LOTUS:
A Search Engine for the Web of Data

Filip Ilievski Wouter Beek Marieke van Erp
Laurens Rietveld Stefan Schlobach

It is surprisingly difficult to find things on today's Web of Data. You need an IRI to start traversing the interconnected knowledge graph. But how do you find such a resource-denoting IRI? On the Web of Documents you can use a search engine. Unfortunately, there are currently no search engines for the Web of Data. *This is where LOTUS kicks in.*

LOTUS, or Linked Open Text UnleaShed, is a text-based entry point to a large subset of today's Linked Open Data (LOD) Cloud. It allows 2 billion unique Linked Data IRIs to be searched based on an index that covers 4.3 billion literals. Advanced users can further customize the text-based retrieval in LOTUS to fit better a wealth of use cases. Try it out over at `http://lodsearch.org`.

1 Introduction

It would be very difficult to use today's Web of Documents without a good search engine. Information is stored at URLs, but new URLs arise all the time. URLs are hard to memorize and difficult to keep up-to-date. The same is true for the Web of Data. Linked Data best practices tell us that once we have an IRI we are able to find more information (the 'follow your nose'-principle). Unfortunately, we cannot easily obtain those first IRIs that kick-start our use of Linked Data.

LOTUS, or Linked Open Text UnleaShed, is a text-based entry point to a large subset of today's Linked Open Data (LOD) Cloud. LOTUS allows over 2 billion unique Linked Data IRIs to be searched by matching a user-supplied free text search string to over 4.3 billion RDF literals.

LOTUS can be used by machines and humans alike. Machines use a Web API (Section 5) that follows open standards like SPARQL and LDF to get the job done. Humans use a Web UI that is similar to search engines for the Web of Documents (Figure 1). The Web UI does not bother the user with unnecessary detail.

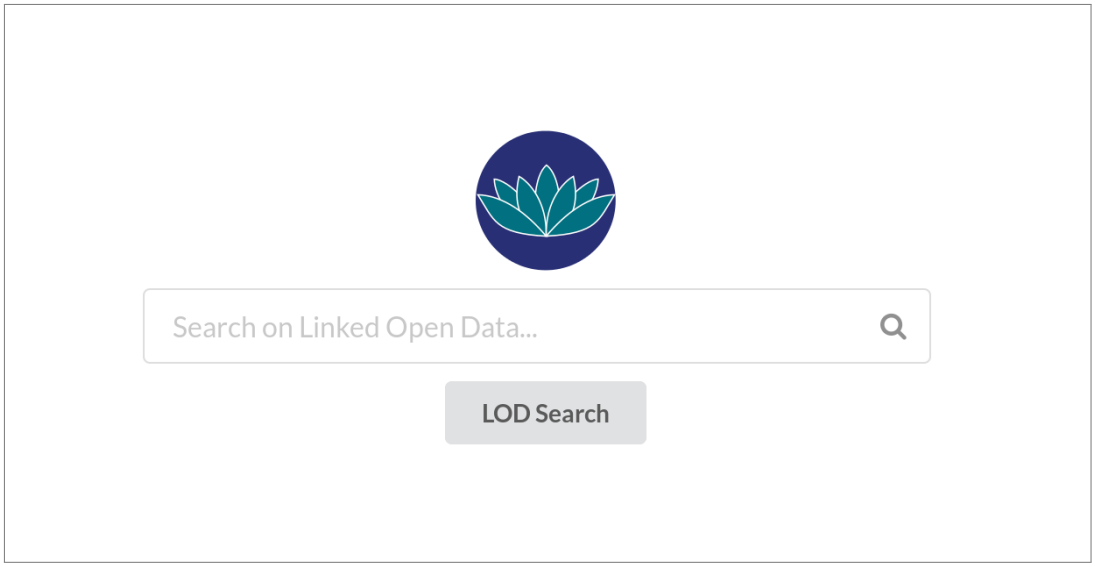

Figure 1: The LOTUS front-end for human users. Free text can be entered as with search engines for the Web of Documents.

