Entity List Completion using the Semantic Web

David Urbansky, James A. Thom, Daniel Schuster, Alexander Schill

Dresden University of Technology
RMIT University
{david.urbansky,daniel.schuster,alexander.schill}@tu-dresden.de
james.thom@rmit.edu.au

Abstract. Entity List Completion is the task of finding and ranking related entities for a given set of seed entities. While most researchers have used semi-structured Web pages to approach this task, we describe a novel technique for finding related entities using the Semantic Web as a source. Our evaluation and analysis shows that using linked data from the Semantic Web for related entity finding outperforms current state-of-the-art systems.

1 Introduction

Entity List Completion (ELC), also known as *Entity Set Expansion* or *Related Entity Finding*, is the task of finding and ranking related entities for a given set of *seed entities* and sometimes a given narrative of the desired relation. For example, with the seeds `Homer Simpson`, `Ned Flanders`, and `Barney Gumble` we would expect an ELC system to extract more characters from the TV show "The Simpsons" such as `Bart Simpson` or `Lisa Simpson`. A given explicit description of the expected relation could, however, state that more "male cartoon character" would make the list complete. Our contribution in this paper is the development of an ELC algorithm that uses the Semantic Web as a source and we show that it outperforms current state-of-the-art approaches.

2 Related Work

Dalvi et al. [3] use the SEAL/Boo!Wa![1] system [7] to extract entity candidates from semi-structured Web pages that have similar character patterns as the seed entities. They then rank the candidates for which they found a URI in their billion triple index. The ranking is done by comparing the type of entity candidate with the target type given in the query with the seeds. If the type matches the given type from DBPedia a score of two is given, if the broader given type matches a score of one is given, and if nothing matches a score of zero is given.

[1] A live demo of ELC with SEAL can be found at `http://boowa.com/`

Google Sets[1] is another example of a Web-based set expansion technique[2]. Similarly to Boo!Wa! [3], it uses HTML patterns around the seeds to find similar entities.

As of this writing, there are very few related works pursuing the ELC task on the Semantic Web. One such system is "RelFinder" [6, 5], which takes a few seeds from DBpedia and visually shows relations among them. They use a distance measure based on the connection path length between the entities - a variant of Dijkstra's shortest path algorithm [4]. The fewer hops one has to take to arrive at another node, the more likely it is to be related.

More related to our work, Balog et al. [2] use a Semantic Web crawl to answer queries from the entity finding task. For each seed entity, given relation, and target entity type, they search for entities that match that relation. Their first approach is a simple SPARQL query that returns all entities which are either subject or object in at least one triple with the source entity and are of the target type. Their second approach relies on an exhaustive search following all links from the source entity to the target type. All instances that are on the path are considered to be entities fulfilling the given relation. While there is extensive research on the ELC task using semi-structured HTML pages from the Visible Web, researchers have just begun employing the Semantic Web for the ELC task. Their results, however, have not yet been compared existing work. We will try to close this gap by crafting an entity extraction technique which works on interlinked data of the Web of Data comparing it to state-of-the-art approaches that do not use the Linked Open Data (LOD) cloud.

3 SWELC: Semantic Web Entity List Completion

Information on the Semantic Web is structured and often contains semantics which should make it easy for machines to read and process the data. However, the information is highly distributed over thousands of Web pages. The Semantic Web contains a few major sources such as DBpedia and Freebase, but every Web page containing RDFa, for example, provides more triples for the Web of Data. In the conception of the algorithm, we assume that there is an index[3] or triple store that aggregated triples from many different sources. We then use this index as the single query point, removing the need to work with distributed data.

3.1 Detecting Ontology Concepts using Seed Entities

Fig. 1 shows the first step of SWELC in greater detail.

We will demonstrate the process with an example: $concept = Actor$, $SeedSet = \{$"`Jim Carrey`", "`Josh Brolin`"$\}$. First, we search the index for URIs with information about these seeds. From the result list we now need to find the URI

[2] Google Sets has gone offline in the time of writing this paper, it was available at `http://labs.google.com/sets`

[3] In our later experiments we use `http://sindice.com` and an index over the Billion Triple Challenge 2011 corpus

