Efficient and Convenient SPARQL+Text Search: a Quick Survey

Hannah Bast and Niklas Schnelle

University of Freiburg
79110 Freiburg, Germany
{bast,schnelle}@cs.uni-freiburg.de

Abstract. This is a quick survey about efficient search on a text corpus combined with a knowledge base. We provide a high-level description of two systems for searching such data efficiently. The first and older system, Broccoli, provides a very convenient UI that can be used without expert knowledge of the underlying data. The price is a limited query language. The second and newer system, QLever, provides an efficient query engine for SPARQL+Text, an extension of SPARQL to text search. As an outlook, we discuss the question of how to provide a system with the power of QLever and the convenience of Broccoli. Both Broccoli and QLever are also useful when only searching a knowledge base (without additional text).

Keywords: Knowledge Bases · Semantic Search · SPARQL+Text · Efficiency · User Interfaces

1 Introduction

This short survey is about efficient search on a text corpus combined with a knowledge base. For the purpose of this paper, a knowledge base is a collection of subject-predicate-object triples, like in the following example. Note that objects can also be strings, called *literals*, and that such literals can contain a qualifier indicating the language.

```
<Neil_Armstrong> <Space_Agency> <NASA>
<Neil_Armstrong> <Place_of_birth> <Wapakoneta>
<Neil_Armstrong> <Date_of_birth> "1930-08-05"
<NASA> <Slogan> "For the Benefit of All"@en
```

Knowledge bases are well suited for structured data, which has a natural representation in the form above. With information cast in this form, we can ask queries with precise semantics, just like on a database. Here is an example query, formulated in SPARQL, the standard query language on knowledge bases. The query asks for astronauts and their agencies. The result is a table with two columns. If an astronaut works for k agencies, the table has k rows for that astronaut.

```
SELECT ?x ?y WHERE {
  ?x <is-a> <Astronaut> .
  ?x <Space_Agency> ?y
}
```

Note that it is crucial that in the knowledge base the same identifier is used to denote the same entity in different triples. This is easier said than done: in practice, knowledge bases are often co-productions of large teams of people, and it is not trivial to ensure that different people use the same identifier when referring to the same entity.

Much of today's information is available only in text form. The main reasons for this are as follows. First, text as a direct representation of spoken language is a very natural form of communication for humans and it requires extra effort to convert a given piece of information into a structured form. Second, as a particular consequence of the first reason, especially current and expert information is much more likely to be available in text form than in structured form. Third, not all information can be meaningfully cast in the above mentioned triple form. For example, consider the following sentence, which, among other pieces of information, expresses that Neil Armstrong walked on the moon:

On July 21st 1969, Neil Armstrong became the first man to walk on the Moon.

The statement that a certain person walked on the moon is rather specific and applies to only few entities. Casting this into triple form might be reasonable for statements of historical importance. Doing it for all statements that are mentioned in a large text corpus, would lead to a knowledge base that has hardly more structure than the text corpus itself. Also note that the larger the set of predicates becomes, the harder it becomes to maintain the above-mentioned property of using consistent identifiers.

It is therefore very natural to consider both knowledge bases and text for search. The simplest way to achieve this is to query both data sets separately and return the results as distinct result sets (provided that matches were found). This is essentially what the big commercial search engines currently do. In this paper, we consider a more powerful search, which considers the knowledge base and the text in combination. In the following, Section 2 explains the nature of this combination. Section 3 presents a system, called *Broccoli*, which enables a user to interactively and conveniently search on such a combined data set. Section 4 presents *SPARQL+Text*, a powerful extension of *SPARQL*, along with *QLever*, an efficient query engine implementing this extension. Section 5 briefly discusses how to combine the power of a system like QLever with the convenience of a system like Broccoli.

Both Broccoli and QLever are available online. A demo of Broccoli is available under `http://broccoli.cs.uni-freiburg.de`. A demo of QLever is available under `http://qlever.cs.uni-freiburg.de`. The source code of QLever is available under `http://github.com/ad-freiburg/qlever`.

2 Combining text and knowledge base data

A natural way to connect a given text corpus to a given knowledge base is to identify which pieces of text refer to an entity from the knowledge base, and exactly which entity is meant in each case. These problems are known as *named entity recognition* (NER) and *named entity disambiguation* (NED). For example, consider the following sentence:

Buzz Aldrin joined Armstrong and became the second human to set foot on the Moon.