2 Approach

When a user enters a search string, LOTUS *matches* that search string against a very large collection (i.e., over four billion) of literals that contain natural language text. Some literals have language tags that denote the language in which the textual string is written. For instance, `"Pet Sounds"@en` denotes an English literal.

Literals are related to IRIs. For example, a literal may give the name of the thing denoted by the IRI, or it may give a human-readable description of that thing. Once LOTUS has matched the user input string to the literals in its collection, it therefore is able to deduce the IRIs that the user is most likely interested in.

Sometimes LOTUS finds a lot of matches, not all of which have the same relevance. LOTUS therefore *ranks* the results of the matching operation, in order to show the most relevant results first. Since there are billions of possible matches, it is a challenge to perform this search operation very quickly.

As with search engines on the Web of Documents, LOTUS is only the beginning of using the Web of Data: it provides IRI entry points from which users and applications can traverse the interconnected LOD Cloud.

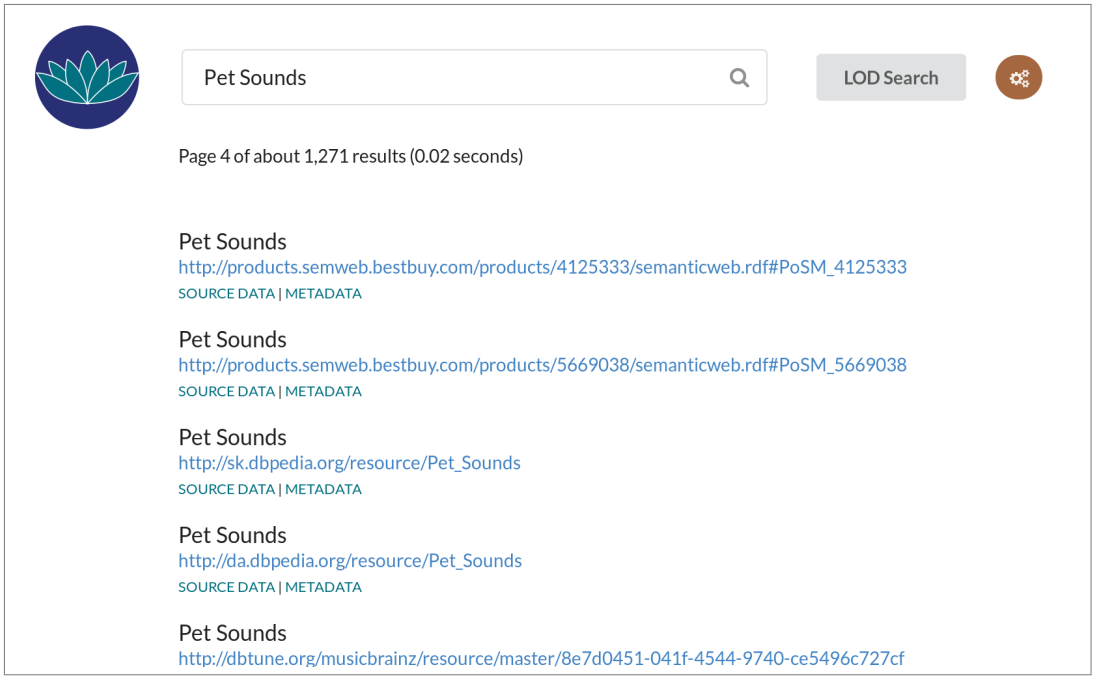

Figure 2: Search results for user input "Pet Sounds". The search string matches and ranks 1,271 IRIs in 0.02 seconds. The displayed results include two Best Buy products, two DBpedia entries (Slovak and Danish language versions) and one MusicBrainz entry.

3 Example search: "Pet Sounds"

Let's illustrate how a typical search operation in LOTUS works. The search string "Pet Sounds" (Figure 2) matches and ranks 1,271 IRIs. Results are retrieved in 0.02 seconds. The top result is a DBpedia IRI[1] that denotes the classic Beach Boys album and describes it from an encyclopedic angle. If we follow the IRI we find generic information about the album such as links to its creators and the year in which it was recorded. Sometimes an IRI cannot be followed because the source data is temporarily offline or is no longer available. To address this problem, every search result in LOTUS has a 'source data' link that points to a cached version of the data.

