Semantic Full-Text Search with ESTER: Scalable, Easy, Fast

Holger Bast Fabian Suchanek Ingmar Weber
Max-Planck-Institut für Informatik
Saarbrücken, Germany
{bast,suchanek,iweber}@mpi-inf.mpg.de

Abstract

We present a demo of ESTER, a search engine that combines the ease of use, speed and scalability of full-text search with the powerful semantic capabilities of ontologies. ESTER supports full-text queries, ontological queries and combinations of these, yet its interface is as easy as can be: A standard search field with semantic information provided interactively as one types. ESTER works by reducing all queries to two basic operations: prefix search and join, which can be implemented very efficiently in terms of both processing time and index space. We demonstrate the capabilities of ESTER on a combination of the English Wikipedia with the Yago ontology, with response times below 100 milliseconds for most queries, and an index size of about 4 GB. The system can be run both stand-alone and as a Web application.

1 Introduction

The prevailing paradigm in search today is ranked keyword retrieval: the user types in a few query words, and the engine returns a list of documents, ordered by estimated relevance to the user's intent. For many kinds of information needs this has proven to give very satisfactory results. There are obvious limits to this approach, however, when it comes to queries that require an understanding of the actual *meaning* of words in a query or document and how they relate to each other.

For example, assume we are looking for politicians who have had an audience with the pope. Then the keyword query `audience pope politician` is likely to give us disappointing results, because relevant documents can be expected to contain the words `audience` and `pope`, but not the literal word `politician`; rather we would expect them to contain *instances of that class*.

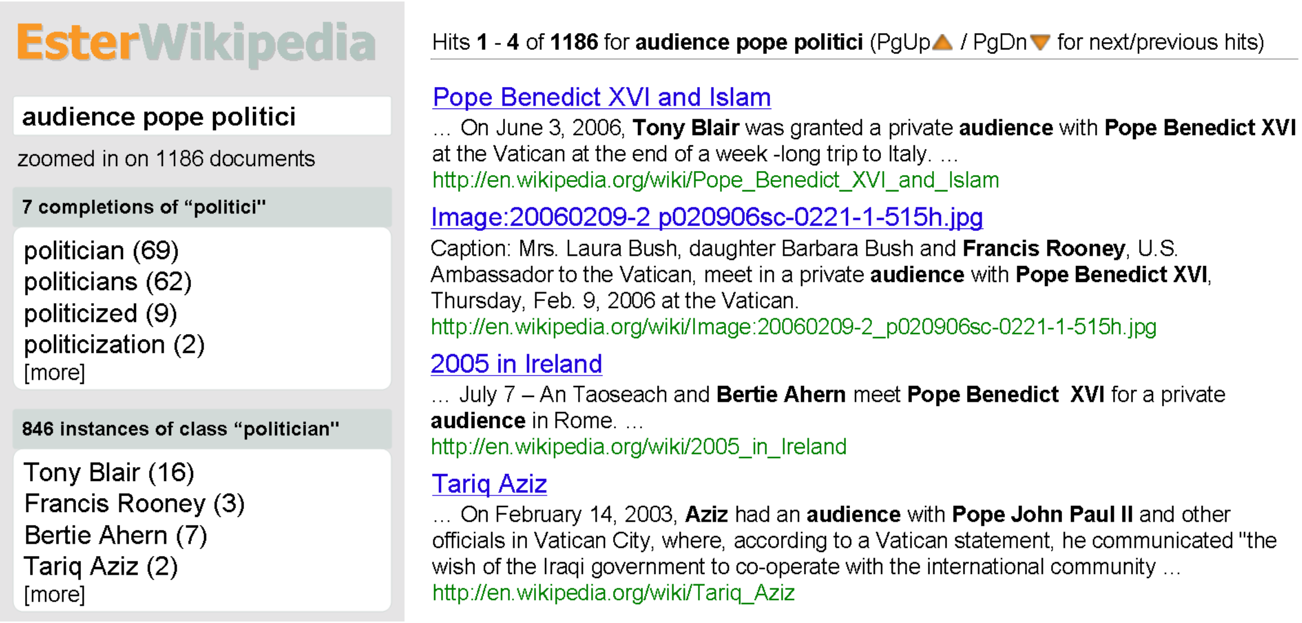
A classical approach to deal with such problems is to represent the knowledge more explicitly, in an ontology, which, for the purposes of this paper, we take to be a list of subject - predicate - object triples, for example: *Tony Blair - is a - politician*, *Tony Blair - born in - 1953*, or *Tony Blair - has nationality - British*. Now, potentially all queries for combinations of these facts can be answered, for example: *Which british politicians were born in the 1950s?*

Both approaches have complementary strengths and weaknesses: Full texts are the natural way to capture human knowledge without loss, and permit surprisingly effective retrieval without requiring any understanding on part of the search engine for many queries, but not all. Ontologies, on the other hand, represent human knowledge explicitly in a form suitable for automated processing, but they are hard to come down with and bound to be incomplete. For example, there is hardly an existing ontology that would contain information on which politicians had audiences with which popes.