Underlining the correct tokens is the NER problem. This entails figuring out the extent of the text used to refer to one entity. This can be just a single word, but it can also be a sequence of two or more words. Identifying which entities from the knowledge base the underlined pieces of text refer to is the NED problem. Note that there are many entities which could be referred to by the word *Armstrong*. Figuring out the correct entity can be a hard problem. In the sentence above, the other words in the sentence make it pretty clear that Neil Armstrong is meant. In the ERD'14 challenge [10], the best approach achieved an F-measure of 76% [12]. A quick overview of the state of the art in NER and NED can be found in a recent survey [7, Section 3.2.2]. In the following, we simply assume that the NER+NED problem has been solved satisfactorily and we thus have a text and a knowledge base linked in the way described.

With a text corpus and a knowledge base linked in this manner, we now have a notion of *co-occurrence* of an entity from the knowledge base with one or more words from the text corpus. The two systems described in Sections 3 and 4 allow to specify such co-occurrences as part of a query. For example, both systems allow a query that searches for entities in the knowledge base with the profession *astronaut* (that is, astronauts), which somewhere in the text co-occur with a word starting with *walk* (walk, walked, walking, ...) and the word *moon*. The scope of the co-occurrence is an additional parameter: for example, we can consider co-occurrence within the same sentence or co-occurrence within the same grammatical sub-clause of a sentence; this is explained further in Section 3. Note how such a co-occurrence is a good indicator that the respective astronaut indeed walked on the moon. The more such co-occurrences we find, the more likely it is. The examples given in the next sections will clarify this further.

To get a feeling for the amount of data which these systems can handle, here are two concrete datasets, which have been used in the evaluation of these systems.

Wikipedia+Freebase Easy: the text from a dump of the English Wikipedia with links to a curated and simplified version of Freebase called *Freebase Easy* [2]. In particular, all entities and predicates in Freebase Easy are denoted by unique human-readable names, like in the example triples at the beginning of the introduction. The dataset is available at `http://freebase-easy.cs.uni-freiburg.de`. The dimensions of this dataset are: 360,744,363 triples, 2,316,712,760 word occurrences, 494,253,129 entity links.

Clueweb+Freebase: the text from the Clueweb12 collection [11] with links to the latest complete version of Freebase (see below). Freebase uses alphanumerical IDs as identifiers for entities (for example, *fb:m.05b6w* for Neil Armstrong) and human-readable strings with a directory-like structure as identifiers for predicates (for example, *fb:people.person.profession* for the predicate that links a person to their profession).[1] The dimensions of this dataset are: 1,934,771,338 triples, 32,281,516,161 word occurrences, 3,263,384,664 entity links.

Freebase was initiated by a company called Metaweb in 2007. The company was eventually acquired by Google in 2010. In August 2015, the Freebase dataset was frozen. Wikidata is a general-purpose knowledge base which is very similar in spirit to Freebase [15]. Wikidata has grown steadily but slowly until August 2017. It has then almost tripled in size over the next 12 months. A full dump of Wikidata from May 2018 contains 4,157,785,636 triples, of which 1,204,269,433 have a literal as an object. Wikidata uses numerical IDs both for entity and predicate names (for example, *wd:Q1615* for Neil Armstrong and *wdt:P106* for the predicate that links a person to their profession). An instance of the query engine described in Section 4 running on the complete Wikidata is available under `http://qlever.cs.uni-freiburg.de`.

3 Broccoli: interactive search on a knowledge base combined with a text corpus

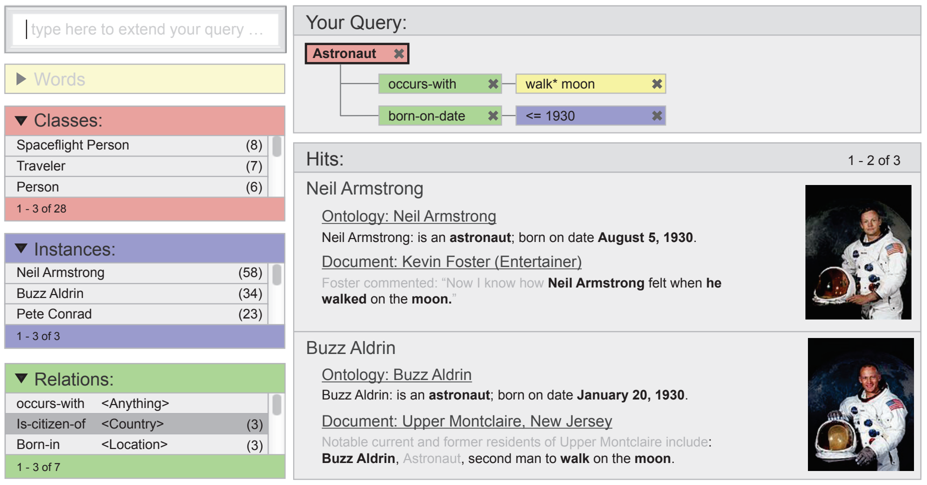
Search on a knowledge base alone is not easy, and in combination with text, the task becomes even harder. The main reason is that such a search requires knowledge of the names of the entities and the predicates in the knowledge base, as well as on how the information is structured in the knowledge base in the first place. This is hard even when the identifiers are human-readable (because there are so many of them, and names are ambiguous). It is complicated further when the identifiers are just numerical (or alphanumerical) IDs.