Other search results denote particular versions of the album that can be bought online through Best Buy[2] or KaufKauf.[3] Following those links results in information that is not included in an encyclopedia, like price information and information about the manufacturer of specific (re-)releases of the album.

[1] `http://dbpedia.org/resource/Pet_Sounds`
[2] `http://products.semweb.bestbuy.com/products/16304508/semanticweb.rdf#PoSM_16304508`
[3] `http://openean.kaufkauf.net/id/EanUpc_0077774842129`

Other IRIs describe the Pet Sounds album as a music piece[4] focusing on the technical aspects of the album such as its recording style. The results also include Yago IRIs to the instrumental version, the Pet Sounds recording sessions, and various live performances of the album.

Further down the list of results we find less popular resources like the Pet Sounds Studio,[5] which was named after the Beach Boys album, and albums that were recorded there.

From this example we see that there are many sides to an entity and different IRIs are used to denote the same entity in different roles (e.g., art piece, audio recording, commercial product).

4 Implementation

LOTUS is implemented on a 5-server ElasticSearch/Lucene cluster hosted at SURF.[6] Because the indexes of LOTUS are very large, they have to be created with Big Data tools, specifically Hadoop[7] and RocksDB.[8] Cached versions of the data are stored in Header Dictionary Triples (HDT)[9] and are exposed through a Linked Data Fragments (LDF)[10] API. Metadata about the source documents is stored in a Virtuoso triple store and exposed through a SPARQL endpoint. Because all components of the LOTUS framework are exposed using standards-compliant web APIs, it is easy for developers to extend LOTUS' functionality.

Search in ElasticSearch works in two steps: a subset of the results are first *matched* and then *ranked* according to some relevance metric. LOTUS implements multiple matching and ranking algorithms. Some matchers focus on fuzzy matching to cover typo's while others perform conjunctive or phrase-based matching. Some rankers allow concepts to be prioritized over instances, while others prioritize IRIs that occur in multiple documents. It is also possible to match literals in a specific language, which opens up the possibility of building a multi-lingual search interface (currently to be found under the 'expert options' button).

The code of the LOTUS API and indexer are available on GitHub.[11]

[4] `http://dbtune.org/musicbrainz/resource/master/8e7d0451-041f-4544-9740-ce5496c727cf`
[5] `http://dbpedia.org/resource/Pet_Sounds_Studio`
[6] See `https://www.surf.nl/en`
[7] See `http://hadoop.apache.org`
[8] See `http://rocksdb.org`
[9] See `http://www.rdfhdt.org/`
[10] See `http://linkeddatafragments.org/`
[11] See `https://github.com/filipdbrsk/LOTUS_Search` and `https://github.com/filipdbrsk/LOTUS_Indexer`