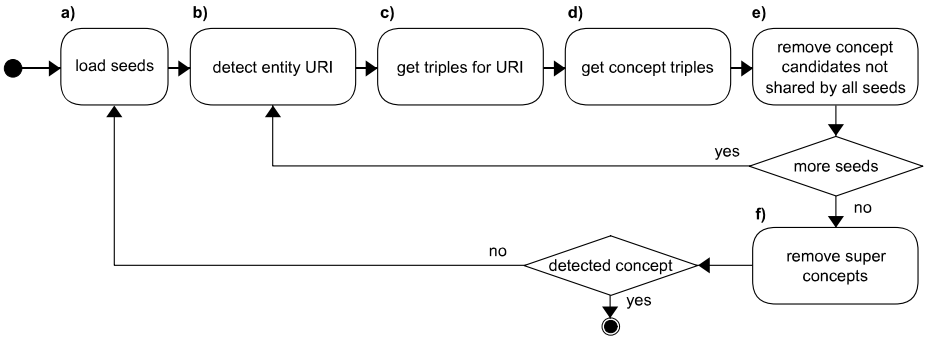

Fig. 1. The concept URI detection process of the Semantic Web Entity Extractor

which most likely is about the seed entity (Fig. 1b-d). We do this by detecting the label of the subject that is described by the URI (see Section 3.2). Only if the label matches our seed entity name exactly, do we take the URI as the subject and query the index for all triples that belong to that subject (Fig. 1c).

In our example we might have found that the URI that matches our seed "Jim Carrey" is `http://rdf.freebase.com/ns/en.jim_carrey`. We then retrieved the following predicates (p) and objects (o) for this URI (subject):

```
p: <http://rdf.freebase.com/ns/film.actor.film>
o: <http://rdf.freebase.com/ns/m.0jykww>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/film.actor>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/award.award_nominee>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/people.person>
```

In the next step, we must determine the type of our *seed*. To ascertain which type an entity belongs to, different vocabularies can be used. We want the algorithm to extract from arbitrary ontologies and use only the following ontology-independent predicates in the $TypePredicateSet$:

```
p: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
p: http://www.w3.org/2004/02/skos/core#subject
p: http://purl.org/dc/terms/subject
```

After filtering the first triple set to only contain triples with a predicate which is element of $TypePredicateSet$, we are left with the following URIs (o) as type candidates:

```
o: <http://rdf.freebase.com/ns/film.actor>
o: <http://rdf.freebase.com/ns/award.award_nominee>
o: <http://rdf.freebase.com/ns/people.person>
```

We can now process our next $seed = JoshBrolin$. We use the same procedure as with the first seed with one difference: when finding the $seedURI$ for the $seed$ we limit the results to URIs which belong to the same ontology that the first URI is from. This step is necessary for finding the common entity type among all seeds of the $SeedSet$. For example, if the first $seedURI$ is from DBpedia `http://dbpedia.org/resource/Jim_Carrey` and the second one from Freebase `http://rdf.freebase.com/ns/en.josh_brolin` their candidate types will not match since both ontologies are likely to use their own terms for the type, i.e. in DBpedia Jim Carrey is of type `http://dbpedia.org/ontology/Actor` while in Freebase his type would be `http://rdf.freebase.com/ns/film.actor`.

After processing our second seed "Josh Brolin", and finding its seed URI to be `http://rdf.freebase.com/ns/en.josh_brolin`, we can add types to the $TypeCandidateList$ from the following triples:

```
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/film.actor>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/tv.tv_guest_role>
p: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
o: <http://rdf.freebase.com/ns/people.person>
```

After adding the types to the $TypeCandidateList$, it contains the following URIs:

```
o: http://rdf.freebase.com/ns/award.award_nominee
o: http://rdf.freebase.com/ns/people.person (2 times)
o: http://rdf.freebase.com/ns/film.actor (2 times)
o: http://rdf.freebase.com/ns/tv.tv_guest_role
```

We now want to find all types that all seeds from the $SeedSet$ have in common, that is, we remove all types from the $TypeCandidateList$ that occur fewer times than $|SeedSet|$ (Fig. 1e). After this step, our set is reduced to the two types `freebase:film.actor` and `freebase:people.person`:

Our goal is, however, is to find the most precise concept among all seeds. In the final step, we resolve the URIs from the $TypeCandidateSet$ and remove all super concepts from each candidate from the set (Fig. 1f). That is we remove all concepts that are objects to the predicate `http://www.w3.org/2000/01/rdf-schema#subClassOf` for a subject from the $TypeCandidateSet$. In our example, we can remove freebase:people.person from the set after finding out that freebase:film.actor is a subclass of that concept. Our final detected concept URI is therefore `http://rdf.freebase.com/ns/film.actor`.

3.2 Extraction of Entities

Fig. 2 shows the second step of the SWELC in greater detail.

After we have detected the common type URIs for all the seeds from the $SeedSet$, we can query the index to find more entity mentions and extract them.

