LamAPI: a Comprehensive Tool for String-based Entity Retrieval with Type-base Filters

Roberto Avogadro[1], Marco Cremaschi[1], Fabio D'Adda[1], Flavio De Paoli[1] and Matteo Palmonari[1]

[1]*Università degli Studi di Milano - Bicocca, 20126 Milano, Italy*

Abstract

When information available in unstructured or semi-structured formats, *e.g.*, tables or texts, comes in, finding links between strings appearing in these sources and the entities they refer to in some background Knowledge Graphs (KGs) is a key step to integrate, enrich and extend the data and/or KGs. This *Entity Linking* task is usually decomposed into *Entity Retrieval* and *Entity Disambiguation* because of the large entity search space. This paper presents an Entity Retrieval service (LamAPI) and discusses the impact of different retrieval configurations, *i.e.*, query and filtering strategies, on the retrieval of entities. The approach is to augment the search activity with extra information, like types, associated with the strings in the original datasets. The results have been empirically validated against public datasets.

Keywords

Entity Linking, Entity Retrieval, Entity Disambiguation, Knowledge Graph

1. Introduction

A key advantage of developing Knowledge Graphs (KGs) consists in effectively supporting the integration of data coming with different formats and structures [1]. In semantic data integration, KGs provide identifiers and descriptions of entities, thus supporting data integration like tables or texts. The table-to-KG matching problem, also referred to as semantic table interpretation, has recently collected much attention in the research community [2, 3, 4] and is a key step to enrich data [1, 5] and construct and extend KGs from semi-structured data [6, 7]. When information available in unstructured or semi-structured formats, *e.g.*, tables or texts, comes in, *finding links* between strings (or mentions) appearing in these sources and the entities they refer to in some background KGs is a key step to integrate, enrich and extend the data and/or KGs. We name this task Entity Linking (EL), which comes in different flavours depending on the considered data formats but with some shared features.

For example, because of the ample entity search space, most of the approaches to EL include a first step where candidate entities for the input string are collected, *i.e.*, Entity Retrieval (ER) [8], and a second step where the string is disambiguated by eventually selecting one or none of the candidate entities, *i.e.*, Entity Disambiguation (ED) [9]. In most approaches, ER returns

OM-2022: The 17th International Workshop on Ontology Matching, Hangzhou, China

roberto.avogadro@unimib.it (R. Avogadro); roberto.avogadro@unimib.it (M. Cremaschi); f.dadda4@campus.unimib.it (F. D'Adda); flavio.depaoli@unimib.it (F. D. Paoli); matteo.palmonari@unimib.it (M. Palmonari)

CEUR Workshop Proceedings (CEUR-WS.org)

a ranked list of candidates, while disambiguation consists of re-ranking the input list. Entity Disambiguation is at the heart of EL, with different approaches that leverage different kinds of evidence depending on the format and features of the input text [10]. However, the ER step is also significant considering that its results define an upper bound for the performance of the end-to-end linking: if an entity is not among the set of candidates, it cannot be selected as the target for the link. Also, while it is, in principle, possible to scroll the list of candidates at arbitrary levels of depth, maintaining acceptable efficiency levels requires cutting off the results of ER at a reasonable depth.

Approaches to entity searches can either resort to existing lookup APIs, *e.g.*, DBpedia SPARQL Query Editor[1], DBpedia Spotlight[2] or Wikidata Query Service[3], or use recent approaches to dense ER [11], when entities are searched in a pre-trained dense space, an approach becoming especially popular in EL for textual data. The APIs reported above provide access to the SPARQL endpoint because the elements are stored in Resource Description Framework (RDF) format. Such endpoints are usually offered on local dumps of the original KGs to avoid network latency and increase efficiency. For instance, DBpedia can be accessed by OpenLink Virtuoso, a row-wise transaction-oriented RDBMS with a SPARQL query engine[4], and Wikidata by Blazegraph[5], a high-performance graph database providing RDF/SPARQL-based APIs. An issue faced with these solutions is the time required for downloading and setting up the datasets: Wikidata 2019 requires some days to set up[6] since the full dump is about 1.1TB (uncompressed). Moreover, writing SPARQL queries may be an issue since specific knowledge of the searched Knowledge Graph (KG) is required, besides the knowledge of the required syntax. Some limitations related to the use of these endpoints are:

