

Broccoli: Semantic Full-Text Search at your Fingertips

Hannah Bast, Florian Bäumle, Björn Buchhold, Elmar Haussmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast,baeurlef,buchholb,haussmann}@informatik.uni-freiburg.de

ABSTRACT

We present Broccoli, a fast and easy-to-use search engine for what we call semantic full-text search. Semantic full-text search combines the capabilities of standard full-text search and ontology search. The search operates on four kinds of objects: ordinary words (e.g., *edible*), classes (e.g., *plants*), instances (e.g., *Broccoli*), and relations (e.g., *occurs-with* or *native-to*). Queries are trees, where nodes are arbitrary bags of these objects, and arcs are relations. The user interface guides the user in incrementally constructing such trees by instant (search-as-you-type) suggestions of words, classes, instances, or relations that lead to good hits. Both standard full-text search and pure ontology search are included as special cases. In this paper, we describe the query language of Broccoli, the main idea behind a new kind of index that enables fast processing of queries from that language as well as fast query suggestion, the natural language processing required, and the user interface. We evaluated query times and result quality on the full version of the English Wikipedia (40 GB XML dump) combined with the YAGO ontology (26 million facts). We have implemented a fully functional prototype based on our ideas and provide a web application to reproduce our quality experiments. Both are accessible via <http://broccoli.informatik.uni-freiburg.de/repro-corr/>.

1. INTRODUCTION

In this paper, we describe a novel implementation of what we call *semantic full-text search*. Semantic full-text search combines traditional *full-text search* with structured search in knowledge databases or *ontology search* as we call it in this paper.

In traditional full-text search you type a (typically short) list of keywords and you get a list of documents containing some or all of these keywords, hopefully ranked by some notion of relevance to your query. For example, typing *broccoli leaves edible* in a web search engine will return lots of web pages with evidence that broccoli leaves are indeed edible.

In ontology search, you are given a knowledge database which you can think of as a store of subject-predicate-object triples. For example, *Broccoli is-a plant* or *Broccoli native-to Europe*. These triples can be thought of to form a graph of entities (the nodes) and relations (the edges), and ontology search allows you to search for subgraphs matching a given pattern. For example, find all plants that are native to Europe.

Many queries of a more “semantic” nature require the combination of both approaches. For example, consider the query *plants with edible leaves and native to Europe*, which will be our running example in this paper. A satisfactory answer for this query requires the combination of two kinds of information. First, a list of plants native to Europe. This is hard for full-text search but a showcase for ontology search, see above. Second, for each plant the information whether its leaves are edible or not. This kind of information can be easily found with a full-text search for each plant, see above. But it is quite unlikely (and unreasonable) to be contained in an ontology, for reasons explained in Section 2.3.

The basic principle of our combined search is to find *contextual* co-occurrences of the words from the full-text part of the query with entities matching the ontology part of the query. Consider the sentence: *The stalks of rhubarb are edible, but its leaves are toxic*. Assume for now that we can recognize entities from the ontology in the full text (we come back to this in Section 3.2). In this case, the two underlined words both refer to *rhubarb*, which our ontology knows is a plant that is native to Europe. Obviously, this sentence should *not* count as evidence that *rhubarb leaves are edible*. We handle this by decomposing each sentence into what we call its *contexts*: the parts of the sentence that “belong” together. In this case the *stalks of rhubarb are edible* and *rhubarb leaves are toxic*. An arc from the query tree now matches if and only if its elements co-occur in one and the same context.

Figures 1 and 2 show screenshots of our search engine in action for our example query. The figures and their captions also explain how the query can be constructed incrementally in an easy way and without requiring knowledge of a particular query language on the part of the user. We encourage the reader to try our online demo that is accessible via <http://broccoli.informatik.uni-freiburg.de/repro-corr/>.

▶ Words

▼ Classes:

Garden plant	(24)
House plant	(17)
Crop	(16)

1 - 3 of 28

▼ Instances:

Broccoli	(58)
Cabbage	(34)
Lettuce	(23)

1 - 3 of 421

▼ Relations:

occurs-with	<Anything>	
cultivated-in	<Location>	(67)
belongs-to	<Plant family>	(58)

1 - 3 of 7

Your Query:

Plant

- occurs-with edible leaves
- native-to Europe

Hits: 1 - 2 of 421


Broccoli

Ontology: Broccoli

Broccoli: is a **plant**; native to **Europe**.

Document: Edible plant stems

The **edible** portions of **Broccoli** are the stem tissue, the flower buds, as well as the **leaves**.



Cabbage

Ontology: Cabbage

Cabbage: is a **plant**; native to **Europe**.

Document: Cabbage

The only part of the **plant** that is normally **eaten** is the **leafy** head.




Figure 1: A screenshot of the final result for our example query. The box on the top right visualizes the current query as a tree. There is always one node in focus (shown in bold), in this case, the root of the tree. The large box below shows the hits grouped by instance (of the class from the root node) and ranked by relevance (if Broccoli is among the hits, we always rank it first). Evidence both from the ontology and the full text is provided. For the latter, a whole sentence is shown, with parts outside of the matching context grayed out. With the search field on the top left, the query can be extended further. The four boxes below provide context-sensitive suggestions that depend on the current focus in the query, here: suggestions for subclasses of plants, suggestions for instances of plants that lead to a hit, suggestions for relations to further refine the query. One of the suggestions is always highlighted, in this case the *cultivated-in* relation. It can be directly added to extend the query by pressing Return.

1.1 Our contribution

Broccoli supports a subset of SPARQL¹ (essentially trees with a single free variable at the root) for the ontology part of queries. Moreover, it allows a special *occurs-with* relation that can be used to specify co-occurrence of a class (e.g., *plant*) or instance (e.g., *Broccoli*) with an arbitrary combination of words, instances, and further subqueries. Both traditional full-text search and pure ontology search are subsumed as special cases. This gives a very powerful query language. See Section 4 for details.

