Collecting University Rankings for Comparison Using Web Extraction and Entity Linking Techniques

Nick Bassiliades

Department of Informatics, Aristotle University of Thessaloniki,
Thessaloniki, Greece
nbasssili@csd.auth.gr

Abstract. University rankings are rankings of institutions in higher education, ordered by combinations of factors. Rankings are conducted by various organizations, such as news media, websites, governments, academics and private corporations. Due to huge financial and other interests, the rankings of universities worldwide recently received increasing attention. The rankings are based on different criteria and collect data in various ways. As a result, there is a large divergence in the specific rankings of different institutions. In order to compare rankings so that safe conclusions about their reliability are drawn, data from the sites of different such ranking lists must be collected. In this paper we present this first step for university ranking comparison, namely we discuss in detail how we have developed a Prolog application, called URank, that collects the data, by a) extracting them from the various ranking list web sites using web data extraction techniques, b) uniquely identifying the University entities within the above lists by linking them to the DBpedia linked open data set, and c) constructing a combined data set by merging the individual ranking list data sets using their DBpedia URI as a primary key.

Keywords: University rankings, Web data extraction, Entity linking, Linked open data, Semantic Web.

1 Introduction and Problem Definition

University / College / Higher Education rankings are rankings of institutions in higher education, ordered by combinations of factors, such as measures of wealth, research excellence and/or influence, student choices, eventual success and/or demographics, on surveys, and others. Rankings are conducted by various organizations, such as news media, websites, governments, academics and private corporations. Rankings can evaluate institutions within a single country and / or region, or worldwide. In this paper we consider worldwide / global university rankings.

Due to huge financial and sometimes political interests, the rankings of universities worldwide recently received increasing attention. The rankings are based on different criteria and collect data in various ways. As a result, there is a large divergence in the specific rankings of different institutions. Therefore, rankings have produced much debate about their usefulness and accuracy. The expanding diversity in rating

methodologies and accompanying criticisms of each indicate the lack of consensus in the field. In order to compare rankings so that safe conclusions about their reliability are drawn, data from the sites of different such ranking lists must be collected and then statistically tested [1, 2, 3, 4, 9, 10, 19, 21, 22].

In this paper we present the first step needed in order to compare university rankings, which is data collection. Actually, we have developed a Prolog application, called URank after "University Ranking", using SWI-Prolog [24], that a) extracts data from the various ranking list web sites, b) uniquely identifies the University entities within the above lists, and c) constructs a combined data set that can be fed to the statistical comparison test. The actual comparison of rankings is beyond the scope of this paper; an initial report of a statistical comparative analysis of rankings (based on the data collection methodology described in this paper) can be found at [2].

Table 1 contains the University ranking lists we have used in this study, performed during academic year 2012-2013. In order to collect data from all those different ranking lists / sites several technical challenges exist. First of all is the acquisition of data, which are published in heterogeneous ways and formats. Usually, there are no downloadable and /or structured data, which in most cases must be extracted (scrapped) from HTML pages. To this end, web data extraction tools must be employed [7]. In our case, we have used DEiXTo [12], a powerful web data extraction tool based on the W3C DOM. It allows users to create highly accurate "extraction rules" (wrappers) that describe what pieces of data to scrape from a website. Actually, we have used only the GUI of DEiXTo, a friendly graphical user interface that is used to manage extraction rules (build, test, fine-tune, save and modify). Then, we have used the extraction rules built with DEiXTo GUI for the wrapper component of URank to extract data at run-time.

Table 1. University Ranking Lists used in the paper.

Acronym	**Full name**	**URL**	**Collected Universities**
ARWU	Academic Ranking of World Universities	www.shanghairanking.com	500 / 500
Leiden	CWTS Leiden Ranking	www.leidenranking.com	500 / 750
QS	Quacquarelli Symonds	www.topuniversities.com	600 / 800
THE	Times Higher Education	www.timeshighereducation.co.uk	400 / 400
URAP	University Ranking by Academic Performance	www.urapcenter.org	750 / 2000
Webometrics	Ranking Web of Universities	www.webometrics.info	600 / ~12000

Data acquisition also "suffers" from the heterogeneity of the schemata of the data extracted from the various sites. In order to resolve this we have developed a small OWL ontology that describes ranked universities homogeneously and we have customized extraction rules (using Prolog) in order to map the extracted data (sometimes using tailored transformations) into this common schema. Actually, as a byproduct of our project, each extracted data set takes the form of RDF data that can

be published into the Linked Open Data (LOD) cloud individually from the rest of the datasets.

The second and third challenges depend on each other. In order to merge different ranking lists into a single table (third challenge) one has to find a unique identification key for the Universities along the different ranking lists (second challenge). This is not a trivial task, since the names used in the different ranking lists are not always the same. For example, in the ARWU list (Table 1) the Imperial College[1] is mentioned as "The Imperial College of Science, Technology and Medicine", whereas in the QS list it is mentioned as "Imperial College London". In order to find a unique primary key for each list that can be safely used across datasets in order to merge them together, we should consider finding a unique immutable identifier for each University entity. We decided to consider DBpedia[2], a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web. DBpedia offers the ability to ask sophisticated queries against Wikipedia and to link the different data sets on the Web to Wikipedia data. So, linking the entities extracted from the different ranking datasets to DBpedia could serve two goals: a) linking the data extracted in the first step with a very well-known and rich linked open dataset, and b) using the DBpedia ID (actually a URI) as a unique primary key across datasets to enable dataset merging.

Linking entities to DBpedia is not a trivial task either. DBpedia (and Wikipedia) contain crowd-sourced data, so they not always accurate or complete. For example, there might cases where a DBpedia entity that represents a University is not classified correctly under the University or Educational Institution class, but to a class higher in the hierarchy of the DBpedia ontology (e.g. *owl*:*Thing*). Furthermore, there might be synonym Universities in different places (e.g. Newcastle University[3] in the UK, University of Newcastle[4] in Australia) or there might be University mergers or splits along history, whose names still appear for historical reasons in Wikipedia and DBpedia (e.g. University of Paris[5] which split in 1970 into 13 Universities named very similarly some times as "University of Paris I, II, …").

In order to resolve all the above issues, general purpose entity linking software, such as DBpedia Spotlight [14] or SILK [23], cannot possibly have a 100% accuracy, simply because domain-specific knowledge on University naming, geographical reasoning and temporal reasoning (to name a few), must be used additionally to disambiguate University entities in DBpedia. Even using domain-specific knowledge, sometimes the official DBpedia dataset does not contain up-to-date information because Wikipedia articles are constantly being revised, so when some pieces of information cannot be found at DBPedia, DBpedia Live[6] is used. Finally, when neither DBpedia nor DBpedia Live can provide a satisfactory disambiguation for an entity, URank uses Wikipedia text search (which proved to be better than DBpedia's text search) and web extraction techniques to find better candidate entities.

---

1 http://www3.imperial.ac.uk/
2 http://dbpedia.org/
3 http://www.ncl.ac.uk/
4 http://www.newcastle.edu.au/
5 http://en.wikipedia.org/wiki/University_of_Paris
6 http://wiki.dbpedia.org/DBpediaLive

In the rest of the paper, we present the architecture and functionality of the URank system in section 2, we report on the extensive evaluations we have performed on URank, and finally we conclude with a critical discussion on the ability to extend URank to become a general purpose tool, some thought for future work and a small comparison to relevant systems.

2 URank Architecture and Functionality

The architecture of the URank application is shown in Fig. 1. The main components of our system are: a) the Web data extractor or *Entity Extractor*, that extracts the University entities from the ranking sites, b) the *Entity Linker*, that links the extracted University entities with DBpedia entities, and c) the *Entity Merger*, that generates a single entity for each University by merging the different datasets, using the DBpedia entity URI as a primary key. In the following subsections we present in detail each of these components.