- the SPARQL endpoint response time is directly proportional to the size of the returned data. As a consequence, sometimes it is not even possible to get a result because the endpoint fails for timeout;
- the number of requests per second may be severely limited for online endpoints (to ensure feasibility) or computationally too expensive for local endpoints (a reasonable configuration requires at least 64GB of RAM with tons of CPU cycles);
- there are some intrinsic limits in the SPARQL language expressiveness (*i.e.*, full-text search capability, which is required for matching table mentions, can be obtained only with extremely slow "contains" or "regex" queries[7]).

Regarding the approaches to dense ER, some limitations can be mentioned [12, 13]:

- the results are strictly related to the type of representation used. Consequently, careful and tedious feature engineering is required when designing these systems;

[1] dbpedia.org/sparql

[2] www.dbpedia-spotlight.org

[3] query.wikidata.org

[4] virtuoso.openlinksw.com

[5] blazegraph.com

[6] addshore.com/2019/10/your-own-wikidata-query-service-with-no-limits-part-1/

[7] docs.openlinksw.com/virtuoso/rdfsparqlrulefulltext/

- generalising the trained entity linking model to other KGs or domains is challenging due to the strong dependence on the specific KG and domain knowledge in the process of designing features;
- these systems depend excessively on external data, and the effectiveness of the algorithms is directly affected by the quality of the training data, and their utility is indispensable restricted.

Information Retrieval (IR) approaches based on search engines still provide valuable solutions to support entity search, mainly because they do not require training, work with any KG, and easily adapt to changes in the reference KG. Although IR-based entity search has been used extensively, especially in table-to-kg matching [14, 15, 16, 17], their use has been frequently left to custom optimisations and not adequately discussed or documented in scientific papers. As a result, researchers willing to apply such solutions must develop from scratch, including data indexing techniques, query formulation and service set-up.

In this paper, we aim to present: i) LamAPI, a comprehensive tool for IR-based ER, augmented with type-based filtering features, and ii) a study of the impact of different retrieval configurations, *i.e.*, query and filtering strategies, on the retrieval of entities. The tool supports string-based retrieval but also hard and soft filters [18] based on an input entity type (*i.e.*, `rdf:type` for DBpedia and `Property:P31` for Wikidata). Hard type filters remove non-matching results, while soft type filters promote or demote results when an exact match is not feasible. These filters are useful to support either EL in texts (*e.g.*, by exploiting entity types returned by a classifier [19, 20]), or in tables (*e.g.*, by exploiting a known column type (`rdf:type`) to filter out irrelevant entities). While the approach is general, the tool provides support EL for semi-structured data. In our study, we, therefore, focus on evaluating different retrieval strategies with/without filters on EL in the table-to-KG matching settings, considering two different large KG such as WikiData and DBpedia. Finally, the tool also contains mappings among the latter two KGs and Wikipedia, thus supporting cross-KG bridges. The tool and all the resources used for the experiments are released following the FAIR Guiding Principles[8]. LamAPI is released under the Apache 2.0 licence.

The rest of this article is organised as follows. Section 2 will be presented a brief analysis of state of the art on String-based entity retrieval techniques. We will describe the services offered by LamAPI in Section 3. Section 4 introduces the Gold Standards, the configuration parameters and finally discusses the evaluation results. Finally, we conclude this paper and discuss the future direction in Section 6.

2. String-based entity retrieval

Given a KG containing a set of entities E and a collection of named-entity mentions M, the goal of EL is to map each entity mention $m \in M$ to its corresponding entity $e \in E$ in the KG. As described above, a typical EL service consists of the following modules [10]:

1. Entity Retrieval (ER). In this module, for each entity mention $m \in M$, irrelevant entities in the KG are filtered out to return a set E_m of candidate entities: entities that mention

[8] www.nature.com/articles/sdata201618

m may refer to. To achieve this goal, state-of-the-art techniques have been used, such as name dictionary-based techniques, surface form expansion from the local document, and methods based on search engines.