For the *occurs-with* relation, we provide a novel kind of pre-processing that decomposes sentences into *contexts* of words that belong together. In particular, this considers enumerations and sub-clauses. Previous approaches have used co-occurrence in a whole paragraph or sentence, or based on word proximity; all of these often give poor results. See Section 3 for details.

We present the key idea behind a novel kind of index that supports fully interactive query times of around 100 milliseconds and less for a collection as large as the full English Wikipedia (40 GB XML dump, 418 million contexts of the kind just described). Previous approaches, including adaptations of the classic inverted index, yield query times on the order of seconds or even minutes for the kind of queries

we support on collections of this size. See Section 2.1 for related work, and Section 5 for details.

All the described features have been implemented into a fully functional system with a comfortable user interface. There is a single search field, as in full-text search, and suggestions are made after each keystroke. This allows the user to incrementally construct semantic full-text queries without prior knowledge of a query language. Results are ranked by relevance and grouped by instance, and displayed together with context snippets that provide full evidence for why that particular instance is shown. See Figures 1 and 2 for an example, and Section 6 for details.

We provide experimental results on the result quality for the English Wikipedia combined with the YAGO ontology [20]. For the quality results, we used 46 Queries from the SemSearch List Search Track (e.g., *Apollo astronauts who walked on the Moon*), 15 queries from the TREC 2009 Entity Track benchmarks (e.g., *Airlines that currently use Boeing 747 planes*) and 10 lists from Wikipedia (e.g. *List of participating nations at the Winter Olympic Games*). We allow reproducing our results at <http://broccoli.informatik.uni-freiburg.de/repro-corr/>. See Section 7 for the details of our experiments.

We want to remark that the natural language processing, the index, and the user interface behind Broccoli are com-

¹<http://www.w3.org/TR/rdf-sparql-query>

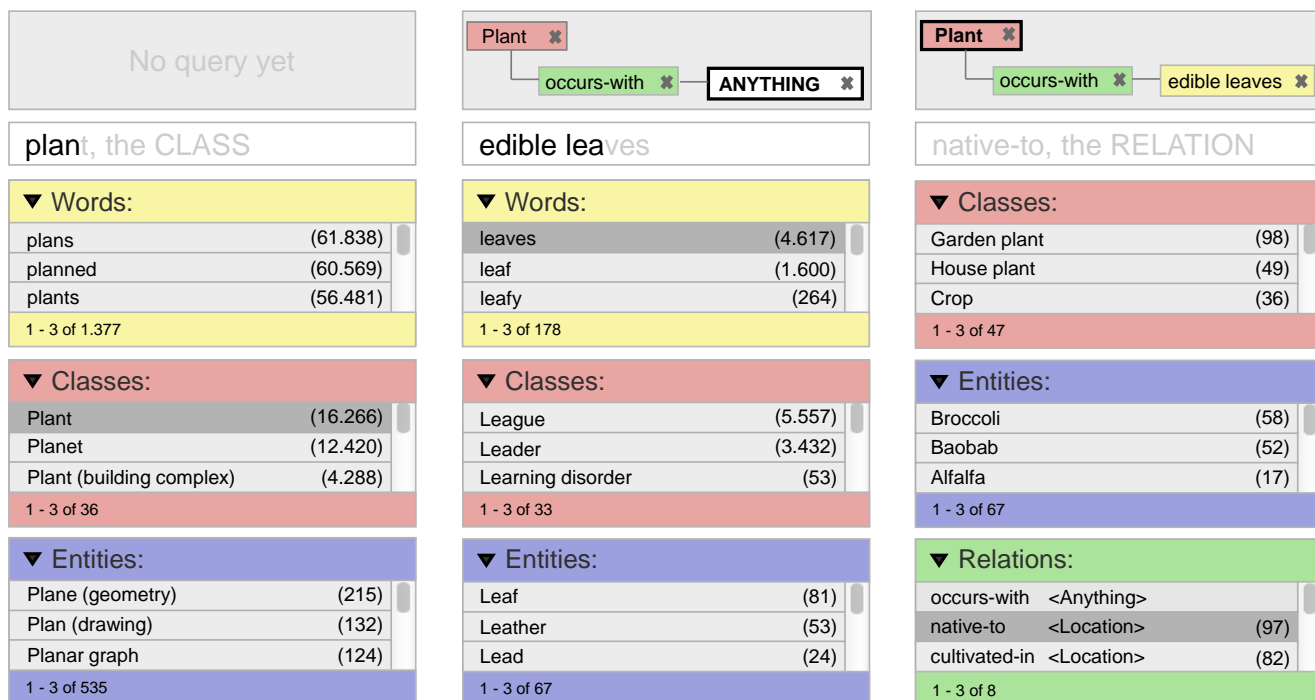


Figure 2: Snapshots of the query, search field, and suggestion boxes for three stations in the construction of our example query. Column 1: At the beginning of the query, after having typed *plan*. Column 2: After the class *plant* has been selected and the *occurs-with* relation has been added and having typed *edible lea*. Column 3: After having selected *edible leaves*. The focus automatically goes back to the root node.

plex problems each on their own. The contribution of this paper is the overall design of the system, the basic ideas for each of the mentioned components, an implementation of a fully functional prototype based on these ideas, and a first performance and quality evaluation providing a proof of concept. Optimization of the various components is the next step in this line of research; see Section 8.

2. RELATED WORK

Putting the work presented in this paper into context is hard for two reasons. First, the literature on semantic search technologies is vast. Second, “semantic” means so many different things to different researchers. We roughly divide work in this broad area into four categories, and discuss each category separately in the following four subsections.

2.1 Combined ontology and full-text search

Ester [7] was the first system to offer efficient combined full-text and ontology search on a collection as large as the English Wikipedia. Broccoli improves upon Ester in three important aspects. First, Ester works with inverted lists for classes and achieves fast query times only on relatively simple queries. Second, Ester does not consider contexts but merely syntactic proximity of words / entities. Third, Ester’s simplistic user interface was ok for queries with one relation, but practically unusable for more complex queries.