Another problem is that query languages like SPARQL have no concept of a ranking by *relevance*, as we know it from text search engines. Instead, a SPARQL query primarily delivers a *set* of entities or table rows, and any desired order needs to be explicitly specified. For some queries, there are natural predicates by which the data can be ordered. For example, for a list of cities, an order by descending population is natural. For other queries, there is no such predicate. For example, for a list of people, one probably wants to see the better known individuals first, but knowledge bases usually do not have predicates expressing the relative "popularity" of an entity.

Broccoli is a system that tries to address all of these problems. Broccoli guides the user in incrementally constructing a query by providing suggestions for extending the query after each keystroke and by visualizing, at each step of the construction process, the current query and the current result in an intuitive

[1] The identifiers are actually URIs and the prefix *fb:*stands for the common beginning of these URIs. See Section 5 for more explanation of this.

way. Figure 1 provides a screenshot of the system in action for a particular query. The caption of that figure provides some additional explanations on the various components and features. Note that Broccoli refers to entities as *instances* and predicates as *relations*. Broccoli also knows *classes*, which are simply groups of entities with the same type, according to a fixed type relation, which every general-purpose knowledge base has. The semantics of the query should be self-explanatory; if not, the formalization to SPARQL+Text in Section 4 should help to clarify this. A demo of Broccoli is available online at http://broccoli.cs.uni-freiburg.de.



Classes:

Spaceflight Person	(8)
Traveler	(7)
Person	(6)
1 - 3 of 28	

Instances:

Neil Armstrong	(58)
Buzz Aldrin	(34)
Pete Conrad	(23)
1 - 3 of 3	

Relations:

occurs-with	<Anything>	
Is-citizen-of	<Country>	(3)
Born-in	<Location>	(3)
1 - 3 of 7		

Hits: 1 - 2 of 3

Neil Armstrong

Ontology: Neil Armstrong

Neil Armstrong: is an **astronaut**; born on date **August 5, 1930**.

Document: Kevin Foster (Entertainer)

Foster commented: "Now I know how **Neil Armstrong** felt when **he walked** on the **moon.**"

Buzz Aldrin

Ontology: Buzz Aldrin

Buzz Aldrin: is an **astronaut**; born on date **January 20, 1930**.

Document: Upper Montclaire, New Jersey

Notable current and former residents of Upper Montclaire include: **Buzz Aldrin**, Astronaut, second man to **walk** on the **moon**.

Fig. 1. A screenshot of Broccoli in action for an example query. The box on the top right visualizes the current query as a tree. The large box below shows the hits grouped by instances that match the query root and ranked by relevance. Comprehensive evidence for each hit is provided. For matches in the text corpus, a whole sentence is shown, with parts outside of the matching semantic unit (this is explained in the text) greyed out. On the left, there are suggestions for classes, instances and relations, which can be used to extend or narrow down the query. Suggested classes are parent classes of the entities from the current result. Relations and instances are context sensitive with respect to the current query. That is, all suggestions, if clicked, would lead to at least one hit. There are no word suggestions in the screenshot, because the search field on the top left is empty at this point of the query construction process. As soon as letters are typed, word suggestions appear and the other suggestions are narrowed down to those matching the typed prefix.

The design and realization of Broccoli was a complex endeavour, which required several person-years. The architecture and the technology behind Broccoli is described in a series of papers. The system architecture has first been described in [1]. The index data structures and algorithms used for the interactive query processing and suggestions are described in [4]. The curation and simplification of the Freebase dataset is described in [2]. The engineering behind the public demo is described in [3]. The relevance scores which form the basis of the ranking are described in [6]. The natural language processing used to split the text into semantic units is described in [8].