2. Entity Disambiguation (ED). In this module, the entities in the set E_m are more accurately ranked to select the correct entity among the candidate ones. In practice, this is a re-ranking activity that considers other information (*e.g.*, contextual information) besides the simple textual mention m used in the ER module.

According to the experiments conducted [21], the role of the ER module is critical since it should ensure the presence of the correct entity in the returned set to let the ED module to find it. Hence, the main contribution of this work is to discuss retrieval configurations, *i.e.*, query and filtering strategies, for retrieving entities.

Name dictionary-based techniques are the main approaches to ER; such techniques leverage different combinations of features (*e.g.*, labels, alias, Wikipedia hyperlinks) to build an offline dictionary D of links between string names and mapping entities to be used to generate the set of candidate entities. The most straightforward approach considers exact matching between the textual mention m and string names inside D. Partial matching (*e.g.*, fuzzy and/or n-grams search) can also be considered.

Besides pure string matching, type constraints (using types/classes of the KG) associated with string mentions can be exploited to filter candidate entities. In such a case, the dictionary needs to be augmented with types associated with linked entities to enable hard or soft filtering. Listing 1 and 2 report an example of how type constraints can influence the result of the candidate entity retrieval for “manchester” textual mention. The former shows the result without constraint: cities like Manchester situated in England or Parish in Jamaica are reported (note the similarity score equal to 1.00). The latter shows the result when type constraints are applied: types like “SoccerClub” and “SportsClub” allows for the promotion of soccer clubs such as “Manchester United F.C.”, which is now ranked first (similarity score 0.83).

Similar approaches have been proposed in this domain, such as the MTab [14] entity search, where keyword search, fuzzy search and aggregation search are provided. Another relevant approach is EPGEL [15], where the candidate entity generation uses both a keyword and a fuzzy search. This approach also uses BERT [22] to create a profile for each entity to improve the search results. The LinkingPark [16] method proposes a weighted combination of keywords, trigrams and fuzzy search to maximise recall during the candidate generation process. In addition, this approach involves verifying the presence of typos before generating candidates. Concerning the other work, LamAPI provides a n-grams search and the possibility to include type constraints in the candidate search to apply type/concept filtering in the ER. Furthermore, LamAPI provides several services to help researchers in tasks like EL.

Listing 1: DBpedia lookup without type constraints.

```
{
"id": Manchester
"label": Manchester
"type": City Settlement ...
"ed_score": 1
},
{
"id": Manchester_Parish
"label": Manchester
"type": Settlement PopulatedPlace
"ed_score": 1
}
```

Listing 2: DBpedia lookup with type constraints.

```
{
"id": Manchester_United_F.C.
"label": Manchester U
"type": SoccerClub SportsClub ...
"ed_score": 0.833
},
{
"id": Manchester_City_F.C.
"label": Manchester C
"type": SoccerClub SportsClub ...
"ed_score": 0.833
}
```

3. LamAPI

The current version of LamAPI integrates DBpedia (v. 2016-10 and v. 2022.03.01) and Wikidata (v. 20220708), the most popular free KGs. However, any KG, even private and domain-specific could be integrated. The only constraint is to support indexing, as described in Section 3.1.

3.1. Knowledge Graphs indexing

DBpedia, Wikidata and the like are very large KGs that require an enormous amount of time and resources to perform ER, so we created a more compact representation of these data suitable for ER tasks. For each KG, we downloaded a dump (*e.g.*, 'latest-all.json.bz2' for DBpedia that sizes 71 GB with multiple files), created a local copy in a single file by extracting and storing all triples (*e.g.*, 96580491 entities for DBpedia). We then created an index with ElasticSearch[9], an engine that can search and analyse huge volumes of data in near real-time. These customised local copies of the KGs are then used to create endpoints to provide EL retrieval services. The advantage is that these services can work on partitions of the original KGs to improve performance by saving time and using fewer resources.