Various other systems offering combinations of full-text and ontology search have been proposed. Semplore [22] supports a query language similar to ours. However, elements from the ontology are not recognized in their contexts, but

there is simply one piece of text associated with each instance (which would correspond to a single large context in our setting). Queries are processed with a standard inverted index, and no particular UI is offered. In Hybrid Search [8], the full text and the ontology are searched separately with standard methods (Lucene and Sesame), and then the results are combined. There is no particular natural language processing. Concept Search [15] adds information about identified noun phrases and hyponyms to the index. Queries are bags of words, which are interpreted semantically. The query processing uses standard methods (Lucene), with very long inverted lists for the semantic index items. GoNTogle [14] combines full text with annotations which are searched separately and then combined, similarly as in [8]. Queries are bags of words. There is no full ontology search and no particular natural language processing. Faceted Wikipedia Search [16] offers a user interface with similarities to ours. However, the query language is restricted, there is nothing comparable to our contexts but only a small abstract per entity like in [22], and query processing is DB-based and very slow, despite the relatively small amount of data. SIREN² provides an integration of pure ontology search into Lucene. How to combine the then possible full-text and ontology searches is up to the user of the framework. Finally, systems like [23] try to interpret a given keyword query semantically and translate it into a suitable SPARQL query for pure ontology search.

²<http://siren.sindice.com>

2.2 Systems for entity retrieval

Entity retrieval is a line of research which focuses on search requests and corresponding result lists centered around entities (instead of around documents, as in traditional search). Since 2009, there is also a corresponding Entity Track at TREC³. The tasks of this track are both simpler and harder than what we aim at in this paper.

They are harder because the overall goal is entity retrieval from *web pages*. The ClueWeb09 collection introduced at TREC 2009 is 25 TB of text. The relative information content is, however, low as is typical for web contents. Moreover, identifying a representative web page for an entity is part of the problem.

To make the tasks feasible at all under these circumstances, the queries are relatively simple. For example, *Airlines that currently use Boeing 747 planes*.⁴ Even then the tasks remain very hard, and, for example, *NDCG@R* figures average only around 30% even for the best systems [4].

Broccoli queries can be trees of arbitrary degree and depth. All entities that have a Wikipedia page are supported. And, most importantly, the query process is interactive, providing the user with *instant* feedback of what is in the collection and why a particular result appears. This is key for constructing queries that give results of high quality.

The price we pay is a more extensive pre-processing assuming a certain “cleanliness” of the input collection. Our natural language processing currently requires around 1600 core hours on the 40 GB XML dump of the English Wikipedia. And Wikipedia’s rule of linking the first occurrence of an important entity in an article to the respective Wikipedia article helps us for an entity recognition of good quality; see Section 3.2. Bringing Broccoli’s functionality to web search is a very reasonable next step, but out of scope for this article.

Another popular form of entity retrieval is known as *ad-hoc object retrieval* [18]. Here, the search is on structured data, as discussed in the next subsection. Queries are given by a sequence of keywords, similar as in full-text search, for example, *doctors in barcelona*. Then query interpretation becomes a non-trivial problem; see Section 2.4.

2.3 Information extraction and ontology search

Systems for ontology search have reached a high level of sophistication. For example, RDF-3X can answer complex SPARQL queries on the Barton dataset (50 million triples) in less than a second on average [17].

As part of the Semantic Web / Linked Open Data [9] effort, more and more data is explicitly available as fact triples. The bulk of useful triple data is still harvested from text documents though. The information extraction techniques employed range from simple parsing of structured information (for example, many of the relations in YAGO or DBpedia [2] come from the Wikipedia info boxes) over pattern matching (e.g., [1]) to complex techniques involving non-trivial natural language processing like in our paper (e.g., [5]). For a relatively recent survey, see [19].

Our work differs from this line of research in two important aspects: (1) the full text remains part of the index that

is searched at query time; and (2) our system is fully interactive and keeps the human in the loop in the information extraction process. This has the following advantage:

Ontologies are good for facts like *which plants are native to which regions, who was born where on which date*, etc. Such facts are easy to define and can be extracted from existing data sources in large quantity and with reasonable quality. And once in the ontology, they are easily combinable, permitting queries that would not work with full-text search.

But for more complex facts like our *broccoli has edible leaves*, it is the other way round. They are easy to express and search in full text, but tedious to define, include, and maintain in an ontology. Let alone the problem of guessing the right relation names when searching for them.

By keeping the full text, we can leverage the intelligence of the user at query time. The query *Plant occurs-with edible leaves* does not specify the type of the relation between the occurrence of the plant and the occurrence of the words *edible* and *leaves*. Yet a moment’s thought reveals that it is quite likely that a context matching these elements gives us what we want. Similarly as in full-text search, there is often no need to be overly precise in order to get what you want. And just like the result snippets in full-text search, Broccoli’s result snippets provide instant feedback on whether the listed plant is really one with edible leaves.

Finally, if information extraction is desired nevertheless, Broccoli can be a useful tool for interactively exploring the collection with respect to the desired information, and for formulating appropriate queries.

2.4 Systems for question answering

Question answering (QA) systems provide similar functionality as our semantic full-text search. The crucial difference is that questions can be asked in natural language, which makes the answering part much harder. The system is burdened with the additional and very complex task of “translating”, in one way or the other, the given natural language query into a more formal query or queries that can be fed to a search engine and / or a knowledge database.

The perfect QA system would obviate the need for a system like ours here. But research is still far from achieving that goal. All state-of-the-art QA systems, including the big commercial ones, are specialized to quite particular kinds of questions. For example, Wolfram Alpha works perfectly for *Which cities in China have more than 10 million inhabitants*, but does not work if *more* is replaced by *less* or *China* by *Asia*, and does not even understand the question *Which plants have edible leaves*. IBM’s Watson was tuned for finding the single most probable entity when given one of the (intentionally obscured) clues from the Jeopardy! game. And both of these systems lack transparency: it is hard to predict whether a question will be understood correctly, it is hard to understand the reasons for a missing or wrong answer, and there is no possibility of interaction or query refinement.