Broccoli does what it does extremely well. The convenience and the high speed come at a price though. Here are the most important shortcomings of Broccoli:

1. Broccoli only supports tree-shaped SPARQL queries. Many typical queries are tree-shaped, but by far not all.
2. Broccoli only supports SPARQL queries with one variable in the SELECT clause. Again, many typical queries have this property, but by far not all.
3. Broccoli has no special treatment for n-ary relations, that is, relations which connect more than two entities. Section 5 gives an example of such a query on Wikidata.
4. Broccoli does not support SPARQL queries with predicate variables (that is connecting two entities, which themselves may be variables, by an unknown predicate).
5. Broccoli has no query planner. Since the queries are constructed incrementally, by adding one part of a triple at a time, the order of the basic operations (index scans and joins, see Section 4), is completely determined by the way the query is constructed.
6. Broccoli uses a non-standard API. The original focus of the project was on usability aspects and efficiency, not on the underlying query language.

The query engine presented in the next section addresses all of these shortcomings.

4 QLever: a query engine for SPARQL+Text

QLever is a query engine for what we call SPARQL+Text. SPARQL+Text contains standard SPARQL, so QLever can also be used to process standard SPARQL queries (and quite efficiently so, see below). SPARQL+Text queries operate on a knowledge base linked to a text corpus, as explained in Section 2. It is assumed that the text has been segmented beforehand. These segments can be the semantic units of the sentences (as briefly explained in Section 3), or simply the sentences of the text. The search results are best if the segmentation is such that co-occurrence in the same segment has a semantic meaning, as in the "astronauts who walked on the moon" example above.[2]

Specifically, SPARQL+Text extends SPARQL by two built-in predicates *ql:contains-entity* and *ql:contains-word*. Here is an example query, which is similar to the query from Figure 1 (but asking for the space agencies of the astronauts instead of restricting their birth date).

[2] We sweep under the rug here that this is not a matter of co-occurrence alone. For example, a text segment may additionally contain the word *not* and thus negate the meaning. There are different approaches to handle this which we do not discuss here.

```
SELECT ?astronaut ?agency TEXT(?text) WHERE {
  ?astronaut <is-a> <Astronaut> .
  ?astronaut <Space_Agency> ?agency .
  ?text ql:contains-entity ?astronaut .
  ?text ql:contains-word "walk*" .
  ?text ql:contains-word "moon" .
}
ORDER BY DESC(SCORE(?text))
```

The last three triples in the WHERE clause express that there is a segment of text, denoted by *?text*, which contains a mention of an entity *?astronaut* (which was identified as part of the NER+NED preprocessing described in Section 2) as well as a word starting with *walk* and the word *moon*. With a reasonable segmentation, a large number of such segments means that the respective candidate is indeed an astronaut who walked on the moon. On the two collections listed in Section 2, this query already gives very good results when the segments are simply sentences. The *TEXT(?text)* yields a third column in the result table, containing a matching text segment for each match for the remaining variables in the SELECT clause.[3] The final *ORDER BY ...* clause orders the results by the number of matching text segments, results with most matches first.

The details of the query engine behind QLever and the results of an extensive evaluation are provided in [5]. The source code and documentation is available under `http://github.com/ad-freiburg/qlever`. A front-end for entering SPARQL+Text queries on the datasets from Section 2 as well as on the complete Wikidata is available under `http://qlever.cs.uni-freiburg.de`.

We briefly provide the main ideas and results in a nutshell. The basic idea of the index structure behind QLever is similar to that of Broccoli, but with various engineering improvements. For example, the index data structures are laid out (with slight redundancy) such that all basic operations (such as obtaining a range of items from an index list) can efficiently read only exactly that data which is actually needed for the operations (for example, only the subjects from a range of triples). Due to these improvements, QLever is 2-3 times faster on queries which can also be processed by Broccoli.

Unlike Broccoli, QLever has a full-featured query planner, suited for arbitrary query graphs and with a special treatment of the two special *ql:contains-...* predicates. The query planner finds the query execution plan with the best estimated cost, using dynamic programming. For accurate cost estimations, for each subquery not only the result size is estimated, but also the number of distinct elements in each column. This is crucial for a good estimation of the result size of the join operations. Computing good estimates for the result size and the number of distinct elements in each column for each of QLever's basic operations (including the operations involving text search) is not trivial.

[3] The number of matching text segments shown (per match for the remaining variables in the SELECT clause) can be controlled with a *TEXTLIMIT <k>* clause. The default is TEXTLIMIT 1.

The performance improvements over existing SPARQL engines are considerable. Even for SPARQL queries without a text search component, QLever is several times faster than existing query engines like Virtuoso [14] (widely used in the commercial world) and RDF-3X [13] (a state-of-the-art research prototype) for most queries. On SPARQL+Text queries, which can only be simulated on query engines like Virtuoso and RDF-3X, QLever is faster by several orders of magnitude.