3.2. LamAPI services

Among the services provided by LamAPI to search and retrieve information in a KG, we discuss the Lookup and Type-similarity, which are the relevant services for entity retrieval.
Lookup: given a string input, it retrieves a set of candidate entities from the reference KG. The request can be qualified by setting some attributes:

limit: an integer value specifies the number of entities to retrieve. The default value is 100, and it has been empirically demonstrated how this limit allows a good level of coverage.

kg: specifies which KG and version to use. The default is dbpedia_2022_03_01, and other possible values are dbpedia_2022_03_01, dbpedia_2016_10 or wikidata_latest.

fuzzy: a boolean value. When true, it matches tokens inside a string with an edit distance (Levenshtein distance) less than or equal to 2. This gives a greater tolerance for spelling errors. When false, the fuzzy operator is not applied to the input.

[9] www.elastic.co

ngrams: a boolean value. When true, it permits to search n-grams. After many empirical experiments, we set ‘n’ of n-grams equal to 3. A lower value can bring some bias in the search, while a higher value could not be very effective in terms of spelling errors. “albert einstein” using n-grams equal to 3 is split in [’alb’, ’lbe’, ’ber’, ’ert’, ...]. When false is not applied on input.

types: this parameter allows the specification of a list of types (*e.g.*, `rdf:type` for DBpedia and `Property:P31` for Wikidata) associated with the input string to filter the retrieved entities. This attribute plays a key role in re-ranking the candidates, allowing a more accurate search based on input types.

The following example discusses the difference between a SPARQL query and the LamAPI Lookup service. Listings 3 and 4 show a search using the mention “Albert Einstein”. The evidence is that LamAPI syntax is simpler than the one in SPARQL. The Lookup service allows for managing the presence of misspelled mentions. Finally, another advantage over SPARQL is the ranking of candidates.

Listing 3: Search example using a SPARQL query.

```
select distinct ?s where {
?s ?p ?o .
FILTER( ?p IN (rdfs:label)).
?o bif:contains "Albert Einstein".
}
order by strlen(str(?s))
LIMIT 100
```

Listing 4: Example of LamAPI Lookup service.

```
/lookup/entity-retrieval?
name="Albert Einstein"&
limit=100&
token=insideslab-lamapi-2022&
kg=dbpedia_2022_03_01&
fuzzy=False&
ngrams=False
```

Examples of results with the input string “Albert Einstein” returned by LamAPI are shown in Listing 5 and Listing 6, referred to Wikidata and DBpedia, respectively. Each candidate entity is described, in W3C specification[10] format, by the unique identifier *id* in the chosen KG, a string label *name* reporting the official name of the entity, a set of types associated with the entity, each one described by its unique identifier *id* and the corresponding string label *name*, and an optional description of the entity (*e.g.*, DBpedia does not provide descriptions, while Wikidata does). Moreover, a score with the edit distance measure (Levenshtein distance) between the input textual mention and the entity label is reported.

Listing 5: Lookup: returned data from Wikidata.

```
{
"id": Q937
"label": Albert Einstein
"description": German-born ...
"type": Q19350898 Q16389557 ... Q5
"score": 1.0
},
{
"id": Q356303
"label": Albert Einstein
"description": American actor ...
"type": Q33999 Q2526255 ... Q5
"score": 1.0
}
```

Listing 6: Lookup: returned data from DBpedia.

```
{
"id": Albert_Einstein
"label": Albert Einstein
"description": ...
"type": Scientist Animal ...
"score": 1.0
},
{
"id": Albert_Einstein_ATV
"label": Albert Einstein ATV
"description": ...
"type": SpaceMission Event ...
"score": 0.789
}
```

The score provides a candidate ranking that can be used by the Entity Disambiguation (ED)

[10] reconciliation-api.github.io/specs/latest/

module for a straightforward selection of the actual link. The intuition is that when there is one candidate with a score above a certain threshold, it can be selected, whereas when multiple candidates share the same score, or the highest score is very low, further investigation is needed to find the correct entity.

The types present in the response can be used to iterate a Lookup request to filter the results and obtain more accurate candidate lists, as shown in Listing 1 and 2. Thanks to the Type-similarity service described below, it is possible to identify similar types of a given type (`rdf:type`), which allows for relaxing the constraints in case of uncertainty on which type to use as a filter.

Type-similarity: given the unique *id* of a type as input, it retrieves the top k most similar types by calculating a ranking based on cosine similarity.

Examples of returned results with the input string Philosopher and Scientist are shown in Listing 7 and Listing 8, referred to Wikidata and DBpedia, respectively.