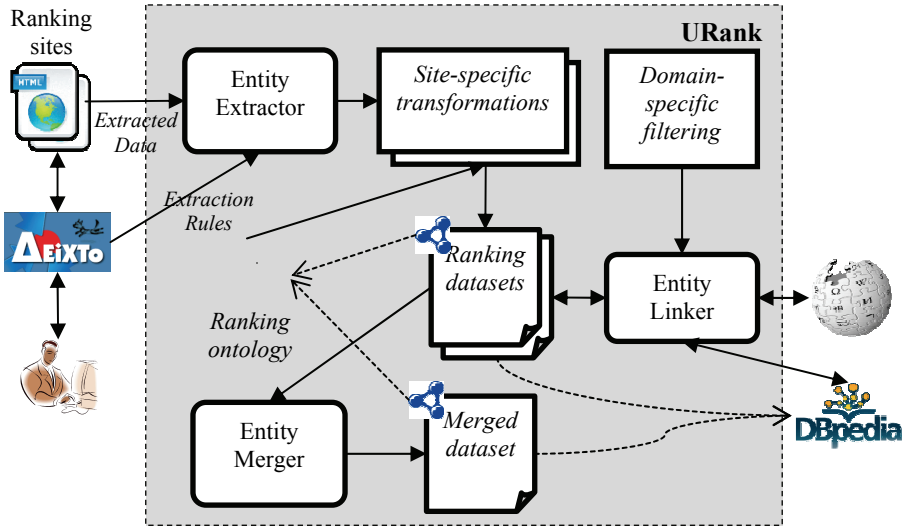

Fig. 1. URank Architecture

2.1 Entity Extractor

The Entity Extractor is the component of URank that acquires needed data from University ranking sites. The Entity Extractor is driven by users who use the DeiXTo GUI in order to define site-specific extraction rules for each ranking list web site. Fig. 2 shows an example of using DeiXTo for defining an extraction rule for the ARWU site. More details about using DeiXTo are beyond the scope of this paper and

can be found at DeiXTo site[7] and at reference [12]. What is important to notice is that the extraction rule (or pattern) defined using DeiXTo is exported in an XML file (Fig. 3). The contents of this file are fed to the Web data extractor component of URank which uses the XML, XPath and http libraries of SWI-Prolog to extract the data from the ranking sites. Although the interpretation of the XML DeiXTo extraction rules by our wrapper component involves a rather sophisticated algorithm, its detailed presentation is beyond the scope of this paper. For each site the name of the University, its global rank and its country is collected. Notice that countries are needed for name disambiguation purposes later, as already discussed in the introduction. Furthermore, the URL that contains details about the specific University is also extracted, in case the data transformation component needs to access it for disambiguation purposes.

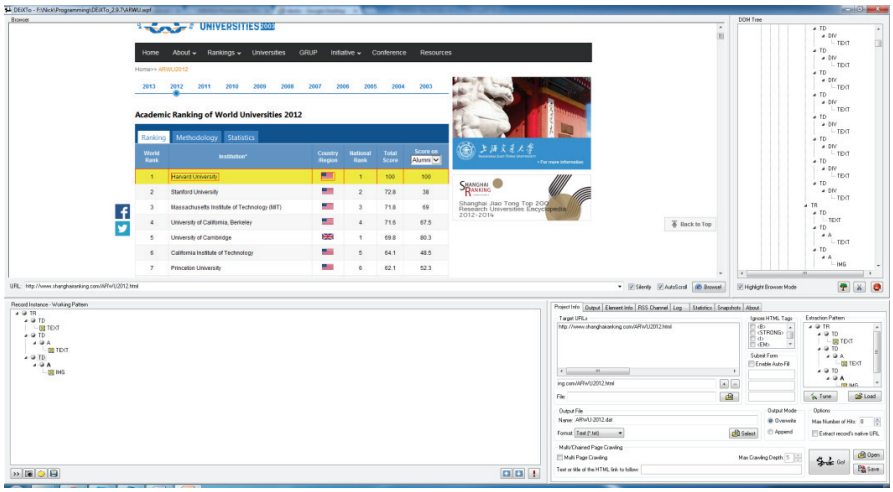

Fig. 2. DeiXTo GUI screenshot for defining extraction rule for the ARWU site

The extraction rules differ a lot, depending on the site. In the simplest case, such as ARWU (Table 1), all data are found in a single page. However, there are cases where data are found in several linked pages, such as Webometrics; therefore, web extraction must load and scrape several consecutive pages, navigating through a "next"-type link. In the case of THE list there is no "next" link, so the pages of all the ranges of ranking must be manually collected and fed to the extractor.

After the extractor completes the task of retrieving every piece of raw data that can be retrieved from the ranking web sites, site-specific transformations clear and homogenize the data in order to create the site-specific datasets in RDF. These transformations mostly deal with converting the retrieved country-related data into a proper country name, common across the different ranking sites. For example, in ARWU the country information is retrieved as a URL that contains all the

[7] http://deixto.com/

Universities of this specific country contained in the ARWU list[8]. In this case, specific string processing rules retrieve the name of the country. Other sites, such as Webometrics, do not have a link to country pages / profiles, but they just show the flag of the country, using a short country code in the image URL[9]. In this case, string processing isolates the country code and a transformation table derived from the ISO 3166 Country Codes standard[10] transforms it into a proper country name.

```
<!DOCTYPE Project SYSTEM "wpf.dtd">
<Project>
    <TargetUrls>
       <URL Address="http://www.shanghairanking.com/ARWU2012.html"/>
    </TargetUrls>
    <MultiplePage Enabled="false" ContainsText="" MaxCrawlDepth="5"/>
    <ExtractionPattern>
       <Node tag="TR" stateIndex="grayed" IsRoot="true">
          <Node tag="TD" stateIndex="grayed">
             <Node tag="TEXT" stateIndex="checked"/>
          </Node>
          <Node tag="TD" stateIndex="grayed">
             <Node tag="A" stateIndex="grayed">
                    <Node tag="TEXT" stateIndex="checked"/>
             </Node>
          </Node>
          <Node tag="TD" stateIndex="grayed">
             <Node tag="A" stateIndex="grayed">
                <Node tag="IMG" stateIndex="checked"/>
             </Node>
          </Node>
       </Node>
    </ExtractionPattern>
    <OutputFile Filename="ARWU-2012.dat" Format="TabDelimited"/>
</Project>
```

Fig. 3. DeiXTo extraction rule for the ARWU site

The rest of the site-specific transformations deal with clearance of the University names, such as removing extra spaces, transforming names from URL to ASCII encoding, removing trailing numbers from Webometrics entries when Universities maintain multiple web domains[11], etc. Finally, in the case of Leiden the main ranking page used to contain abbreviated University names only, while full names could be found in the detailed pages of the Universities. Therefore, data transformation included additional web data extraction activities. In the current version of the Leiden ranking site full University names are included in the main ranking page as tooltips.

After extracted data are cleared and transformed the individual datasets for each ranking site can be constructed. These datasets are in RDF and can be published in the LOD cloud. In order to have a common schema for all sites, we have developed a lightweight University ranking ontology which consists of two classes (Fig. 4): *RankingOrganization* and *RankedInstitution*. The former has six instances,

[8] E.g. http://www.shanghairanking.com/World-University-Rankings-2012/USA.html
[9] E.g. http://www.webometrics.info/sites/default/files/logos/us.png
[10] http://www.iso.org/iso/country_codes.htm
[11] http://www.webometrics.info/en/node/36

representing the six ranking list/sites of Table 1 included in this study. The latter will have as many instances as per University entities extracted from each ranking site. Table 2 presents the properties for the two classes, while Fig. 5 shows the instance of the *RankingInstitution* class for the ARWU list. Notice the use of the *dc:title* property for the name of the ranking institution and *owl:sameAs* property for linking our datasets to the LOD cloud, i.e. the DBpedia entry for the ranking list. Instances for the *RankedInstitution* class will be shown later, after the entity linking with DBpedia entries is discussed.