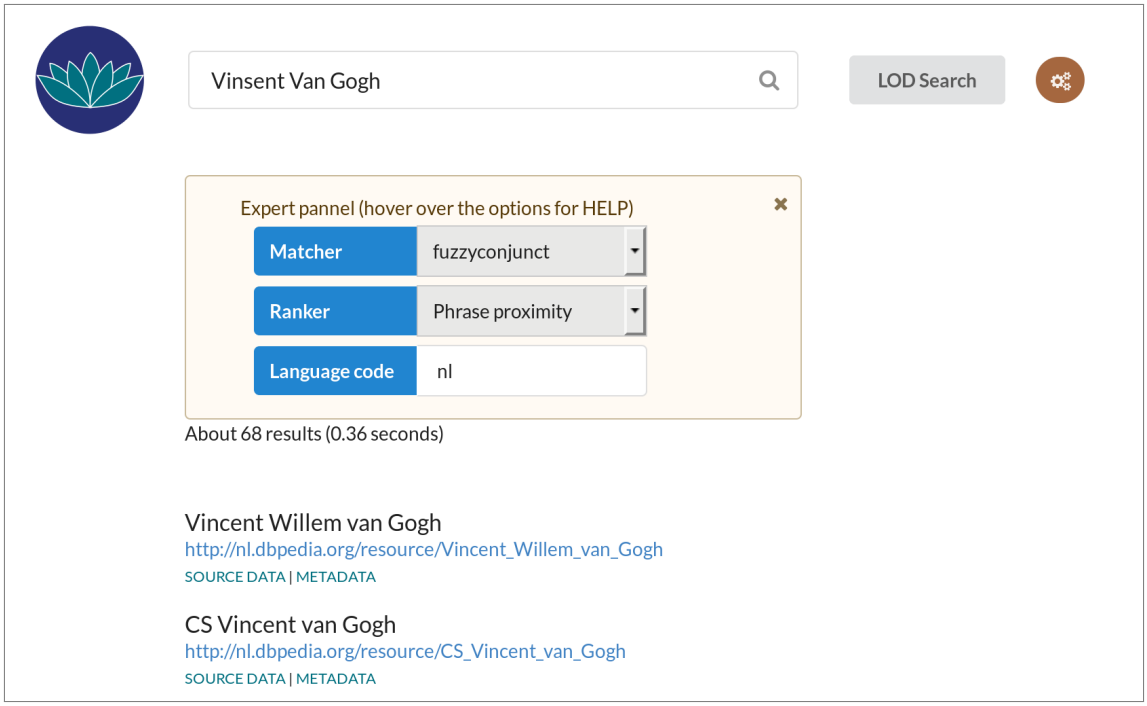

Figure 3: Search results with advanced search options, using fuzzy matching and preferring results from Dutch data sources.

5 Customization & advanced features

Because 'finding the right resource' is often a domain-dependent task, LOTUS provides an adaptive framework in which researchers and developers are able to combine multiple matching and ranking algorithms. Figure 3 illustrates this by using a non-standard matcher, ranker and language code. The fuzzy matcher provides resilience against typo's like the one in Vincent van Gogh's name. The highest-ranked results are all from Dutch data sources because the language code has been set to 'nl'. Notice that some features do come at a price: this particular query takes 0.36 seconds to complete, which is 15 times as slow as the same query under default settings. In total, LOTUS offers 32 combinations of matching and ranking algorithms to play with.

All LOTUS features are exposed through a machine-processable API for others to use. A comprehensive list of query parameters can be found online. It includes the following functionalities:

- Choose a matching algorithm and a (combination of) ranking algorithms.
- Filter on predicate terms (e.g., only match postal codes) and on subject terms (e.g. only results from Cyc).
- Include/exclude blank node results.
- Filter on language codes (e.g., only Dutch labels).

6 Conclusion

LOTUS is a big improvement over existing ways of finding things on the Web of Data. It builds upon a commonly understood application paradigm (search engines) and applies that to the Web of Data. As with the early search engines on the Web of Documents, LOTUS will have to be significantly improved over time. This is why all APIs use Open standards, are freely accessible, and all components can be readily reused by others to build upon. Here are some links to more detailed information about LOTUS:

- Video from ESWC 2016's presentation on LOTUS (`http://videolectures.net/eswc2016_ilievski_linked_data/`)[12]
- Research paper from ESWC 2016 (`http://link.springer.com/chapter/10.1007%2F978-3-319-34129-3_29`)
- Slides from ESWC 2016's presentation on LOTUS (`http://www.slideshare.net/FilipIlievski1/lotus-adaptive-text-search-for-big-linked-data`)
- Workshop paper from the ISWC 2015 COLD Workshop (`http://ceur-ws.org/Vol-1426/paper-06.pdf`)
- Try LOTUS out over at `http://lodsearch.org`

[12]Note: While the visual appearance of LOTUS has been changed since it has been presented at ESWC 2016, its functionality has been kept unchanged. The ESWC 2016 interface of LOTUS is still functional and can be accessed at `http://lodsearch.org/eswc`.