5 Outlook

QLever supports SPARQL+Text, a powerful extension of SPARQL to integrate search on a given text corpus linked to a given knowledge base. In particular, QLever addresses all the shortcomings of Broccoli listed at the end of Section 3. The price is that there is currently no UI that enables the construction of arbitrary SPARQL+Text queries with the same level of convenience as Broccoli.

To illustrate the need for such a UI, let us look at one more query. It is a SPARQL query on Wikidata for computing a table of all astronauts and where they studied and for which degree. This query involves a 3-ary predicate (linking a person to a university and the degree) and it is already surprisingly complex. What adds to the complexity is that Wikidata uses IDs for both entities and predicates and that we need to use a special predicate for obtaining the human-readable labels and additional FILTER clauses to get the labels only in one language instead of hundreds. Note that this query does not even involve a text search part (we could, however, just add the two text search triples from the query above). The prefixes at the beginning are abbreviations for the common prefixes of Wikidata's IDs, which are URIs. For example, in the SPARQL query below, *wd:Q11631* is equivalent to *<http://www.wikidata.org/entity/Q11631>*.

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX pq: <http://www.wikidata.org/prop/qualifier/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?astronautLabel ?universityLabel ?degreeLabel
WHERE {
  ?astronaut wdt:P106 wd:Q11631 .
  ?astronaut rdfs:label ?astronautLabel .
  ?astronaut p:P69 ?educatedAt .
  ?educatedAt ps:P69 ?university .
  ?university rdfs:label ?universityLabel .
  ?educatedAt pq:P512 ?degree .
  ?degree rdfs:label ?degreeLabel .
  FILTER langMatches(lang(?astronautLabel), "en")
  FILTER langMatches(lang(?universityLabel), "en")
  FILTER langMatches(lang(?degreeLabel), "en")
}
```

Enabling such queries, or more complex ones, without requiring knowledge of a particular query language or of the particularities of the knowledge base is a challenging problem. One line of attack is to try to generalize the UI of Broccoli to SPARQL+Text. This looks feasible, but it remains to be seen how practical such a UI would be when queries become more complex. Another line of attack is to automatically translate questions in natural language to SPARQL+Text queries as the above. This has been proven very hard already for simple SPARQL queries, for example see [9]. A compromise might be a hybrid approach, which allows an incremental construction of such a query via elements formulated in natural language.

References

1. Bast, H., Bäurle, F., Buchhold, B., Haussmann, E.: Broccoli: Semantic full-text search at your fingertips. CoRR **abs/1207.2615** (2012)
2. Bast, H., Bäurle, F., Buchhold, B., Haußmann, E.: Easy access to the freebase dataset. In: WWW (Companion Volume). pp. 95–98. ACM (2014)
3. Bast, H., Bäurle, F., Buchhold, B., Haußmann, E.: Semantic full-text search with broccoli. In: SIGIR. pp. 1265–1266. ACM (2014)
4. Bast, H., Buchhold, B.: An index for efficient semantic full-text search. In: CIKM. pp. 369–378. ACM (2013)
5. Bast, H., Buchhold, B.: Qlever: A query engine for efficient sparql+text search. In: CIKM. pp. 647–656. ACM (2017)
6. Bast, H., Buchhold, B., Haussmann, E.: Relevance scores for triples from type-like relations. In: SIGIR. pp. 243–252. ACM (2015)
7. Bast, H., Buchhold, B., Haussmann, E.: Semantic search on text and knowledge bases. Foundations and Trends in Information Retrieval **10**(2-3), 119–271 (2016)
8. Bast, H., Haussmann, E.: Open information extraction via contextual sentence decomposition. In: ICSC. pp. 154–159. IEEE Computer Society (2013)
9. Bast, H., Haussmann, E.: More accurate question answering on freebase. In: CIKM. pp. 1431–1440. ACM (2015)
10. Carmel, D., Chang, M., Gabrilovich, E., Hsu, B.P., Wang, K.: Erd'14: Entity recognition and disambiguation challenge. SIGIR Forum **48**(2), 63–77 (2014)
11. ClueWeb: (2012), the Lemur Projekt http://lemurproject.org/clueweb12
12. Cucerzan, S.: Large-scale named entity disambiguation based on wikipedia data. In: EMNLP-CoNLL. pp. 708–716. ACL (2007)
13. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. VLDB J. **19**(1), 91–113 (2010), http://dx.doi.org/10.1007/s00778-009-0165-y
14. OpenLink: The Virtuoso Project https://virtuoso.openlinksw.com
15. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)