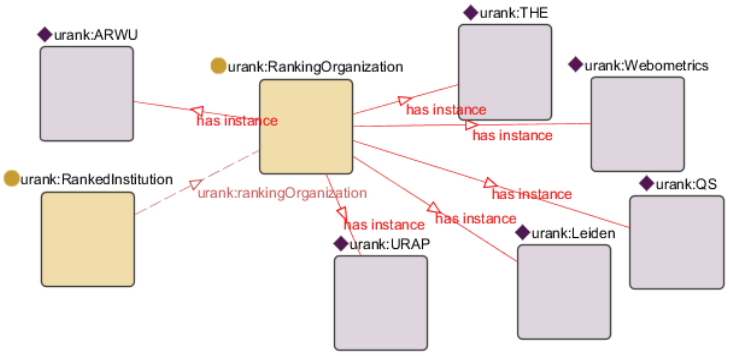

Fig. 4. The University Ranking Ontology and the 6 ranking list instances

Table 2. Properties of the University Ranking Ontology

Property	Domain	Range
hasURL	RankingOrganization	xs:anyURI
rankingOrganization	RankedInstitution	RankingOrganization
rankURL	RankedInstitution	xs:anyURI
institutionRegion	RankedInstitution	xs:string
rank	RankedInstitution	xs:int

```
Individual: urank:ARWU
Types: urank:RankingOrganization
Facts:
 dc:title "Academic Ranking of World Universities"^^xs:string
 urank:hasURL  "http://www.shanghairanking.com/"^^xs:anyURI
SameAs:
 dbpedia:Academic_Ranking_of_World_Universities
```

Fig. 5. The ARWU instance of *RankingOrganization* class

2.2 Entity Linker

Linking our dataset entities to DBpedia is performed as indicated by Algorithm 1, which is a non-formal high-level description of the main DBpedia matching algorithm. The algorithm consists of two main loops, for each ranking list and for each University entry retrieved from each list. Inside the second loop, there are a number of steps to retrieve matching DBpedia entries using 3 different approaches, explained later. At each step, if a satisfactory match is found the algorithm terminates

Algorithm 1. The basic matching algorithm.

1. **For each** Ranking list *R*
2. **For each** University *U* **in** *R*
3. Candidate Universities *Cand* = ∅
4. Find *Top-N1* instances of *EducationalInstitution* class using DBpedia lookup service with keyword *U.name*. Each instance must adhere to domain-specific spatiotemporal restrictions.
5. **If** *CheckMatchFound*(*Top-N1*) **then** *Result* = *CheckMatchFound*(*Top-N1*); **break**; **else** *Cand* = *Cand* ∪ *Top-N1*
6. Find *Top-N2* instances of *EducationalInstitution* class using DBpedia SPARQL endpoint and a query template with list of words from *U.name*. Each instance must adhere to domain-specific spatiotemporal restrictions.
7. **If** *CheckMatchFound*(*Top-N2*) **then** *Result* = *CheckMatchFound*(*Top-N2*); **break**; **else** *Cand* = *Cand* ∪ *Top-N2*
8. Find *Top-N3* entries in Wikipedia using keyword search (keyword *U.name*)
9. *Top-N3'* = ∅
10. **For each** alternative University name *A* **in** *Top-N3* find the corresponding DBpedia entry *A'* and form *Top-N3'*
11. Check if Wikipedia entry *A* corresponds to a DBPedia entry *A'* with URI transformation. *A'* must adhere to domain-specific spatiotemporal restrictions.
12. **If** the above is true **then** *Top-N3'* = *Top-N3'* ∪ {*A'*} **Else** Repeat steps 4, 6 using alternative name *A*. Give priority to *EducationalInstitution* instances. In case of failure relax this requirement. For each alternative name *A*, a set of *Top-N1'* and *Top-N2'* entries will be returned. *Top-N3'* = *Top-N3'* ∪ *Top-N1'* ∪ *Top-N2'*
13. **end for**
14. **If** *CheckMatchFound*(*Top-N3'*) **then** *Result* = *CheckMatchFound*(*Top- N3'*); **break**; **else** *Cand* = *Cand* ∪ *Top-N3'*
15. Score each *C* **in** *Cand* using the string distance between *U.name* and *C.name*, source of *C* and the method that *C* was obtained
16. Sort *Cand* in descending score order
17. *Result* = *Cand*[1]
18. **end for**
19. **end for**

immediately and returns that match for each University. Otherwise, candidate DBpedia matching entries are collected into a candidate set, scored according to our own scoring function and then the best scored candidate is returned as a match.

Table 3. Thresholds for satisfactory maching.

Steps	Threshold
4, 6	0.98
14 (*EducationalInstitution* instances)	0.97
14 (*owl*:*Thing* instances)	0.99
14 (search Wikipedia via Google)	1.00

The "satisfactory match" (*CheckMatchFound* function in Algorithm 1) for a DBpedia entry depends on the string distance between the name of the University extracted from the ranking list and the name of the matching DBpedia University entry. The distance threshold depends on the step of the algorithm. More specifically, in order to measure string distance we use the *isub*/*4* built-in function of SWI-Prolog, which is based on a string metric for ontology alignment [20]. This metric is more appropriate in our case than the Levenshtein distance metric, since it mainly concerns substring matching, which is appropriate for matching names of Universities. For example, recall the case for "Imperial College" from the introductory section. Table 3 shows these thresholds, which have been experimentally found and are very high to ensure termination only for almost absolutely certain matches.

The two main methods for retrieving DBpedia entities by matching extracted University names is a) DBpedia lookup service[12] and b) OpenLink Virtuoso built-in SPARQL endpoint, on the DBpedia host instance[13], using a template query derived from the Faceted Browser, and Search & Find Service[14]. For the DBpedia lookup service the query template is:

```
http://lookup.dbpedia.org/api/search.asmx/KeywordSearch?
      QueryClass=<Class>&MaxHits=<Top-N1>&QueryString=<U.Name>
```

Notice that the above query returns results in an XML file, which is parsed using the same SWI-Prolog libraries as for extracting Universities from HTML files, above.

The query template for DBpedia SPARQL endpoint is as follows:

```
1. select ?u, ?n where {
2.      ?u rdf:type <Class> .
3.      ?u ?p ?v .
4.      ?v bif:contains <U.Name.Words> option (score ?sc) .
5.      ?u rdfs:label ?n .
6.      FILTER (lang(?n) = "en") }
7. order by desc (?sc*0.3+sql: rnk_scale(<LONG::IRI_RANK> (?u)))
9. limit <Top-N2>
```

[12] http://wiki.dbpedia.org/lookup/
[13] http://dbpedia.org/sparql
[14] http://dbpedia.org/fct/

In the above query, `?u` is the URI of the matched DBpedia entry and `?n` its name. The query retrieves the values `?v` of all properties `?p` of the University and searches them for words contained within the extracted University name (<*U.Name.Words*>) using Virtuoso's built-in *bif*:*contains* predicate.

In the above searches, the query class <*Class*> is *EducationalInstitution* for steps 4, 6 and *owl*:*Thing* for the relaxed search in step 12. Furthermore, the maximum number of hits <*Top-N1*> and <*Top-N2*> are 2 for steps 4, 6 and 4 for the relaxed search in step 12 and they have been established experimentally. Furthermore, in the case of step 12, at line 3 in the SPARQL query template property `?p` becomes *rdfs*:*label*; therefore, search concentrates only on the property that contains the name of the University. Notice that in step 12, DBpedia is searched using as a keyword the name of a Wikipedia-retrieved University, not the name of the originally-retrieved University.