Listing 7: Type-similarity: returned data from Wikidata.

```
Q4964182(philosopher)
{
  "type": Q4964182(philosopher)
  "cosine_similarity": 1.0
},
{
  "type": Q2306091(sociologist)
  "cosine_similarity": 0.865
}
Q901(scientist)
{
  "type": Q901(scientist)
  "cosine_similarity": 1.0
},
{
  "type": Q19350898(theoretical...)
  "cosine_similarity": 0.912
}
...
```

Listing 8: Type-similarity: returned data from DBpedia.

```
Philosopher
{
  "type": Philosopher
  "cosine_similarity": 1.0
},
{
  "type": Economist
  "cosine_similarity": 0.684
}
Scientist
{
  "type": Scientist
  "cosine_similarity": 1.0
},
{
  "type": Medician
  "cosine_similarity": 0.723
},
...
```

4. Validation

In this Section, different retrieval configurations, *i.e.*, query and filtering strategies, are illustrated and validated.

The dataset used for validation is 2T_2020 [23]: 2T comprises 180 tables with around 70.000 unique cells. It is characterised by cells with intentionally orthographic errors, so the ER with misspelled words can be tested. The dataset is available for both Wikidata and DBpedia KG; it is possible to compare the results for both KG using the same tables.

The validation process starts with a set of mentions M, and a number k of candidates associated with each mention. The *Lookup* service returns a set of candidates E_m that includes all the candidates found. The returned set is then checked against the 2T to verify which among the correct entities are present and in what position in the ranked results in E_m. We compute the coverage following this formula:

$$coverage = \frac{\#\ candidates\ found}{\#\ total\ candidates\ to\ find} \tag{1}$$

Where # represents "number of".
In Table 1 the various coverage values are presented for lookup based on label matching on a mention by enabling *fuzzy* and *n-grams* searches. The experiments were conducted using 20 parallel processes on a server with 40 CPU(s) Intel Xeon Silver 4114 CPU @ 2.20GHz and 40GB RAM.

Table 2 and 3 show the coverage using the constraint on types. To select and expand types, four methods were applied.

1. **Type**: This method considers only the type or set of types (seed types) indicated in the call to the Lookup service, and it does not carry out any expansion of types.
2. **Type Co-occurrency**: For the seed types, it extracts additional types based on the co-occurency of types in the KG. The co-occurency score represents the number of times each type co-occurs with another type in a KG at entity level.
3. **Type Cosine Similarity**: The seed types are extended by the cosine similarity of RDF2Vec[11].
4. **Soft Inference**: The seed types are extended using a Feed Forward Neural Network that takes as input the RDF2Vec vector of an entity, linked to a mention and predicts the possible types for the input entity [18].

In Table 2, it is possible to notice that the first method achieves a higher coverage. The best result is obtained by adding two types. Co-Occurrencies and Type Cosine Similarity are both 'idempotent' methods. The Soft Inference technique uses the entities obtained by a prelinking. Not all entity vectors are available, so we cannot always extend the set of types. In Table 3 we report the results for Wikidata. Also, in this case, the best results here are achieved using the first method. The achieved coverage is highest because this KG has a comprehensive hierarchy with more detailed types.

Even if lower, the coverage values obtained with type expansion methods are promising. We must consider how the exact type to use as a filter is often not known a priori in a real scenario. For example, to select a type, a user should know the profile of a KG and how it is used to describe entities. Thanks to the methods described above, the search results will contain entities belonging to other types but still related to the input.

5. The LamAPI retrieval service

LamAPI is implemented in Python using ElasticSearch and MongoDB. A demonstration setting is publicly available[12] through a Swagger documentation page for testing purposes (Figures 1 and 2). LamApi Repository[13] is publicly available, so the code can be downloaded and customised if needed.

[11] rdf2vec.org
[12] lamapi.ml
[13] bitbucket.org/discounimib/lamapi

Table 1
Coverage results and response times for different searches in Wikidata and DBpedia v. 2022.03.01.