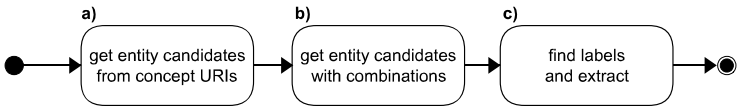

Fig. 2. The entity extraction process of the Semantic Web Entity Extractor

First we collect an *EntityCandidatesSet*. Fig. 2a shows the first method in which we resolve each URI from the *TypeCandidateSet* and add all subject URIs to the *EntityCandidatesSet*, that have a predicate which is element of *TypePredicateSet* and have a URI that is element of the *TypeCandidateSet* as object. In a second step, as shown in Fig. 2b, we query the index for all combinations of predicates and detected types.

After having collected candidate URIs in the *EntityCandidatesSet*, we need to find the corresponding label for each of the candidate URIs. This step is shown in Fig. 2c. To retrieve the label, we resolve the entity candidate URI and analyze all triples with a predicate element of *LabelPredicateSet*. Again, we want to be ontology independent and use only generic vocabulary. The *LabelPredicateSet* contains the following predicates:

```
p: http://www.w3.org/2000/01/rdf-schema#label
p: http://www.purl.org/dc/elements/1.1/title
```

If we find a label we extract the entity. If there are several labels, we search until we find the one labeled as English ("@en"). If we are not able to find the label by analyzing the entity candidate's triples, we try to guess the label from the entity candidate URI itself.

We perform these steps for each candidate entity in the *EntityCandidateSet* and move on to the ranking step when we have processed all candidates.

3.3 Ranking Extractions

Fig. 3 shows the third and last step of the SWELC in greater detail.

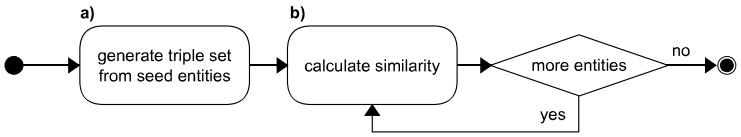

Fig. 3. The entity ranking process of the Semantic Web Entity Extractor

To rank the entities, we compute the similarity of each extracted entity with the entities from the *SeedSet* (3b). The more similar the entity, the higher the rank. As a similarity function we use the Jaccard similarity coefficient as shown in Equation 1, where $Triples_{seeds}$ is the union set of all predicates and objects (URIs and literals) that were found for the seed entities from the *SeedSet* and

$Triples_{entity}$ is the set of all predicates and objects that were found for the entity that is being ranked (3a).

$$Rank(entity) = \frac{Triples_{seeds} \cap Triples_{entity}}{Triples_{seeds} \cup Triples_{entity}} \quad (1)$$

The rationale behind this similarity approach for ranking is that ranking should be relative to the entities from the *SeedSet*. For instance, if we used only comedy actors as seeds, other comedy actors should be ranked higher than other actors. The amount of predicates and objects that comedy actors have in common is usually higher than that of musical actors and comedy actors.

4 Evaluation

In this section we evaluate SWELC[4]. In all our experiments, we used the API of Sindice.com as the gateway to the Semantic Web.

4.1 ELC on 17 concepts

Fig. 4 shows the precision of SWELC on a set of 17 different concepts. For each concept, we evaluated SWELC with and without the automatic concept detection step. For each concept we manually assigned over three URIs on average from different ontologies. For the automatic detection step we used four randomly picked seeds and tried to detect a concept with 40 attemps maximum. If nothing was found, we reduced the seed set size to three, tried 40 more times, and so on. Surprisingly, the average precision of SWELC was only about 3% lower when automatically detecting the concept compared to the manually assigned ones. Indeed, for several concepts, automatic detection found better matching concept URIs than we selected manually (e.g for `Sports Team`).

4.2 Comparison to Related Work

In this section, we compare the SWELC to the two state-of-the-art systems Boo!Wa! and Google Sets. We randomly chose five concepts from different domains which are namely `Lake` (location), `Movie` (product), `Politician` (person), `Newspaper` (organization), and `Airport` (construction). As the ELC task is mainly user oriented, we use the precision@k measure for evaluation, which correlates well with the user's satisfaction. We use $k = 10$ largeley because Google Sets seldom returns more results even in the large result list. For each concept, we evaluated the result lists three times, each time with three randomly selected seed entities. If a system did not produce an answer, we removed an entity and tried again. We also removed the seed entities from the result list since they would be of no help to a user. Table 1 shows the comparison of the precision@10 values for the three systems across the five concepts[5]. Interestingly, the

[4] A demo is available online within the WebKnox project: `http://webknox.com/wi`

[5] Online evaluation on Google Sets and Boo!Wa! was performed in August 2011.