During all DBpedia searches (steps 4, 6, 12), the retrieved instances are filtered according to spatiotemporal domain-specific constraints. Namely, the retrieved DBpedia University must be located in the same country as the University extracted from the site and it must also be still operating. The check for the latter is performed by checking if the property *dbpprop*:*closed* exists. Of course, this is not always the case for all closed / suspended Universities, such as the *dbpedia*:*University_of_Paris*, for example. When such information does not exist, then URank is susceptible to errors, unless a better match is found.

The check for location / country compatibility is not always easy, since DBpedia entries stem from Wikipedia articles and sometimes the infoboxes of these articles are not complete. For example, University DBpedia entries may not have a country-related property, but only City- or State-related information (USA and Spanish Universities, mainly). Therefore, spatial inclusion reasoning must be employed in this case, with additional SPARQL queries to find out in which country a city or State is located, etc. Furthermore, sometimes there are multiple entries for the same entity in DBpedia, similarly to Wikipedia. In this case, the original DBpedia search may not retrieve the entry with the country-related information. In this case redirection links are followed and the country-related search is repeated. Table 4 summarizes the properties used for retrieving the country of the University DBpedia entry.

Table 4. DBpedia properties related to Location.

Location Information	**DBpedia properties**
Country	dbpedia-owl:country, dbpprop:country
State	dbpedia-owl:state, dbpprop:state
City	dbpedia-owl:city, dbpprop:city
Location	dbpedia-owl:location, dbpprop:location
{*Redirection to another instance*}	owl:sameAs, dbpedia-owl:wikiPageRedirects

To make things even more difficult, the retrieved country information may not be exactly the same as the country data extracted from the raking site. For example, the country information associated with *dbpedia*:*Harvard_University* is "U.S.", while the country data for this University from the ARWU list is "USA". So, there is a need to create a compatibility matrix for country names. This can be done only empirically / experimentally by collecting compatible names for some countries. The majority of

country names though do not have such a synonymy problem. We do not include this matrix in the paper because some synonyms we came up with are not "politically correct" and may raise conflicts.

There are a few more subtle domain-dependent filtering criteria that must be taken into account, such as using Roman or Arabic numbers in University names (e.g. "University of Montpellier II" vs. "Montpellier 2 University") or using synonyms for the word University in other languages (e.g. "University of Freiburg" vs. "Universität Freiburg"). These are also crucial for achieving a 100% precision and recall, but are too detailed heuristics to be presented here.

In case steps 4 and 6 of Algorithm 1 do not retrieve a high match, step 8 uses the keyword search engine of Wikipedia to retrieve Wikipedia articles as candidates for alternative (and possibly better) names for the retrieved Universities. For example, the ARWU list contains the entry "University of Paris Sud (Paris 11)". This does not return any result at DBpedia lookup service. Even if the string in parenthesis is stripped, because for the ARWU list it is considered a synonym (therefore, redundant), the DBpedia lookup service returns the entry *dbpedia*:*Paris-Sud_11_University*, with label "*Paris-Sud 11 University*". The string distance between the two names is 0.93, which is lower than the thresholds in Table 3. A query to Wikipedia returns as the best result the page with title "University of Paris-Sud"[15], which corresponds to the DBpedia entry *dbpedia*:*University_of_Paris-Sud*, with *rdfs*:*label* "*University of Paris-Sud*". The string distance now between the Wikipedia article title and the DBpedia entry label is exactly 1, therefore above the threshold of Table 3.

Of course, things are not simple here either. Wikipedia is asked to return <*Top-N3*> articles with *Top-N3* found experimentally to be 3 with the following query:

```
http://en.wikipedia.org/w/index.php?search=<U.name>&
      limit=<Top-N3>&go=Go
```

Sometimes Wikipedia just returns the most probable result, when its score exceeds some threshold. When this happens, the returned page is scraped to extract the article title and to check whether it involves indeed a University, located in the same country as the University extracted from the ranking site and still operational, namely using the same domain-dependent filters as in steps 4, 6. However, in the case of Wikipedia this is done by scraping the HTML of the returned page, and most specifically, the infobox and the categories box. For example, see Fig. 6 for an active public University located in Australia, the University of Sydney[16], and Fig. 7 for a suspended University. Furthermore, general pages such as "*Template*:…", "*List of Universities in …*", "*Higher education in …*", and similar ones, must be excluded, along with disambiguation pages. When a single result page does not exist, Wikipedia returns a list of results and the above checks are performed for the *Top-N3* results.

Finally, if the Wikipedia keyword search does not generate any alternative names due to all the above restrictions, Google search restricted in the Wikipedia domain is used as a last resort, using the query below and concentrating on the first result:

[15] http://en.wikipedia.org/wiki/University_of_Paris-Sud
[16] http://sydney.edu.au/

```
http://www.google.com/search?as_q=<U.name.words>&
      as_sitesearch=en.wikipedia.org
```

After step 8, list *Top-N3* contains alternative names / entries for the original University retrieved from the ranking site. These Wikipedia entries should lead to DBpedia entries, possibly giving better results than the original University name. This is the task of the loop in steps 10-13. There are 2 ways to map Wikipedia entries to DBpedia entries. The first one is direct and rewrites the Wikipedia URL to a DBpedia URI:

```
http://en.wikipedia.org/wiki/<Univ> →
          http://dbpedia.org/resource/<Univ>
```

However, before the DBpedia URI is considered a final match it must be verified for the same domain-specific restrictions already discussed above for steps 4 and 6. If the verification step succeeds then this DBpedia entry is added to list *Top-N3'* which contains candidate matching DBpedia entries. If the verification fails, then at step 12, which is the second way to map a Wikipedia entry to a DBpedia entry, the Wikipedia article titles are used as alternative University names that lead to new DBpedia searches with these alternative names, as in steps 4, 6. From this search, new candidate matching DBpedia entries are retrieved, which are added to list *Top-N3'*.

Fig. 6. Infobox and categories box for the University of Sydney Wikipedia entry[17]

Fig. 7. Infobox for the University of Paris Wikipedia entry[18]

Notice that actually step 12 performs two searches (using both search methods): one strict with *EducationalInstitution* as the target class and one relaxed with

[17] http://en.wikipedia.org/wiki/University_of_Sydney
[18] http://en.wikipedia.org/wiki/University_of_Paris

owl:*Thing* as the target class. The results for the two searches are scored differently, giving higher score to the stricter search, as it will be discussed later. The reason for this is that step 12 is the last chance of the algorithm to discover a match, so in case the strict match does not retrieve any DBpedia instances, the result of the relaxed search will cover for it. Outside the loop, at step 14, list *Top-N3'* is checked for immediate results, i.e. results that give a string distance above the threshold of Table 3.

The last important step of the matching algorithm is the scoring function for the candidate DBpedia entities collected into set *Cand*. Recall that entities in this list have string distances less than the thresholds of Table 3; otherwise, the algorithm would have stopped and returned a confirmed match. So, the purpose of step 15 is to score each candidate match using the string distance between the University name of the DBpedia entry and the name of the University exported from the ranking site or the alternative name retrieved from Wikipedia articles (in step 8). Furthermore, the scoring function takes into account a) the source of the candidate match (original or alternative name from Wikipedia), b) the method that the candidate match was obtained (DBpedia lookup service, SPARQL endpoint, and direct transformation of Wikipedia URL to DBpedia URI), and c) if the search was strict or relaxed, concerning the target class. Table 5 summarizes the score additions that each of the above dimensions adds to the string distance metric, which is in the range between 0 and 1. For example, when a candidate match was obtained from the original University name retrieved from the ranking site using a strict search at the DBpedia lookup service (step 4) and the string distance of the candidate match from the University name is 0.92, the total score is 1000+200+10+0.92=1210.92. On the other hand, if a candidate match is coming from step 11 (Wikipedia search, direct URL/URI transformation) with a 0.95 string distance, the total score is 2000+200+20+0.95=2220.95. Notice that direct searches of step 11 and DBpedia searches in steps 4 and 6 are always strict.