For our semantic full-text search both the query language and the relation between a given query and its result are well-defined and maximally transparent to the user; see the discussion in Section 2.3. The price we pay is query formulation in a non-natural language. The success of full-text search has shown that as long as the language is simple enough, it can work.

³<http://ilps.science.uva.nl/trec-entity>

⁴In our framework these are queries with two nodes and one *occurs-with* edge.

3. INPUT DATA AND NATURAL LANGUAGE PRE-PROCESSING

3.1 Input data

Broccoli requires two kinds of inputs, a text collection and an ontology. The text collection consists of documents containing plain text. The ontology consists of typed *relations* with each relation containing an arbitrary set of fact triples. The subjects and objects of the triples are called *instances*. Each instance belongs to one or more *classes*. The classes are organized in a taxonomy; the root class is called *Entity*.

3.2 Entity recognition

The first step is to identify mentions of or referrals to instances from the ontology in the text documents. Consider the following sentence, which will be our running example for this section:

(S) *The usable parts of rhubarb, a plant from the Polygonaceae family, are the medicinally used roots and the edible stalks, however its leaves are toxic.*

Both *rhubarb* and *its* refer to the instance *Rhubarb* from our ontology, which in turn belongs to the classes *Plant* and *Vegetable* (among others).

Our entity recognition on the English Wikipedia is simplistic but reasonably effective. As a rule, first occurrences of entities in Wikipedia documents are linked to their Wikipedia page. When parsing a document, whenever a part or the full name of that entity is mentioned again in the same section of the document (for example, *Einstein* referring to *Albert Einstein*), we recognize it as that entity.

We resolve anaphora in an equally simplistic way. Namely, we assign each occurrence of *he*, *she*, *it*, *her*, *his*, etc. to the last recognized entity of matching gender. We also recognize the pattern *the <class>* as the entity of the document if it belongs to *<class>*, for example, *the plant* in the document of *Broccoli*.

Our results in Section 7.5 suggest that, on Wikipedia, these simple procedures give already a reasonable accuracy.

3.3 Natural language processing

The second step is to decompose document texts into what we call *contexts*, that is, sets of words that “belong” together. The contexts for our example sentence (S) from above are:

- (C1) *rhubarb, a plant from the Polygonaceae family*
- (C2) *The usable parts of rhubarb are the medicinally used roots*
- (C3) *The usable parts of rhubarb are the edible stalks*
- (C4) *however rhubarb leaves are toxic*

This will be crucial for the quality of our results, because we do not want to get *rhubarb* in our answer set when searching for *plants with edible leaves*. Note that we assume here that the entity recognition and anaphora resolution have already been done (underlined words). Also note that we do not care whether our contexts are grammatically correct and form a readable text. This distinguishes our approach from a line of research called *text simplification* [12].

In the following, we will only consider contexts that are part of a single sentence. Indeed, after anaphora resolution, it seems that most simple facts are expressed within one and the same sentence. Our evaluation in Section 7.5 confirms this assumption.

Our context decomposition consists of two parts, each described in the following subsections.

3.3.1 Sentence constituent identification (SCI)

The task of SCI is to identify the basic “building blocks” of a given sentence. For our purposes various kinds of *sub-clauses* and *enumeration items* will be important, because they usually contain separate facts that have no direct relationship to the other parts of the sentence. For example, in our sentence (S) from above, the relative clause *a plant from the Polygonaceae family* refers to *rhubarb* but has nothing to do with the rest of the sentence. Similarly, the two enumeration items *the medicinally used roots* and *the edible stalks* have nothing to do with each other (except that they both refer to *rhubarb*); in particular, *rhubarb roots* are not edible and *rhubarb stalks* are not medicinally used. Finally the part *however its leaves are toxic* needs to be considered separate from the preceding part of the sentence. As will become clear in the following, we consider these as enumeration items on the top level of the sentence.

Formally, SCI computes a tree with three kinds of nodes: *enumeration (ENUM)*, *sub-clause (SUB)*, and *concatenation (CONC)*. The leaves contain parts of the sentence and a concatenation of the leaves from left to right yields the whole sentence again. See Figure 3 for the SCI tree of the above sentence.

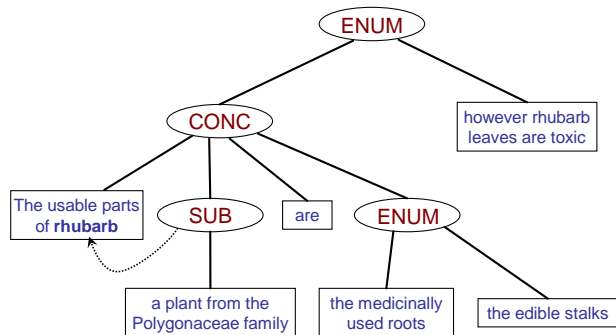


Figure 3: The SCI tree for our example sentence after anaphora resolution. The head of the sub-clause is printed in bold.

We construct our SCI trees based on the output of a state-of-the-art constituent parser. We use SENNA [13], because of its good trade-off between parse time (around 35ms per sentence) and result quality (see Section 7.5).

We transform the parse tree using a relatively small set of hand-crafted rules. Here is a selection of the most important rules; the complete list consists of only 11 rules but is omitted here for the sake of brevity. In the following description when we speak of an *NP* (noun phrase), *VP* (verb phrase), *SBAR* (subordinate clause), or *PP* (prepositional phrase) we refer to nodes in the parse tree with that tag.

(SCI 1) Mark as ENUM each node, for which the children (excluding punctuation and conjunctions) are either all *NP* or all *VP*.

(SCI 2) Mark as SUB each *SBAR*. If it starts with a word from a positive-list (e.g., *which* or *who*) define the first *NP* on the left as the *head* of this SUB; this will be used in (SCR 0) below.

(SCI 3) Mark as SUB each *PP* starting with a preposition from a positive-list (e.g., *before* or *while*), and all *PPs* at the beginning of a sentence. These SUBs have no head.

(SCI 4) Mark as CONC all remaining nodes and contract away each CONC with only text nodes in its subtree (by merging the respective text).