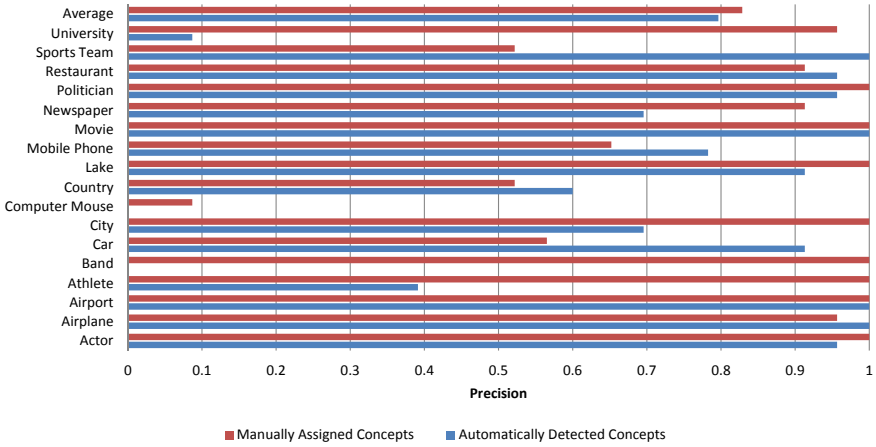

Fig. 4. SWELC Precision across 17 Concepts

Boo!Wa! system was unable to answer any of the 15 queries with a result list. Google Sets performs well on the popular concept `Movie` but is overall rather concept dependent. SWELC performs 5% better than Google Sets overall in our evaluation. It is worth mentioning that SWELC returned many more results (up to 6,000 movies for example) than Google Sets. The problem with SWELC is the concept detection. If the seeds were selected properly and the concept was detected correctly, almost all the answers were correct. If a wrong concept was detected, almost all results were wrong. Improving the concept detection will therefore yield much higher precision scores.

Concept	Lake	Movie	Politician	Newspaper	Airport	Overall
Google Sets	0.07	**1.00**	0.27	0.87	0.70	0.58
SWELC	**0.50**	0.33	**0.33**	**1.00**	**1.00**	**0.63**

Table 1. Precision@10 scores across three different ELC systems for five concepts

5 Conclusion

We have presented an ELC algorithm that utilizes the vast amount of information stored in the Semantic Web. We were able to show that the system reaches about 79% precision across 17 concepts when instantiated with one to four seeds and outperforms the state-of-the-art systems Boo!Wa! and Google Sets by at least 5% in precision@10. The system can still be improved, especially since the choice of seeds has a strong influence on the generated lists.

References

[1] Google Sets. System and methods for automatically creating lists. US Patent: US7350187, March 2008.

[2] K. Balog, E. Meij, and M. de Rijke. Entity Search: Building Bridges between Two Worlds. In *SemSearch2010*, Raleigh, NC, 2010. ACM.

[3] B. Dalvi, J. Callan, and W. Cohen. Entity List Completion Using Set Expansion Techniques. In *Proc. of the Nineteenth Text REtrieval Conference*, 2011.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Proc. SAMT 2009*, pages 182–187, Berlin/Heidelberg, 2009. Springer.

[6] J. Lehmann, J. Schüppel, and S. Auer. Discovering Unknown Connections - the DBpedia Relationship Finder. In *Proc. of the CSSW 2007*, volume 113 of *LNI*, pages 99–110. GI, 2007.

[7] R. C. Wang and W. W. Cohen. Language-Independent Set Expansion of Named Entities Using the Web. In *The 2007 IEEE International Conference on Data Mining*, pages 342–350, 2007.

A Meeting the Minimal Requirements

1. The application is end-user oriented in the same way Google Sets and Boo!Wa! are. Users searching for related entities can use the ELC service, for example, someone trying to generate lists of newspapers can type three newspapers and gets a list of more related newspapers. This scenario is probably not for the "general Web user" but rather domain experts or users in need for lists.
2. Information sources are under diverse ownership and are semantically heterogeneous (we query sindice.com which again has many different sources in its index). There are millions of tuples index by sindice.com which makes it more than a toy example.
3. Meaning is represented using RDF. We need this meaning to know which classes a certain entity is of.

B Meeting the Desired Requirements

1. A Web interface to the application is available at `http://webknox.com/wi` (Entity List Completion).
2. The application scales as it is based on sindice.com. The time for generating the entity lists does not scale well, however. The user wants to find something fast and the more tuples we search through the long it takes.
3. As discussed in the related work section, we apply Semantic Web technology to a well-studied task of Entity List Completion. Semantic Web technologies have not been researched in depth for this task and in our evaluation we show that it is beneficial to use the Semantic Web.
4. We also rank the entities using context (which are all triples of the entities).