Table 5. Score additions along various dimensions.

Dimension	Value	Score addition
Source	Original (ranking site)	1000
	Wikipedia / Google search	2000
Target class	Strict	200
	Relaxed	100
Query method	DBpedia lookup service	10
	SPARQL endpoint	10
	Direct URL/URI transformation	20

From Table 5 it is obvious that when steps 4, 6 fail to produce a confirmed match, then priority is given to candidate matches coming from Wikipedia retrieved alternative University names, since Wikipedia keyword search engine is better than DBpedia's free text search engine. Furthermore, strict searches are preferred to relaxed searches, for obvious reasons. Finally, direct URL/URI transformations (valid only in step 11) are preferred to DBpedia searches (step 13), since the latter may introduce more noise due to free text search. Notice that all these preferences have been experimentally established and evaluated.

Finally, after Algorithm 1 terminates, the RDF datasets for each ranking site are generated and saved permanently. Fig. 8 shows the RDF code for the "Imperial College London" entry of the Leiden ranking site dataset. In the future, these datasets will be uploaded into an RDF triplestore with a public SPARQL endpoint.

```
<urank:RankedInstitution
     rdf:about="&urank;Imperial%20College%20London"
     dcterms:title="Imperial College London">
  <urank:institutionRegion rdf:datatype="&xsd;string">United Kingdom
     </urank:institutionRegion>
  <urank:rank rdf:datatype="&xsd;int">41</urank:rank>
  <urank:rankingOrganization rdf:resource="&urank;Leiden"/>
  <owl:sameAs rdf:resource="&dbpedia;Imperial_College_London"/>
</urank:RankedInstitution>
```

Fig. 8. Leiden dataset RDF entry for the "Imperial College London"

2.3 Entity Linker

The Entity Merger component of URank takes as input the datasets of the 6 ranking sites and produces a single dataset that contains all the Universities with all the rankings from every ranking site contained in a single entity. For example, the merged dataset entry for the "Imperial College London" is shown in Fig. 9. The properties for the merged dataset are slightly different from the individual datasets. Specifically, there is no country-related information, since the purpose of the merged dataset is to statistically compare rank positions, and there is no direct link to the *RankingOrganization* instance, since each entry is ranked by multiple ranking organizations. Furthermore, there are 6 new properties, which hold the ranks of the individual ranking sites. All these are sub-properties of the *urank*:*rank* property and are added to the ontology. For example, in Fig. 9 the Imperial College entity has a *urank*:*rankTHE* property for the THE ranking list, a *urank*:*rankQS* property for the QS list, etc. The following piece of OWL code shows how the *urank*:*rankeTHE* property is defined:

```
<owl:DatatypeProperty rdf:ID="rankTHE">
    <rdfs:supbPropertyOf rdf:resource="#rank"/>
</owl:DatatypeProperty>
```

The merge of the datasets into a single one is performed with Algorithm 2. For each RDF graph *R* that holds the corresponding ranking dataset (step 1) and for each University instance in this dataset (step 2), a new University instance is created in the merged dataset graph *M* and the appropriate property values are copied (step 5). From the second iteration of the outer loop and onwards, it might be the case that the

University instance already exists in the merged dataset from previous iterations (step 3). In that case, only the corresponding rank property is copied (step 4).

```
<urank:RankedInstitution
                rdf:about="&urank;Imperial%20College%20London"
                dcterms:title="Imperial College London">
  <urank:rankARWU rdf:datatype="&xsd;int">24</urank:rankARWU>
  <urank:rankLeiden rdf:datatype="&xsd;int">54</urank:rankLeiden>
  <urank:rankQS rdf:datatype="&xsd;int">6</urank:rankQS>
  <urank:rankTHE rdf:datatype="&xsd;int">8</urank:rankTHE>
  <urank:rankURAP rdf:datatype="&xsd;int">14</urank:rankURAP>
  <urank:rankWebometrics rdf:datatype="&xsd;int">261
  </urank:rankWebometrics>
   <owl:sameAs rdf:resource="&dbpedia;Imperial_College_London"/>
</urank:RankedInstitution>
```

Fig. 9. Merged dataset RDF entry for the "Imperial College London"

Algorithm 2. The algorithm form merging the individual ranking datasets into one.

1. **For each** individual RDF graph *R* (or the corresponding ranking site)
2. **For each** instance of *urank*:*RankedInstitution U* **in** *R*
3. Check if there is an instance *U'* of *urank*:*RankedInstitution* in the merged dataset graph *M*, such that *U'.owl*:*sameAs* = *U. owl*:*sameAs*
4. **If** yes, **then** *U'.urank*:*rank*<*R*> = *U.urank*:*rank*
5. **If** no, **then**
   Create a new *U'* instance of *urank*:*RankedInstitution* with *U'.ID* = *U.ID*;
   U'.dc:*title* = *U.dc*:*title*;
   U'.owl:*sameAs* = *U.owl*:*sameAs*;
   U'.urank:*rank*<*R*> = *U.urank*:*rank*;
   end if
6. **end for**
7. **end for**

Notice that the existence check is based on the value of the *owl*:*sameAs* property which is a link to the DBpedia entry for the University, discovered by Algorithm 1. So, it is important to ensure that all DBpedia URIs for the same University are exactly the same. Although this seems expected, it is not always the case. Sometimes there exist many Wikipedia articles for the same topic, which are redirected to a single Wikipedia page. This is also reflected to the corresponding DBpedia instances. For the same real-world entity there might be several DBpedia instances that re-direct to (possibly) a single DBpedia entry through the *dbpedia-owl*:*wikiPageRedirects* property. Thus, the entity linking process (of the previous sub-section) ensures that when a DBpedia URI is matched to a University name, a pointer-chasing algorithm ends up to the instance at the end of the chain of the re-direction links.

Sometimes there are more-than-one instances with the above property, i.e. they are at the end of different paths of the re-direction link sub-graph for the same real-world entity. This is checked by a transitive algorithm that follows the re-direction links until it finds instances that do not re-direct to another instance. In this case, URank selects the most "informative" one, namely the one with the most triples. Another problematic case is when this re-direction sub-graph is not acyclic, something that happens rarely and usually it is temporary until the next DBpedia update. Nevertheless, we catered for this case as well using a closed set search.

3 Evaluation

In order to evaluate URank, we have performed several experiments. First of all, we clarify that we have the correct answers (namely the correct DBpedia entries) for all the ranking sites, so we are able to evaluate and compare the effectiveness of each of the search mechanisms of Algorithm 1. These correct answers have been obtained manually by first running URank and then checking manually only the entries that did not have an absolute match (string distance 1.0). Then, we have conducted for each ranking site experiments turning on and off the following features / mechanisms of URank, in many combinations:

- DBpedia lookup service
- SPARQL endpoint query
- Domain-specific restrictions
- Wikipedia keyword search

For each experiment we count the correct answers *CA* (i.e. those entries that the retrieved URIs coincide with the correct URIs), the incorrect answers *IA* (i.e. those entries that the retrieved URIs do not coincide with the correct URIs), and the unanswered entries *UA* (i.e. the ones that the experiment did not manage to retrieve any URI). Notice that we assume (and it is true for the experiments we have conducted) that all Universities do have a Wikipedia / DBpedia entry. From the above measurements we calculate the precision, recall and F-measure metrics for the queries, using equations (1) – (3), respectively.

$$precision = \frac{CA}{CA + IA} \quad (\mathbf{1})$$

$$recall = \frac{CA}{CA + UA} \quad (\mathbf{2})$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (\mathbf{3})$$

Our first experiment measures the effectiveness of each query method, namely its purpose is to compare the DBpedia lookup service against the SPARQL endpoint

query method using the template derived from the Faceted Browser, and Search & Find Service[14]. Notice that the domain-specific restrictions and the Wikipedia keyword search are turned off. Results are shown in Table 6 for the DBpedia lookup service and Table 7 for the SPARQL endpoint query. Results clearly indicate the superiority of the DBpedia lookup service in terms of Precision, for all ranking sites, and the superiority of the SPARQL endpoint query in terms of recall, for almost all ranking sites, with the sole exception of URAP. This is due to the fact that the DBpedia lookup service is stricter than the SPARQL query; therefore, it returns less results but with a better chance of being correct. The F-measure value is superior for the DBpedia lookup service, with the exception of QS list.