As our quality evaluation in Section 7.5 shows, our rules work reasonably well.

3.3.2 Sentence constituent recombination (SCR)

In SCR we recombine the constituents identified by the SCI to form our *contexts*, which will be the units for our search. Recall that the intuition is to have contexts such that only those words which “belong” together are in the same context. SCR recursively computes the following contexts from a SCI tree or subtree:

(SCR 0) Take out each subtree labeled SUB. If a head was defined for it in (SCI 2), add that head as the leftmost child (but leave it in the SCI tree, too). Then process each such subtree and the remaining part of the original SCI tree (each of which then only has ENUM and CONC nodes left) separately as follows:

(SCR 1) For a leaf, there is exactly one context: the part of the sentence stored in that leaf.

(SCR 2a) For an inner node, first recursively compute the set of contexts for each of its children.

(SCR 2b) If the node is marked ENUM, the set of contexts for this node is computed as the *union* of the sets of contexts of the children.

(SCR 2c) If the node is marked CONC, the set of contexts for this node is computed as the *cross-product* of the sets of contexts of the children.

We remark that once we have the SCI tree, SCR is straightforward, and that the time for both SCI + SCR is negligible compared to the time needed for the full-parse of the sentences.

4. QUERY LANGUAGE

Queries to Broccoli are rooted trees with arcs directed away from the root. The root is either a class or an instance. There are two types of arcs: *ontology arcs* and *occurs-with arcs*. Both have a class or instance as source node.

Ontology arcs are labeled by a relation from the ontology. The two nodes must be classes or instances matching the source and target type of the relation. The class or instance at the target node may be the root of another arbitrary tree.

For occurs-with arcs, the target node can be an arbitrary set of words, prefixes, instances or classes. The instances or classes may themselves be the root of another arbitrary query. Example queries are given in Figures 1 and 2.

To give an example of a more complex query: in Figure 1 we could replace the instance node *Europe* by a class node *Location* and add to it an *occurs-with* arc with the word *equator* in its target node. The intention of this query would be to obtain plants with edible leaves native to regions at or near the equator.

5. INDEX AND QUERY PROCESSING

The index and query processing of Broccoli are described in detail in [6]. In this section, we summarize why standard indexes are not suited for Broccoli and describe the main idea behind our new index.

There are sophisticated systems for both, full-text search and search in ontologies. Since our queries combine both tasks, three ways to answer our queries using those system come to mind: (1) incorporate ontology information into an inverted index; (2) incorporate full-text information into a triple store; (3) use an inverted index for the full-text part of the query, a triple store for the ontology part of the query, and then combine the results somehow.

Neither approach is perfectly suited for our use-case. In a nutshell, approach (1) produces document-centric results and cannot be used to answer complex queries that involve join operations. Approach (2) needs a relation (e.g. *occurs-in-context* featuring both, words and entities) of the size of our entire index to make use of the contexts produced in our contextual sentence decomposition. Efficient queries require a special purpose index over this relation, which already goes in the direction of our approach. Finally, approach (3) will get a list of contexts as a result from the full-text index and has to derive all entities that occur in those contexts. This mapping is not trivial to achieve efficiently, especially since a full mapping from contexts to entities usually does not fit in memory for large collections. Apart from that, we allow queries that demand co-occurrence with some entity from a list that can be the root of another query (e.g. a query for politicians that are friends with an astronaut who walked on the moon). This would require a second mapping in the other direction: from entities to contexts. In summary, the two problems are: Given a list of contexts C , produce a list E of entities that occur in those contexts. Given a list of contexts C and an entity list E , limit C to contexts that include at least one entity from E .

The main idea behind our new index solves these two problems. We use what we call *context lists* instead of standard inverted lists. The context list for a prefix contains one index item per occurrence of a word starting with that prefix, just like the inverted list for that prefix would. But along with that it also contains one index item for each occurrence of an arbitrary entity in the same context as one of these words. For example, consider the context *the usable parts of rhubarb are its edible stalks*, with recognized entities underlined. And let us assume that we have an inverted list for each 4-letter prefix. Then the part of the context list for *edib** pertaining to this context (which has id, say, 14) would be:

...	C14	C14	C14	...
edib*:	#edible	#Rhubarb	#Stalk	...
	1	1	1	...
	8	5	9	...

The numbers in the first row are context ids. The # in the second row means that not the actual entities (with capital letters) or words are stored, but rather unique ids for them. The third row contains the score for each index item. The fourth row contains the position of the word or entity in the respective context. The context lists are sorted by context id, and, for equal context ids, by word/entity id, with entities coming after the words.

Since entity postings are included in those lists, we can easily solve the two problems introduced above. Actually, our index and query processing support many additional features like excerpt generation, suggestions, prefix search, search for documents instead of entities or ranges over values. For details on those features and a detailed description of the query processing, we again refer the reader to [6].

6. USER INTERFACE

For a convincing proof of concept for our interactive semantic search, we have taken great care to implement a fully functional and intuitive user interface. In particular, there is no need for the user to formulate queries in a language like SPARQL. We claim that any user familiar with full-text search will learn how to use Broccoli in a short time, simply by typing a few queries and following the various query suggestions. The user interface is completely written in Java using the Google Web Toolkit⁵.

The introduction and screenshots (Figures 1 and 2) have already provided a foretaste of the capabilities of our user interface. Here is a list of its most important further features:

(UI 1) Search as you type: New suggestions and results with every keystroke. Very importantly, Broccoli’s suggestions for words, classes, instances, and relations are context-sensitive. That is, the displayed suggestions actually lead to hits, and the more / higher-scored hits they lead to, the higher they are ranked.

(UI 2) Pre-select of most likely suggestion: Broccoli knows four kinds of objects: words, classes, instances, and relations. Depending on where you are in the query construction, you get suggestions for several of them. A new user may be overwhelmed to understand the different semantics of the different boxes. For that reason, after every keystroke Broccoli highlights the most meaningful suggestion, which can be selected by simply pressing *Return*.