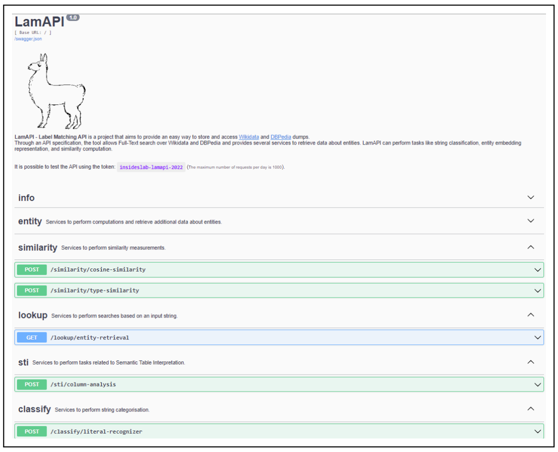
Methods	DBpedia		Wikidata	
	Coverage	Time	Coverage	Time
N-gram	0,842	228 s	0,787	649 s
Fuzzy	0,806	226 s	0,805	766 s
Token	0,561	227 s	0,530	230 s
N-gram + Fuzzy	0,891	267 s	0,926	1649 s
N-gram + Token	0,883	229 s	0,891	807 s
Fuzzy + Token	0,812	226 s	0,825	773 s
N-gram + Fuzzy + Token	**0,895**	270 s	**0,929**	1577 s

Table 2
Coverage results for 2T DBpedia.

Methods	w/o type	1 type	2 types	3	4	5	6	7	8	9	10
Type	0,892	0,904	**0,905**	0,904	0,889	0,884	0,879	0,872	0,870	0,867	0,848
Type Co-occurency	0,892	0,886	**0,896**	0,886	0,856	0,884	0,830	0,834	0,834	0,833	0,823
Type Cosine Similarity	0,892	**0,892**	0,886	0,889	0,885	0,881	0,873	0,869	0,825	0,825	0,830
Soft Inference	**0,892**	0,885	0,872	0,884	0,882	0,879	0,885	0,886	0,878	0,874	0,869

Table 3
Coverage results for 2T Wikidata.

Methods	w/o type	1 types	2 types	3	4	5	6	7	8	9	10
Type	0,929	0,941	0,939	0,946	0,946	**0,947**	**0,947**	0,945	0,945	0,943	0,944
Type Co-occurency	0,929	0,854	0,808	0,796	0,793	0,795	0,797	0,797	0,795	0,796	0,795
Type Cosine Similarity	0,929	0,853	0,853	0,852	0,851	0,850	0,849	0,849	0,848	0,847	0,845


LamAPI 1.0
[Base URL: /]
/swagger.json

LamAPI - Label Matching API is a project that aims to provide an easy way to store and access Wikidata and DBPedia dumps.
Through an API specification, the tool allows Full-Text search over Wikidata and DBPedia and provides several services to retrieve data about entities. LamAPI can perform tasks like string classification, entity embedding representation, and similarity computation.

It is possible to test the API using the token: `insideslab-lamapi-2022` (The maximum number of requests per day is 1000).

info

entity Services to perform computations and retrieve additional data about entities.

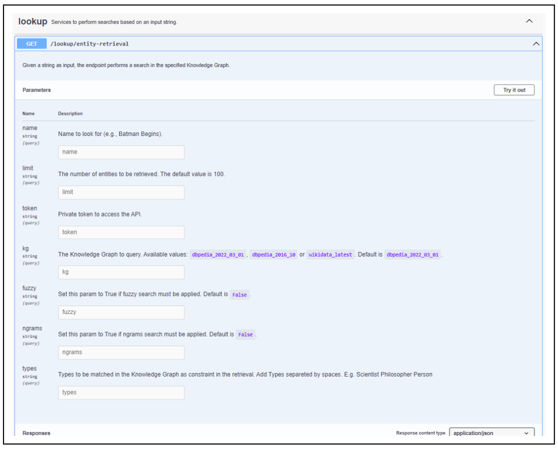
similarity Services to perform similarity measurements.
POST /similarity/cosine-similarity
POST /similarity/type-similarity

lookup Services to perform searches based on an input string.
GET /lookup/entity-retrieval

sti Services to perform tasks related to Semantic Table Interpretation.
POST /sti/column-analysis

classify Services to perform string categorisation.
POST /classify/literal-recognizer

Figure 1: LamAPI documentation page.


lookup Services to perform searches based on an input string.
GET /lookup/entity-retrieval

Given a string as input, the endpoint performs a search in the specified Knowledge Graph.