Table 6. Measurements for the DBPedia lookup service.

Ranking site	CA	UA	IA	Precision	Recall	F
ARWU	433	60	7	98,41%	87,83%	92,82%
Leiden	472	24	4	99,16%	95,16%	97,12%
QS	512	80	8	98,46%	86,49%	92,09%
THE	382	12	6	98,45%	96,95%	97,70%
URAP	627	112	11	98,28%	84,84%	91,07%
Webometrics	521	65	14	97,38%	88,91%	92,95%
Total / Average	**2947**	**353**	**50**	**98,33%**	**89,30%**	**93,60%**

Table 7. Measurements for the SPARQL endpoint query.

Ranking site	CA	UA	IA	Precision	Recall	F
ARWU	398	25	77	83,79%	94,09%	88,64%
Leiden	429	6	65	86,84%	98,62%	92,36%
QS	520	23	57	90,12%	95,76%	92,86%
THE	356	4	40	89,90%	98,89%	94,18%
URAP	541	123	86	86,28%	81,48%	83,81%
Webometrics	511	20	69	88,10%	96,23%	91,99%
Total / Average	**2755**	**201**	**394**	**87,49%**	**93,20%**	**90,25%**

The same conclusion is evident from Fig. 10, Fig. 11 and Fig. 12, where a graphical comparison between the two query methods is presented for the three metrics. Furthermore, in these figures we compare the performance of each of these query methods alone with their combination, namely we have conducted another set of measurements where both query methods are used in combination. Results show that precision is slightly worse, whereas recall and F-measure are better when the two query methods are combined. This happens because the DBpedia lookup service is stricter concerning its answers, whereas the SPARQL endpoint query method more

relaxed, therefore it tends to return more answers with lower accuracy, so their combination exhibits the advantages of both worlds.

Our second experiment measures the effectiveness of the domain-specific restrictions. Fig. 13 shows results for all the metrics and both query methods, with and without the domain-specific restrictions. As expected, restrictions increase the precision, since more restrictions mean more accurate results. The effect is evident for the SPARQL endpoint query, because there was a lot of room for improvement, while it is negligible for the DBpedia lookup service, since its precision was already high. The exact opposite behavior is observed for recall, which is slightly worst for the lookup service, but evidently worse for SPARQL query. This was expected, because more restricted queries mean fewer results. Overall, F-measure is slightly worse for both methods. The same behavior is observed for combining the two query methods.

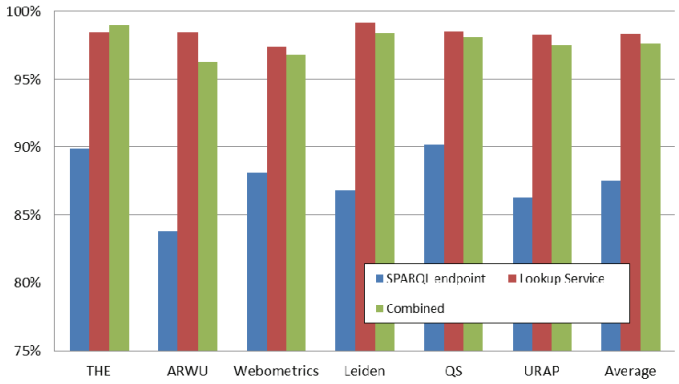

Fig. 10. Precision comparison for query methods

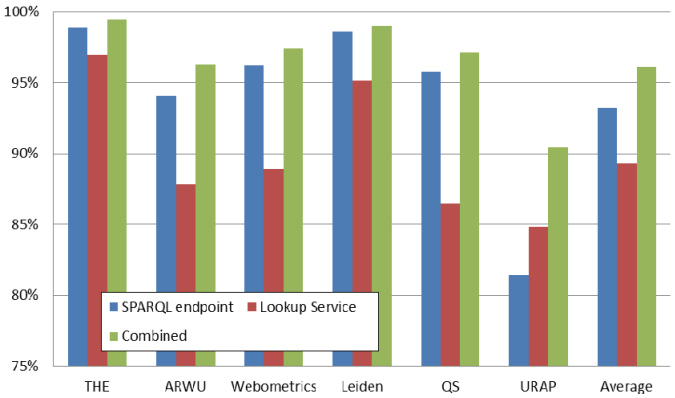

Fig. 11. Recall comparison for query methods

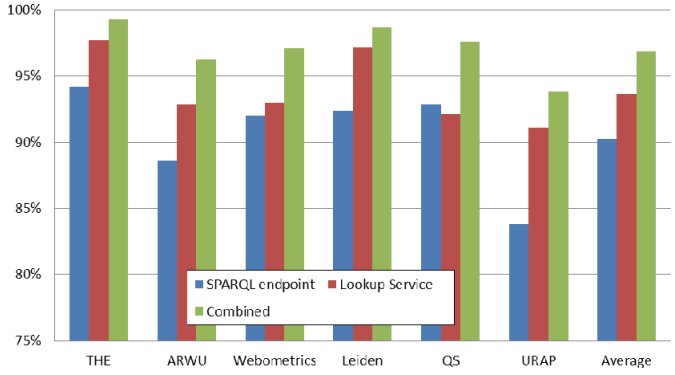

Fig. 12. F-measure comparison for query methods

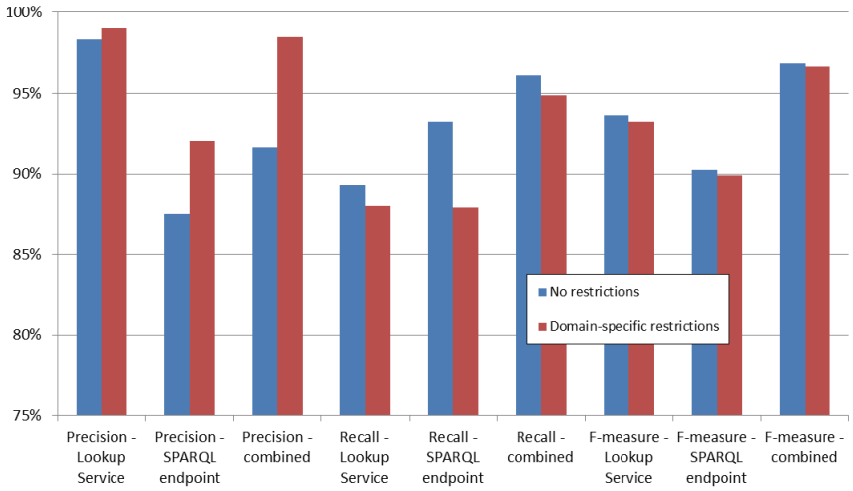

Fig. 13. Effectiveness of the domain-specific restrictions

Our last experiment measures the contribution of each of the query methods in retrieving the correct results. Table 8 shows how many correct results are due to which query method in the complete URank system. It is evident that the majority of the correct results are returned by the DBpedia lookup service (step 4), which is queried first. This choice was justified by the fact that the DBpedia lookup service has better performance (f-measure) than the SPARQL endpoint query method. The second best source of correct results is the direct URL/URI transformation (step 11) of the Wikipedia results returned after step 8. Notice that there are very few (actually 4) results that were returned after with Google search on Wikipedia. Also very few (only 2) are the correct results returned from step 12, making evident that the use of Wikipedia search is a very competent complement of the DBpedia lookup service method. Finally, there are also correct results due to SPARQL endpoint query, but

their overall contribution is very small (<4%). Fig. 14 visualizes this comparison among methods.