(UI 3) Visual query representation: At any time, the current query is shown as a tree, with a color code for the various elements that is consistent with the suggestion boxes.

(UI 4) Change of focus / root: A click on any node in the query tree will change the focus of the query suggestions to that node. A double-click on any class or instance node will make that node the root of the tree and re-group and re-rank the results accordingly.

(UI 5) Full history support: The forward and backward buttons of the browser can be used to undo or redo single steps of the query creation process. Furthermore the current URL of the interface can always be used to store its current state or to exchange created queries with others.

(UI 6) Tutorial: Besides some pre-built example queries, the interface also provides a tutorial mode that shows how to create a search query step by step.

7. EXPERIMENTS

7.1 Input data

Our text collection is the text from all documents in the English Wikipedia, obtained via download.wikimedia.org in January 2013. Some dimensions of this collection: 40 GB

⁵<http://code.google.com/webtoolkit>

XML dump, 2.4 billion word occurrences (1.6 billion without stop-words), 285 million recognized entity occurrences and 200 million sentences which we decompose into 418 million contexts.

As ontology we use the latest version of YAGO from October 2009. We manually fixed 92 obvious mistakes in the ontology (for example, the *noble prize* was a *laureate* and hence a *person*), and added the relation *Plant native-in Location* for demonstration purposes. Altogether our variant of YAGO contains 2.6 million entities, 19,124 classes, 60 relations, and 26.6 million facts.

7.2 Pre-processing

We use a UIMA⁶ pipeline to pre-process the Wikipedia XML. The pipeline includes self-written components to parse the Wikipedia markup, tokenize text, parse sentences using SENNA [13], perform entity-recognition and anaphora resolution (see section 3.2), and decompose the sentences (see section 3.3). We want to note that all these components can easily be exchanged. In principle, this allows Broccoli to work with any given text collection and ontology.

The full parse with SENNA was scaled out asynchronously on a cluster of 8 PCs, each equipped with an AMD FX-8150 8-core processor and 16 GB of main memory. A final non-UIMA component writes the binary index which is kept in three separate files. The file for the context lists has a size of 37 GB. The file for the relation lists has a size of 0.5 GB. And the file for the document excerpts has a size of 276 GB, which could easily be reduced to 85 GB by eliminating the redundant and debug information the file currently contains.

7.3 Computing environment

The code for the index building and query processing is written entirely in C++. The code for the query evaluation is written in Perl, Java, C++ and JavaScript. Our pre-processing components are written in C++ or Java. All performance tests were run on a single core of a Dell PowerEdge server with 2 Intel Xeon 2.6 GHz processors, 96 GB of main memory, and 6x900 GB SAS hard disks configured as Raid-5.

7.4 Query times

For detailed experiments on query times, we refer to the paper describing the index behind Broccoli [6]. In said paper, we have evaluated our system on 8,000 queries of different complexity and 35,000 suggestions. Therefore we here omit a detailed breakdown and limit ourselves to the figures reported in Table 1.

Query set	average	median	90%ile	99%ile
Hit queries	52ms	23ms	139ms	393ms
Suggestion	19ms	6ms	44ms	193ms

Table 1: Statistics of query times over 8,000 queries and 35,000 suggestions.

On our collection, 90% of the queries finish within 140ms, 99% within 400ms. Suggestions are even faster. The breakdown in [6] shows that for a combination of Wikipedia and YAGO, only queries that include text take significant time. Purely ontological queries finish within 2ms on average.

⁶<http://uima.apache.org/>

		#FP	#FN	Precision	Recall	F1	P@10	R-Prec	MAP	nDCG
SemSearch	sections	44,117	92	0.06	0.78	0.09	0.32	0.42	0.44	0.45
	sentences	1,361	119	0.29	0.75	0.35	0.32	0.50	0.49	0.50
	contexts	676	139	0.39	0.67	0.43[†]	0.25	0.52	0.45	0.48
Wikipedia lists	sections	28,812	354	0.13	0.84	0.21	0.46	0.38	0.33	0.41
	sentences	1,758	266	0.49	0.79	0.58	0.82	0.65	0.59	0.68
	contexts	931	392	0.61	0.73	0.64[*]	0.84	0.70	0.57	0.69
TREC	sections	6,890	19	0.05	0.82	0.08	0.28	0.29	0.29	0.33
	sentences	392	38	0.39	0.65	0.37	0.58	0.62	0.46	0.52
	contexts	297	36	0.45	0.67	0.46[*]	0.58	0.62	0.46	0.55

Table 2: Sum of false-positives and false-negatives and averages for other measures over all SemSearch, Wikipedia list and TREC queries for Broccoli when running on sections, sentences or contexts. For contexts, the results for the SemSearch and Wikipedia list benchmarks can be reproduced using our web application at <http://broccoli.informatik.uni-freiburg.de/repro-corr/>. *, † denotes a p-value < 0.02, < 0.003 for the two-tailed t-test against the sentences baseline.

7.5 Result quality

We performed an extensive quality evaluation using topics and relevance judgments from several standard benchmarking tasks for entity retrieval: the Yahoo SemSearch 2011 List Search Track [21], the TREC 2009 Entity Track [4] and, similarly as in [7], a random selection of ten Wikipedia featured *List of ...* pages. To allow reproducibility we provide queries and relevance judgments as well as the possibility to evaluate (and modify) the queries against a live running system for the SemSearch List Track and the Wikipedia lists at <http://broccoli.informatik.uni-freiburg.de/repro-corr/>. The TREC Entity Track queries were used for an in-depth quality evaluation that does not allow for an easy reproduction. Therefore we do not provide them in our reproducibility web application. In the following we first describe each of the tasks in more detail.