Parameters

Name	Description
name string *(query)*	Name to look for (e.g., Batman Begins). name
limit string *(query)*	The number of entities to be retrieved. The default value is 100. limit
token string *(query)*	Private token to access the API. token
kg string *(query)*	The Knowledge Graph to query. Available values: `dbpedia_2022_03_01` , `dbpedia_2016_10` or `wikidata_latest` . Default is `dbpedia_2022_03_01` . kg
fuzzy string *(query)*	Set this param to True if fuzzy search must be applied. Default is `False` . fuzzy
ngrams string *(query)*	Set this param to True if ngrams search must be applied. Default is `False` . ngrams
types string *(query)*	Types to be matched in the Knowledge Graph as constraint in the retrieval. Add Types separeted by spaces. E.g. Scientist Philosopher Person types

Responses
Response content type application/json

Figure 2: LamAPI Lookup service.

For completeness, the list with the relative description of the LamAPI services is provided. **Types**: given the unique *id* of an entity as input, it retrieves all the types of which the entity is an instance. The service relies on vector similarity measures among the types in KG to compute the answer. For DBpedia entities, the service returns both direct types, transitive types, and

Wikidata types of the related entity, while for Wikidata, it returns only the list of concepts/types for the input entity.
Literals: given the unique *id* of an entity as input, it retrieves all relationships (predicates) and literal values (objects) associated with that entity.
Predicates: given the unique *id* of two entities as input, it retrieves all the relationships (predicates) between them.
Objects: given the unique *id* of an entity as input, it retrieves all related objects and predicates.
Type-predicates: given the unique *id* of two types as input, it retrieves all predicates that relate entities of input types with a frequency score associated with each predicate.
Labels: given the unique *id* of an entity as input, it retrieves all the related labels and aliases (`rdfs:label`).
WikiPageWikiLinks: given the unique *id* of an entity as input, it retrieves links from a WikiPage to other Wikipages.
Same-as: given the unique *id* of an entity as input, it returns the corresponding entity for both Wikidata and DBpedia (`schema:sameAs`).
Wikipedia-mapping: given the unique *id* or *curid* of a Wikipedia entity, it returns the corresponding entity for Wikidata and DBpedia.
Literal-recogniser: Given an array as input composed of a set of strings, the endpoint returns the types of each literal by applying a set of regex rules. The list of literals recognised is *dates* (*e.g.*, 1997-08-26, 1997.08.26, 1997/08/26), *numbers* (*e.g.*, 2.797.800.564, 25 thousand, +/- 34657, 2 km), *url*, *email* and *time* (*e.g.*, 12.30pm, 12pm).

6. Conclusions

Effective Entity Retrieval services are crucial to effectively support the task of Entity Linking for unstructured and semi-structured datasets. In this paper, we discussed how different strategies can be beneficial to reduce the search space and therefore deliver more accurate results saving time, computing power and storage capability. The results have been empirically validated against public datasets of tabular data. Preliminary experiments with textual data are encouraging. We plan to complete such validation activities and further develop LamAPI to provide full support for any format of input datasets. In addition, other search and filtering strategies will be implemented and tested to provide users with a complete set of alternatives, along with information on when and how each can be usefully adopted. The tool could be extended for supporting also other tasks in the natural language process like entity linking on free text.

References

[1] V. Cutrona, M. Ciavotta, F. D. Paoli, M. Palmonari, ASIA: a tool for assisted semantic interpretation and annotation of tabular data, in: Proceedings of the ISWC 2019 Satellite Tracks, volume 2456 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 209–212.

[2] E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems, in: The Semantic Web, Springer International Publishing, Cham, 2020, pp. 514–530.

[3] E. Jimenez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, V. Cutrona, Results of semtab 2020, CEUR Workshop Proceedings 2775 (2020) 1–8.

