset, as explained in Section 4.1. *Noisiness seems to favor the embeddings over the other methods, since it can better handle ambiguous mentions to entities in a textual context.* Another challenge in this gold standard is the small number of rows, which makes it more difficult for *FactBase lookup* and *embeddings* to have a decent sample for type refinements and relations extraction, and provide a set of coherent results, respectively. For the same reason, due its strict matching policy favoring precision over recall - at least 50% of the rows must be mapped to KB entities of the same type - *T2K* managed to annotate only 119K out of the 485K Web tables, resulting in a very low recall, presenting an overall performance close to that of *LogMap*. *DBpedia lookup* could not be applied to this gold standard, as the public server hosting it could not handle such a large amount of queries.

Hybrid methods As shown in Table 2, the hybrid methods seem to improve the results of their constituent methods, by enhancing recall, with a minor impact on precision. In *T2D*, *Hybrid I* exhibits an important improvement of the quality results over either of its constituent methods. Its recall is 5% better than that of *FactBase lookup*, while its precision is only 1% lower, resulting in a 2% higher F-score. *Hybrid II* is also better than its constituent methods, but slightly worse than *Hybrid I*. In *T2D*, the individual methods that constitute the hybrid have a 76% Jaccard similarity in the table rows that they annotate correctly, while their recall values are very close. An ideal solution that always chooses the correct annotation among the annotations provided by those methods would yield a recall of 0.88 for *T2D*.

In the *Limaye* gold standard, the benefit of using a hybrid method is not as significant as in *T2D*, but it is still the best-performing method. This is due to the fact that *FactBase lookup* performs much better than the *embeddings*, so the latter has little to offer in their combination. Still, the recall of their combination is better than that of *FactBase lookup* by 1% and the precision is the same as that of both methods (0.84). The two hybrid methods are almost identical, with *Hybrid II* showing a slightly better F-score (+1%). Again, the Jaccard similarity of correctly annotated rows in the constituent methods is 75%, while an ideal combination of those methods would yield a recall of 0.81.

In the *Wikipedia* gold standard, the hybrid methods significantly outperform the individual methods, as both of the constituent methods have a good precision and modest recall, which is the ideal case for such hybrids. Intuitively, in such cases the first constituent method has given only few annotations, still they are mostly correct. Thus, it has skipped to annotate many rows, which can be annotated by the second constituent method, mostly with a correct KB entity (good precision). Specifically, the recall of *Hybrid II* is better than that of the *embeddings* (+7%), while its precision is worse by only 1%, yielding an F-score that is 4% better than the *embeddings*. The difference to the *FactBase lookup* results is even bigger (+10% recall and +6% F-score). *Hybrid II* is much better than *Hybrid I* (+3% F-score) in this gold standard, since it exploits the better performance of *embeddings*.

Lessons Learned The following are the key lessons learned from our results: 1) Both *FactBase lookup* and *embeddings* work better with tables of many rows. In tables with few rows (less than 5), *embeddings* provide better annotations than *FactBase lookup*. 2) When the Web tables contain entity relations, *FactBase lookup* provides the best annotation results. 3) *Embeddings* can cope with noise in the string similarity of labels better than the other methods. 4) Most ontology matching tools are not suited to match a flat ontology to another which has a rich structure. 5) Hybrid methods work better when the constituent methods have modest recall and good precision.

5 Conclusion & Future Work

In this paper, we performed a thorough evaluation of three different families of methods to annotate Web tables, and discussed key lessons learned from our experiments. We introduced a new benchmark and a hybrid approach that outperforms individual methods by up to 16% in F-score. In the future, we plan to expand our evaluation of ontology matching tools and propose a new track for the upcoming OAEI campaign to encourage the community to use and extend ontology matching tools as knowledge base population systems.

Acknowledgments We would like to thank the authors of T2K [28], LogMap [22], and TabEL [7][8] for sharing their code and their assistance.

References

1. DBpedia Lookup. `http://wiki.dbpedia.org/projects/dbpedia-lookup`, [Online; accessed 27-07-2017]
2. Ontology Alignment Evaluation Initiative. `http://oaei.ontologymatching.org/`, [Online; accessed 27-07-2017]
3. T2D Gold Standard for Matching Web Tables to DBpedia. `http://webdatacommons.org/webtables/goldstandard.html`, [Online; accessed 27-07-2017]
4. Wikidata. `http://www.wikidata.org`, [Online; accessed 27-07-2017]
5. Araújo, S., Tran, D.T., de Vries, A.P., Schwabe, D.: SERIMI: class-based matching for instance matching across heterogeneous datasets. IEEE TKDE 27(5) (2015)
6. Balakrishnan, S., Halevy, A.Y., Harb, B., Lee, H., Madhavan, J., Rostamizadeh, A., Shen, W., Wilder, K., Wu, F., Yu, C.: Applying WebTables in Practice. In: CIDR (2015)
7. Bhagavatula, C.S., Noraset, T., Downey, D.: TabEL: Entity linking in web tables. In: ISWC (2015), Project homepage: `http://websail-fe.cs.northwestern.edu/TabEL/`
8. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - A Crystallization Point for the Web of Data. JWS 7(3) (2009)
9. Cafarella, M.J., Halevy, A.Y., Wang, D.Z., Wu, E., Zhang, Y.: WebTables: Exploring the Power of Tables on the Web. PVLDB 1(1) (2008)
10. Christophides, V., Efthymiou, V., Stefanidis, K.: Entity Resolution in the Web of Data. Morgan & Claypool Publishers (2015)
11. Dalvi, B.B., Cohen, W.W., Callan, J.: WebSets: extracting sets of entities from the web using unsupervised information extraction. In: WSDM (2012)

[8] TabEL results not included in our experiments as it was not yet functional at the time of writing this manuscript, missing some custom resources.