The SemSearch 2011 List Search Track consisted of 50 topics asking for lists of entities in natural language, e.g. *Apollo astronauts who walked on the Moon*. The publicly available results were created by pooling the results of participating systems and are partly incomplete. Furthermore, the task used a subset of the Billion Triple Challenge Linked Data as collection, and some of the results referenced the same entity several times, e.g. once in DBPedia and once in OpenCyc. Therefore, we manually created a new ground truth consisting of Wikipedia entities. This is possible because most topics were inspired by Wikipedia lists and can be answered completely by manual investigation. Three of the topics did not contain any result entities in Wikipedia, and we ignored one additional topic because it was too controversial to answer with certainty (*books of the Jewish canon*). This leaves us with 46 topics and a total of 384 corresponding entities in our ground truth⁷. The original relevance judgments only had 42 topics with primary results and 454 corresponding entities, including many duplicates.

The TREC 2009 Entity Track worked with the ClueWeb09 collection and consisted of 20 topics also asking for lists of entities in natural language, e.g. *Airlines that currently use Boeing 747 planes*, but in addition provided the source entity (*Boeing 747*) and the type of the target entity (*organization*). We removed all relevance judgments for pages that were not contained in the English Wikipedia; this approach

⁷ available at <http://broccoli.informatik.uni-freiburg.de/repro-corr/>

was taken before in [11] as well. This leaves us with 15 topics and a total of 140 corresponding relevance judgments.

As third benchmark we took a random selection of ten of Wikipedia’s over 2,400 manually compiled featured en.wikipedia.org/wiki/List_of_..._pages⁸, e.g. the *List of participating nations at the Winter Olympic Games*. Wikipedia lists are manually compiled by humans, but actually they are answers to semantic queries, and therefore perfectly suited for a system like ours. In addition, the featured Wikipedia lists undergo a review process in the community, based on, besides other attributes, comprehensiveness. For our ground truth, we automatically extracted the list of entities from the Wikipedia list pages. This leaves us with 10 topics and a total of 2,367 corresponding entities in our ground truth⁷.

For all of these tasks we manually generated queries in our query language corresponding to the semantics of the topics. We relied on using the interactive query suggestions of our user interface, but did not fine-tune our queries towards the results. An automatic translation from natural language to our query language is part of future work (see section 8). We want to stress that our goal is not a direct comparison to systems that participated in the tasks above. For that, input, collection and relevance judgments would have to be perfectly identical. Instead, we want to show that our system allows to construct intuitive queries that provide high quality results for these tasks.

We first evaluated the impact of our context decomposition from Section 3.3 (*contexts*) on result quality, by comparing it against two simple baselines: taking each sentence as one context (*sentences*) and taking each section as one context (*sections*). Table 2 shows that compared to sentences, our contexts decrease the (large) number of false-positives significantly for all benchmarks. For the TREC benchmark even the number of false-negatives decreases. This is the case because our document parser pre-processes Wikipedia lists by appending each list item to the preceding sentence (before the SCI+SCR phase). These are the only types of contexts that cross sentence boundaries and a rare exception. For the Wikipedia list benchmark we verified that this technique did not cause any results that are in the lists from which we created the ground truth. Since the sentence level

⁸http://en.wikipedia.org/wiki/Wikipedia:Featured_lists

does not represent a true superset of our contexts we also evaluated on the section level. We can observe a decrease in the number of false-negatives (a lot of them due to random co-occurrence of query words in a section) which does not outweigh the drastic increase of the number of false-positives. Overall, context decomposition results in a significantly increased precision and F-Measure, which confirms the positive impact on the user experience that we have observed.

Considering the ranking related measures in Table 2 we see a varying influence for the context based approach. The number of cases where ranking quality improves, remains unchanged or decreases is roughly balanced. This looks surprising, especially since the increase in F-measure is significant, but the reason is simple. So far our system uses simplistic ranks, determined by mere term frequency. We plan to improve on that in the future; see Section 8. We want to stress the following though. Most semantic queries, including all from the TREC and SemSearch benchmark, have a small set of relevant results. We believe that for such queries the quality of the result set as a whole is more important than the ranking within the result set. Still, for the TREC benchmark, R-precision on contexts is 0.62 and, for the SemSearch benchmark, mean average precision is 0.45. The best run from the TREC 2009 Entity Track when restricted to the English Wikipedia had an R-precision of 0.55 as reported in [11, Table 10]. The best result for the SemSearch List Search Track was a mean average precision of 0.279 [3]. Again, these results cannot be compared directly, but they do provide an indication of the quality and potential of our system.

7.6 Error analysis

To identify areas where our system can be improved we manually investigated the reasons for the false-positives and false-negatives when using contexts. We used the TREC benchmark for this, because it has a reasonable number of queries and relevance judgments that still allow a costly manual inspection of the results. We defined the following error categories. For false-positives: (FP1) a true hit which was *missing* from the ground truth; (FP2) the words in the context have a *different meaning* than what was intended by the query; (FP3) due to an error in the *ontology*; (FP4) a mistake in the *entity recognition*; (FP5) a mistake by the *parser*. (FP6) a mistake in our *context decomposition*. For false-negatives: (FN1) there seems to be *no evidence* for this entity in the Wikipedia based on the query we used. It is possible that the fact is present but expressed differently, e.g., by the use of synonyms of our query words; (FN2) the query elements are *spread* over two or more sentences; (FN3) a mistake in the *ontology*; (FN4) a mistake in the *entity recognition*; (FN5) a mistake by the *parser*; (FN6) a mistake in our *context decomposition*.

#FP	FP1	FP2	FP3	FP4	FP5	FP6
297	55%	11%	5%	12%	16%	1%

#FN	FN1	FN2	FN3	FN4	FN5	FN6
36	22%	6%	26%	21%	16%	8%

Table 3: Breakdown of errors by category.

Table 3 provides the percentage of errors in each of these categories. The high number in FP1 is great news for us: many entities are missing from the ground truth but were found by Broccoli. Errors in FN1 occur when full-text search with our queries on whole Wikipedia documents does not yield hits, independent from our contexts. Tuning queries or adding support for synonyms can decrease this number. FP2 and FN2 comprise the most severe errors. They contain false-positives that still match all query parts in the same context but have a different meaning and false-negatives that are lost because contexts are confined to sentence boundaries. Fortunately, both numbers are quite small.