Table 8. Contribution of each query method to the result.

Ranking site	DBpedia Lookup	SPARQL endpoint	Wikipedia / Google	
			Direct URL/URI transformation	DBpedia Lookup/SPARQL
ARWU	418	20	61	1
Leiden	468	9	23	0
QS	485	37	78	0
THE	375	11	14	0
URAP	611	21	118	0
Webometrics	505	33	61	1
TOTAL	**2862**	**131**	**355**	**2**

Finally, Fig. 15 shows the contribution of each query method to the result of each of the experiments reported in this section. In this figure we can also compare the total correct results found by each of the tested stripped-down versions of URank, compared to the complete system. It can be concluded that the effect of using Wikipedia keyword search on top of either DBpedia lookup service or SPARQL endpoint query has almost the same result as having both of them (combined with Wikipedia). The actual results show a very small difference (1-5 less correct results, namely ~0.1%). This finding could be used to remove one of the steps 4 or 6, to increase the execution speed of URank.

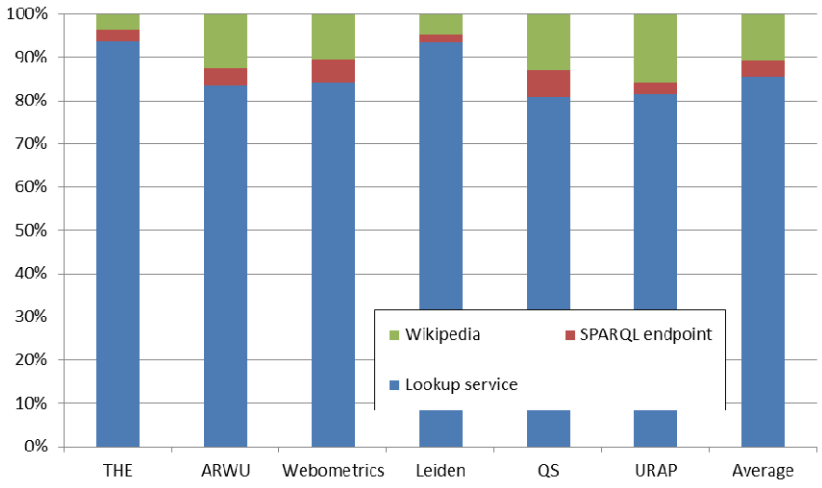

Fig. 14. Contribution of each query method to the result for the complete system

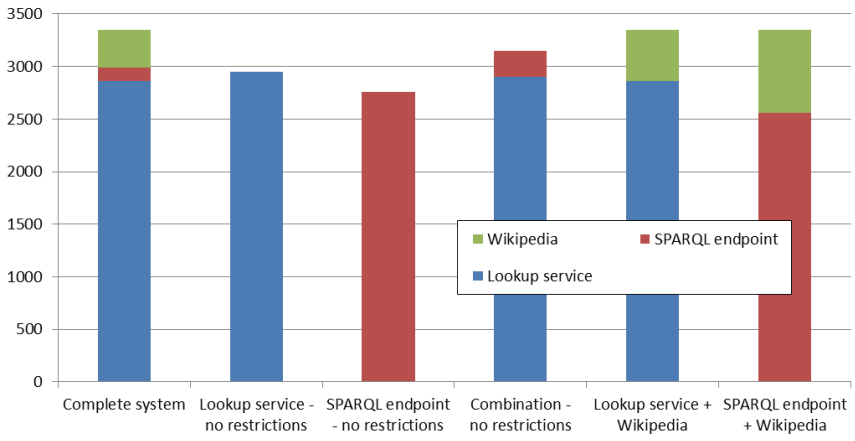

Fig. 15. Contribution of each query method to the result for each experiment

4 Concluding Discussion and Future Work

In this paper we have presented our effort to extract, link and merge University ranking datasets as Linked Open Data in the Semantic Web. This is a part of a larger project [2] that aims to statistically compare different University rankings in order to draw safe conclusions about their reliability. In order to collect the data found at different University ranking sites, we have developed, presented and evaluated a Prolog application, called URank, that a) extracts data (University entities) from the various ranking list web sites, using the DeiXTo web extraction tool [12], b) uniquely identifies the University entities within the above lists, by linking them to the corresponding DBpedia entities, and c) constructs a combined data set that can be fed to the statistical comparison test, by merging the individual ranking datasets based on the discovered DBpedia link, as a unique primary key.

In order to develop the URank system several challenges had to be met. The first one has to do with the heterogeneity of the data formats and schemata of the ranking sites, as well as the different naming schemata for the Universities and the countries they are located. These challenges were met by a) customized data extraction rules that were easily developed using the GUI of the DeiXTo system, b) site-specific data transformations that were developed in Prolog, and c) an ontology we developed for the individual ranking datasets.

The challenges concerning the unique identification of Universities using DBpedia required: a) to use the appropriate querying method with the appropriate list of words in order to search for matching entities, and b) to recognize when a correct match has been found. There are two main querying methods, the DBpedia lookup service and the SPARQL endpoint that uses a query template derived from the Openlink Virtuoso's Search & Find Service of DBpedia. Both have their own advantages and disadvantages and we decided to use them both complementary. In order to recognize

if a correct match has been found we have used the SWI-Prolog's built-in string difference metric, which is tailored to ontology alignment [20] and worked well in our case.

The main problems encountered during the development and testing of URank had mainly to do with the domain-specific nature of our search and the fact that DBpedia is a crowd-sourced knowledge base; therefore, neither correct nor complete. In order to overcome these problems we have developed a few domain-specific filters / restrictions that the retrieved entities should obey in order to be considered correct matches, on top of the string distance metric. This increased the precision of the retrieved instances, as the evaluation phase has shown. However, domain-specific restrictions lowered the recall rate, namely less correct results were retrieved because the required information was simply not present in DBpedia entries.

A major boost to recall was given by a second phase that involved searching for the University name into Wikipedia using either the Wikipedia's search engine or even Google. This was needed because the way University names are found and retrieved from the ranking sites, usually differ a lot from their formal or usual names. This search provided Wikipedia articles for the Universities where a more suitable University name (the title of the Wikipedia article) could be found. Wikipedia entries directly correspond to DBpedia entries through a trivial URL/URI transformation, so in this way better matches can be found in the vast majority of cases. Of course, extracting information from Wikipedia pages involves heuristic (thus error prone) web extraction techniques, because pages are edited by humans. However, the use of Wikipedia, along with the other querying methods and restrictions, increased our precision and recall to 100%.

Finally, a very crucial part of the entity linker is the scoring mechanism for the various DBpedia entries retrieved using all the above methods. Our scoring mechanism gives priority to a) candidate entries retrieved through either the DBpedia lookup service or a SPARQL endpoint query, with a name that is very close (≥98%) to the original University name, then to b) candidate entries retrieved from direct URL/URI transformations of Wikipedia retrieved entries, and finally to c) candidate entries retrieved by the combination of (b) and (a). The results of our evaluation show that 89.3% of the correct answers are given by method (a), 10.5% are given by method (b), and only 0.2% by method (c).