12. Daskalaki, E., Flouris, G., Fundulaki, I., Saveta, T.: Instance matching benchmarks in the era of linked data. Web Semantics: Science, Services and Agents on the World Wide Web 39 (2016)
13. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: KDD (2014)
14. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In: SIGMOD (2011)
15. Efthymiou, V., Stefanidis, K., Christophides, V.: Big data entity resolution: From highly to somehow similar entity descriptions in the web. In: IEEE Big Data (2015)
16. Efthymiou, V., Hassanzadeh, O., Rodrguez-Muro, M., Christophides, V.: Evaluating Web Table Annotation Methods: From Entity Lookups to Entity Embeddings. figshare. `https://doi.org/10.6084/m9.figshare.5229847` (2017)
17. Efthymiou, V., Hassanzadeh, O., Sadoghi, M., Rodriguez-Muro, M.: Annotating web tables through ontology matching. In: OM (2016)
18. Fan, J., Lu, M., Ooi, B.C., Tan, W., Zhang, M.: A hybrid machine-crowdsourcing system for matching web tables. In: ICDE (2014)
19. Guo, X., Chen, Y., Chen, J., Du, X.: ITEM: extract and integrate entities from tabular data to RDF knowledge base. In: APWeb (2011)
20. Hassanzadeh, O., Ward, M.J., Rodriguez-Muro, M., Srinivas, K.: Understanding a large corpus of web tables through matching with knowledge bases: an empirical study. In: OM (2015)
21. Isele, R., Bizer, C.: Learning expressive linkage rules using genetic programming. PVLDB 5(11) (2012)
22. Jiménez-Ruiz, E., Grau, B.C.: Logmap: Logic-based and scalable ontology matching. In: ISWC (2011), Code: `https://github.com/ernestojimenezruiz/logmap-matcher`
23. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and Searching Web Tables Using Entities, Types and Relationships. PVLDB 3(1) (2010)
24. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR abs/1301.3781 (2013)
25. Mulwad, V., Finin, T., Syed, Z., Joshi, A.: Using linked data to interpret tables. In: COLD (2010)
26. Ngomo, A.C.N., Auer, S.: LIMES A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In: IJCAI (2011)
27. Quercini, G., Reynaud, C.: Entity discovery and annotation in tables. In: EDBT (2013)
28. Ritze, D., Lehmberg, O., Bizer, C.: Matching HTML tables to DBpedia. In: WIMS (2015)
29. Ritze, D., Lehmberg, O., Oulabi, Y., Bizer, C.: Profiling the potential of web tables for augmenting cross-domain knowledge bases. In: WWW (2016)
30. Sekhavat, Y.A., Paolo, F.D., Barbosa, D., Merialdo, P.: Knowledge base augmentation using tabular data. In: LDOW (2014)
31. Shao, C., Hu, L., Li, J., Wang, Z., Chung, T.L., Xia, J.: Rimom-im: A novel iterative framework for instance matching. J. Comput. Sci. Technol. 31(1) (2016)
32. Shen, W., Wang, J., Luo, P., Wang, M.: LIEGE: link entities in web lists with knowledge base. In: KDD (2012)
33. Shvaiko, P., Euzenat, J.: Ontology matching: State of the art and future challenges. IEEE TKDE 25(1) (2013)
34. Suchanek, F.M., Abiteboul, S., Senellart, P.: PARIS: probabilistic alignment of relations, instances, and schema. PVLDB 5(3) (2011), Code: `http://webdam.inria.fr/paris/`
35. Venetis, P., Halevy, A.Y., Madhavan, J., Pasca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering Semantics of Tables on the Web. PVLDB 4(9) (2011)
36. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk - A Link Discovery Framework for the Web of Data. In: LDOW (April 2009)
37. Wang, J., Wang, H., Wang, Z., Zhu, K.Q.: Understanding tables on the web. In: ER (2012)
38. Yakout, M., Ganjam, K., Chakrabarti, K., Chaudhuri, S.: InfoGather: entity augmentation and attribute discovery by holistic matching with web tables. In: SIGMOD (2012)
39. Zhang, Z.: Towards efficient and effective semantic table interpretation. In: ISWC (2014)
40. Zwicklbauer, S., Seifert, C., Granitzer, M.: DoSeR - A Knowledge-Base-Agnostic Framework for Entity Disambiguation Using Semantic Embeddings. In: ESWC (2016)