[4] V. Cutrona, J. Chen, V. Efthymiou, O. Hassanzadeh, E. Jimenez-Ruiz, J. Sequeda, K. Srinivas, N. Abdelmageed, M. Hulsebos, D. Oliveira, C. Pesquita, Results of semtab 2021, in: 20th International Semantic Web Conference, volume 3103, CEUR Workshop Proceedings, 2022, pp. 1–12.

[5] M. Palmonari, M. Ciavotta, F. De Paoli, A. Košmerlj, N. Nikolov, Ew-shopp project: Supporting event and weather-based data analytics and marketing along the shopper journey, in: Advances in Service-Oriented and Cloud Computing, Springer International Publishing, Cham, 2020, pp. 187–191.

[6] G. Weikum, X. L. Dong, S. Razniewski, F. M. Suchanek, Machine knowledge: Creation and curation of comprehensive knowledge bases, Found. Trends Databases 10 (2021) 108–490.

[7] M. Kejriwal, C. A. Knoblock, P. Szekely, Knowledge graphs: Fundamentals, techniques, and applications, MIT Press, 2021.

[8] L. Ratinov, D. Roth, Design challenges and misconceptions in named entity recognition, in: Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009), Association for Computational Linguistics, Boulder, Colorado, 2009, pp. 147–155.

[9] D. Rao, P. McNamee, M. Dredze, Entity Linking: Finding Extracted Entities in a Knowledge Base, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 93–115.

[10] W. Shen, J. Wang, J. Han, Entity linking with a knowledge base: Issues, techniques, and solutions, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 443–460.

[11] L. Wu, F. Petroni, M. Josifoski, S. Riedel, L. Zettlemoyer, Zero-shot entity linking with dense entity retrieval, in: EMNLP, 2020.

[12] W. Shen, Y. Li, Y. Liu, J. Han, J. Wang, X. Yuan, Entity linking meets deep learning: Techniques and solutions, IEEE Transactions on Knowledge and Data Engineering (2021) 1–1.

[13] X. Li, Z. Li, Z. Zhang, N. Liu, H. Yuan, W. Zhang, Z. Liu, J. Wang, Effective few-shot named entity linking by meta-learning, 2022.

[14] P. Nguyen, I. Yamada, N. Kertkeidkachorn, R. Ichise, H. Takeda, Semtab 2021: Tabular data annotation with mtab tool., in: SemTab@ ISWC, 2021, pp. 92–101.

[15] T. M. Lai, H. Ji, C. Zhai, Improving candidate retrieval with entity profile generation for wikidata entity linking, arXiv preprint arXiv:2202.13404 (2022).

[16] S. Chen, A. Karaoglu, C. Negreanu, T. Ma, J.-G. Yao, J. Williams, F. Jiang, A. Gordon, C.-Y. Lin, Linkingpark: An automatic semantic table interpretation system, Journal of Web Semantics 74 (2022) 100733.

[17] C. Sarthou-Camy, G. Jourdain, Y. Chabot, P. Monnin, F. Deuzé, V.-P. Huynh, J. Liu, T. Labbé, R. Troncy, Dagobah ui: A new hope for semantic table interpretation, in: European Semantic Web Conference, Springer, 2022, pp. 107–111.

[18] V. Cutrona, G. Puleri, F. Bianchi, M. Palmonari, Nest: Neural soft type constraints to improve entity linking in tables., in: SEMANTiCS, 2021, pp. 29–43.

[19] J. Raiman, O. Raiman, Deeptype: Multilingual entity linking by neural type system evolution, Proceedings of the AAAI Conference on Artificial Intelligence 32 (2018).

[20] Y. Onoe, G. Durrett, Fine-grained entity typing for domain independent entity linking,

Proceedings of the AAAI Conference on Artificial Intelligence 34 (2020) 8576–8583.

[21] B. Hachey, W. Radford, J. Nothman, M. Honnibal, J. R. Curran, Evaluating entity linking with wikipedia, Artificial Intelligence 194 (2013) 130–150. Artificial Intelligence, Wikipedia and Semi-Structured Resources.

[22] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).

[23] V. Cutrona, F. Bianchi, E. Jiménez-Ruiz, M. Palmonari, Tough tables: Carefully evaluating entity linking for tabular data, in: The Semantic Web – ISWC 2020, Springer International Publishing, Cham, 2020, pp. 328–343.