The outcome of the entity linker is the individual RDF datasets for each ranking site, linked to the DBpedia LOD dataset through the *owl:sameAs* property. In order to generate a single merged dataset so that each university includes its rank at every ranking site, a unique primary key for each University must exist across the individual datasets. However, due to the different naming schemes of the ranking sites and due to the multiple DBpedia / Wikipedia entries for the same University, the query methods of the entity linker might end up to different DBpedia entries, for the same University. Usually these DBpedia entries re-direct to a single one (following Wikipedia redirections), so following this re-direction graph gives to URank a stable primary key mechanism. The outcome of the entity merger is an RDF dataset that can be used to compare the rankings of the Universities across the different ranking sites / lists.

Looking critically at URank, we can draw the conclusion that the system has achieved its purpose, namely to correctly collect and merge the University rankings

for further statistical processing. However, one may consider the possibility of making the system independent from the domain of University rankings in the future, thus being able to collect and merge into a single table various lists of similar nature found in the Web. This needs a lot of improvements, mainly in the code itself, but also to the architecture of the system. Currently, domain-independent and domain-dependent features of the system are not so well separated in the code, despite their clear distinction in Fig. 1. This is because there are various heuristics (domain-dependent features) used in the code that need to be tightly integrated with the domain-dependent features, such as querying DBpedia and / or Wikipedia. However, this separation needs further exploration in order to develop a re-usable across domains system.

Other improvements for the system would be a) a GUI (currently the text-based interface of Prolog is used), including a tighter integration with the DeiXTo GUI, b) a more efficient and general purpose web data extractor, exploiting the full range capabilities of DeiXTo extraction rules, and c) integration with an RDF triplestore, such as Openlink Virtuoso[19] or Sesame[20] [11], so that the generated datasets to persist and be shared on the LOD cloud.

Compared to general purpose tools for entity extraction, such as [5, 13, 16, 17], URank does neither have to detect names, since the user does that using DeiXTo through its extraction rules, nor to classify the names by the type of entity (class) they refer to, since it is a domain-dependent application and extracted names are known to be University names. Therefore, URank cannot be characterized as an entity extraction application, nor it can be compared to such software.

On the other hand, URank can be considered as an entity linking software, since its purpose is to determine the identity of entities mentioned in a list of named entities, which is distinct from entity extraction / recognition in that it identifies not the occurrence of names (nor classifies them), but their reference. There are plenty of general-purpose entity-linking tools, such as [6, 14, 15, 18, 24], which usually use one knowledge base target to link entities, such as DBpedia, Wikipedia, or YAGO2 [8] using various techniques for matching (e.g. lexical) and context disambiguation (relatedness, similarity, coherence). URank instead is a domain-specific tool that focuses on a specific target type of the linked entity and takes advantage of domain-specific knowledge in order to improve precision and recall of the system. One of our main future aims is to compare the performance of URank to one or more of the above general purpose tools for entity linking. Initial experimentations with DBpedia Spotlight [14] have resulted in ~86% F-measure, which is not as good as 100% that URank achieved due to its domain-specific tweaking.

References

1. Aguillo, I.F., Bar-llan, J., Levene, M., Priego, J.L.O: Comparing University Rankings. Scientometrics 85(1), 243--256 (2010)

[19] http://virtuoso.openlinksw.com/

[20] http://www.openrdf.org/

2. Angelis, L., Bassiliades, N., Manolopoulos, Y.: Evaluation of University International Rankings (in Greek). In: Proc. Conference on Quality Assurance and Quality Management: Governance and Good Practices. Thessaloniki (2012)
3. Buela-Casal, G., Gutiérrez-Martínez, O., Bermúdez-Sánchez, M.P., Vadillo-Muñoz O.: Comparative Study of International Academic Rankings of Universities. Scientometrics 71, 349--365 (2007)
4. Cheng, Y., Liu, N.C.: Examining Major Rankings According to the Berlin Principles. Higher education in Europe 33 (2-3), 201--208 (2008)
5. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: 40th Anniversary Meeting of the Association for Computational Linguistics (2002)
6. Ferragina, P., Scaiella, U.: TAGME: On-the-Fly Annotation of Short Text Fragments (by Wikipedia Entities). In: 19th ACM Int. Conf. on Information and Knowledge Management (CIKM '10), pp. 1625--1628. ACM (2010)
7. Ferrara, E., de Meo, P., Fiumara, G., Baumgartner, R.: Web Data Extraction, Applications and Techniques: A Survey. CoRR. arXiv:1207.0246 [cs.IR] (2012)
8. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. Artificial Intelligence 194, 28--61 (2013)
9. Huang, M.-H.: A Comparison of Three Major Academic Rankings for World Universities: From a Research Evaluation Perspective. Journal of Library and Information Studies 9(1), 1--25 (2011)
10. Ioannidis, J., Patsopoulos, N., Kavvoura, F., Tatsioni, A., Evangelou, E., Kouri, I., Contopoulos-Ioannidis, D., Liberopoulos, G.: International Ranking Systems for Universities and Institutions: a Critical Appraisal. BMC Medicine 5(1) (2007)
11. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Horrocks, I., Hendler. J. A. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54--68. Springer, London (2002)
12. Kokkoras, F., Ntonas, K., Bassiliades, N.: DEiXTo: A Web Data Extraction Suite. In: 6th Balkan Conference in Informatics (BCI-2013) , pp. 9—12. ACM, Thessaloniki, Greece (2013)
13. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55--60 (2014)
14. Mendes, P.N., Jakob, M., García-Silva, A., Bizer, C.: DBpedia Spotlight: Shedding Light on the Web of Documents. In: 7th Int. Conf. on Semantic Systems (I-Semantics 2011), pp. 1--8. ACM, Graz, Austria (2011)
15. Milne, D., Witten, I.H.: Learning to Link with Wikipedia. In: 17th ACM Conf. on Information and Knowledge Management (CIKM '08), pp. 509--518. ACM (2008)
16. Nothman, J., Ringland, N., Radford, W., Murphy, T., Curran, J.R.: Learning Multilingual Named Entity Recognition from Wikipedia. Artificial Intelligence 194, 151--175 (2013)
17. Ratinov, L., Roth, D.: Design Challenges and Misconceptions in Named Entity Recognition. In: 13th Conf. on Computational Natural Language Learning (CoNLL '09) , pp. 147--155. Association for Computational Linguistics, Stroudsburg, PA, USA (2009)
18. Ratinov, L., Roth, D., Downey, D., Anderson, M.: Local and Global Algorithms for Disambiguation to Wikipedia. In: 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1 (HLT '11), Vol. 1, pp. 1375--1384. Association for Computational Linguistics, Stroudsburg, PA, USA (2011)

19. Rauhvargers, A.: EUA Report on Rankings 2011. Global University Rankings and their Impact. European University Association, Brussels (2011)
20. Stoilos, G., Stamou, G., Kollias, S.: A String Metric for Ontology Alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005, LNCS, vol. 3729, pp. 624--637. Springer, Berlin-Heidelberg (2005)
21. Stolz, I., Hendel, D.D., Horn, A.S.: Ranking of Rankings: Benchmarking Twenty-Five Higher Education Ranking Systems in Europe. Higher Education 60(5), 507--528 (2010)
22. Taylor, P., Braddock, R.: International University Ranking Systems and the Idea of University Excellence. J. of Higher Education Policy and Management 29(3), 245--260 (2007)
23. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and Maintaining Links on the Web of Data. In: Bernstein, A. et al. (eds.) ISWC 2009, LNCS, vol. 5823, pp. 650--665. Springer, Berlin-Heidelberg (2009)
24. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming – Prolog Systems 12(1-2), 67--96 (2012)
25. Yosef, M.A., Hoffart, J., Bordino, I., Spaniol, M., Weikum, G.: AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. Proc. of the VLDB Endowment 4(12), 1450--1453 (2011)