The errors in categories FP and FN 3-5 depend on implementation details and third-party components. The high number in FN3 is due to errors in our current ontology, YAGO. A closer inspection revealed that, although the facts in YAGO are reasonably accurate, it is vastly incomplete in many areas (e.g., the *acted-in* relation contains only one actor for most movies). Preliminary experiments suggest that switching to Freebase [10] in the future will solve this and improve the results considerably (see section 8). To mitigate the errors caused by entity recognition and anaphora resolution (FP4+FN4), a more sophisticated state-of-the-art approach is easily integrated. Parse errors are harder. Assuming a perfect constituent parse for every single sentence, especially those with flawed grammar, is not realistic. Still, those errors do not expose limits of our approach. We hope to enable SCI+SCR without a full-parse in the future (see Section 8). The low number of errors due to our context decomposition (FP6+FN6) demonstrates that our current approach (Section 3.3) is already pretty good. Fine-tuning the way we decompose sentences might decrease this number even further.

Naturally, an evaluation should not treat entities missing in the ground-truth in the same way as actual errors. Table 4 provides quality measures for our benchmark based on sentences and contexts under three conditions: (*original*) evaluation based on the original TREC ground-truth; (*+missing*) with the entities from FP1 added to the ground truth; (*+correct*) with the errors leading to FP and FN 3,4,5 corrected.

		F1	P@10	R-Prec	MAP
Sentences	original	0.37	0.58	0.62	0.46
	+missing	0.55	0.77	0.76	0.60
Contexts	original	0.46	0.58	0.62	0.46
	+missing	0.65	0.79	0.77	0.62
	+correct	0.86	0.94	0.92	0.85

Table 4: Quality measures on TREC 2009 queries for three different levels of corrections.

The numbers for *+correct* show the high potential of our system and motivate further work correcting the respective errors. As argued in the discussion after Table 3, many corrections are easily applied, while some of them remain hard to correct perfectly.

8. CONCLUSIONS AND FUTURE WORK

We have presented Broccoli, a search engine for the interactive exploration of combined text and ontology data. We have described the index, the natural language processing,

and the user interface behind Broccoli. And we have provided reproducible evidence that Broccoli is indeed fast and gives search results of good quality.

So far, we have implemented all the basic ideas we deemed necessary to provide a convincing proof of concept. Based on this work, there are a lot of interesting directions for future research.

The underlying ontology plays a major role for our system. By switching from YAGO to Freebase we expect a great improvement of the overall quality through a better coverage of relations and thus proposals and results (see Tables 3 and 4 in the previous section). Our current approaches to entity recognition and anaphora resolution work well, but it might be possible to further improve result quality by incorporating more elaborate state-of-the-art approaches. This would also allow the system to be more easily applied to other collections than Wikipedia (our current heuristics rely on its structure, see Section 3.2). Integrating simple inference heuristics could help to reduce the number of errors that are caused by facts that are spread over several sentences. A high-quality sentence decomposition *without* the need for an expensive and error-prone full parse should further increase result quality. While query times are already low, optimized query processing and clever caching strategies have the potential to further improve speed. To investigate how to best approach performance and quality improvements, an evaluation of Broccoli on a larger, web-like collection should provide valuable insights. Automatically transforming natural language queries into our query language could help users that are accustomed to keyword queries in constructing their queries. Finally, a user study of our UI and the whole system is an important next step.

Acknowledgments

This work is partially supported by the DFG priority program Algorithm Engineering (SPP 1307) and by the German National Library of Medicine (ZB MED).

9. REFERENCES

- [1] E. Agichtein and L. Gravano. *Snowball*: extracting relations from large plain-text collections. In *ACM DL*, pages 85–94, 2000.
- [2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 722–735, 2007.
- [3] K. Balog, M. Ciglan, R. Neumayer, W. Wei, and K. Nørnvåg. Ntnu at semsearch 2011. In *Proc. of the 4th Intl. Semantic Search Workshop*, 2011.
- [4] K. Balog, A. P. de Vries, P. Serdyukov, P. Thomas, and T. Westerveld. Overview of the TREC 2009 Entity Track. In *TREC*, 2009.
- [5] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, pages 2670–2676, 2007.
- [6] H. Bast and B. Buchhold. An index for efficient semantic full-text search. In *CIKM*, 2013.
- [7] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber. Ester: efficient search on text, entities, and relations. In *SIGIR*, pages 671–678, 2007.
- [8] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In *ESWC*, pages 554–568, 2008.
- [9] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
- [10] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250, 2008.
- [11] M. Bron, K. Balog, and M. de Rijke. Ranking related entities: components and analyses. In *CIKM*, pages 1079–1088, 2010.
- [12] R. Chandrasekar, C. Doran, and B. Srinivas. Motivations and methods for text simplification. In *COLING*, pages 1041–1044, 1996.
- [13] R. Collobert. Deep learning for efficient discriminative parsing. *Journal of Machine Learning Research - Proceedings Track*, 15:224–232, 2011.
- [14] G. Giannopoulos, N. Bikakis, T. Dalamagas, and T. K. Sellis. Gontogle: A tool for semantic annotation and search. In *ESWC*, pages 376–380, 2010.
- [15] F. Giunchiglia, U. Kharkevich, and I. Zaihrayeu. Concept search. In *ESWC*, pages 429–444, 2009.
- [16] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *BIS*, pages 1–11, 2010.
- [17] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.
- [18] J. Pound, P. Mika, and H. Zaragoza. Ad-hoc object retrieval in the web of data. In *WWW*, pages 771–780, 2010.
- [19] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.
- [20] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3):203–217, 2008.
- [21] T. Tran, P. Mika, H. Wang, and M. Grobelnik. Semsearch’11: the 4th semantic search workshop. In *WWW (Companion Volume)*, 2011.
- [22] H. Wang, Q. Liu, T. Penin, L. Fu, L. Zhang, T. Tran, Y. Yu, and Y. Pan. Semplore: A scalable IR approach to search the web of data. *J. Web Sem.*, 7(3):177–188, 2009.
- [23] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries - incremental query construction on the semantic web. *J. Web Sem.*, 7(3):166–176, 2009.