Our engine, ESTER, therefore provides both full-text and ontology search, integrated in a seamless fashion. For example, ESTER allows the user to type in a query like `audience pope politician`, and will then figure out automatically that `politician` is a semantic class, and display instances leading to good hits, as well as a selection of such hits. In the next section, we will describe what happens behind the scenes for that query. Figure 1 provides for a screenshot of our engine in action for that query.

There are a few other engines and approaches, which also combine full-text search with ontology search [3] [2] [4]. All approaches that we know of are based on a backend for XML queries, which are either processed via an off-the-shelf XPath engine, or via a suitably extended inverted index. In any case, query processing times are at least an order of magnitude slower than for ESTER; see [1]. Also, the other engines lack the capability of doing *joins*, which give ESTER part of its querying power and are key to its efficient query processing.

978-0-7695-3503-6/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDMW.2008.101



audience pope politici

zoomed in on 1186 documents

7 completions of "politici"

politician (69)
politicians (62)
politicized (9)
politicization (2)
[more]

846 instances of class "politician"

Tony Blair (16)
Francis Rooney (3)
Bertie Ahern (7)
Tariq Aziz (2)
[more]

Hits **1** - **4** of **1186** for **audience pope politici** (PgUp▲ / PgDn▼ for next/previous hits)

Pope Benedict XVI and Islam
... On June 3, 2006, **Tony Blair** was granted a private **audience** with **Pope Benedict XVI** at the Vatican at the end of a week -long trip to Italy. ...
http://en.wikipedia.org/wiki/Pope_Benedict_XVI_and_Islam

Image:20060209-2 p020906sc-0221-1-515h.jpg
Caption: Mrs. Laura Bush, daughter Barbara Bush and **Francis Rooney**, U.S. Ambassador to the Vatican, meet in a private **audience** with **Pope Benedict XVI**, Thursday, Feb. 9, 2006 at the Vatican.
http://en.wikipedia.org/wiki/Image:20060209-2_p020906sc-0221-1-515h.jpg

2005 in Ireland
... July 7 – An Taoseach and **Bertie Ahern** meet **Pope Benedict XVI** for a private **audience** in Rome. ...
http://en.wikipedia.org/wiki/2005_in_Ireland

Tariq Aziz
... On February 14, 2003, **Aziz** had an **audience** with **Pope John Paul II** and other officials in Vatican City, where, according to a Vatican statement, he communicated "the wish of the Iraqi government to co-operate with the international community ...
http://en.wikipedia.org/wiki/Tariq_Aziz

Figure 1. A screenshot of our search engine for the query `audience pope politici` searching the English Wikipedia. The list of completions and hits is updated automatically and instantly after each keystroke, hence the absence of any kind of search button. The number in parentheses after each completion is the number of hits that would be obtained for that particular completion. The upper box suggests words and phrases that start with `politici` and that occur together with the word `audience` and either the word `pope` or the mentioning of a pope. The lower box suggests *instances of politicians* with that property. Our ranking gives precedence to occurrences in proximity of each other. Fast processing of this apparently simple query requires the whole complexity of our system in the background: ontological knowledge, entity recognition, prefix queries, and joins; see Section 2. Our interactive and proactive (suggest-as-you-type) user interface hides this complexity from the user as much as possible. See Section 3 for other types of queries which ESTER can handle in a similar fashion.

2 System architecture

ESTER is build from three components: a query engine, an entity recognizer, and a user interface. An application of ESTER takes as input a collection of text documents and an ontology of facts. The task of the entity recognizer is to establish the links between the words and phrases in the documents and the entities in the ontology. Then, the ontology is woven into the documents by adding artificial words into the corpus. As a result, the query engine will deliver results from the ontology, even though it operates just on the corpus. The query engine supports only two basic operations: *prefix search* and *join*, both of which can be supported very efficiently in terms of both processing time and index space. In the following, we will explain this whole process by example; for more details see [1].

We first explain how the ontology is woven into the corpus. To integrate, say, two facts about *Tony Blair* into the documents, the following steps are necessary:

O1. Relate the entity *Tony Blair* to a canonical document from the collection. In the case of Wikipedia, this would be the page titled *Tony Blair*. If no such document exists, simply create one.

O2. To that document, add the following artificial words, where the first column indicates the position within that document[1]:

```
0   entity:tony_blair
0   person:tony_blair
1   is_a:2
1   politician_of:3
2   class:politician
3   country:united_kingdom
```

suffixes :2 and :3 are a technicality needed to find the object of the respective relations later; see query Q5 below.

O3. Add the artificial word

[1] Note that our index allows multiple words at the same position

baseclass:politician:person to a special document, which is used for no other purpose but words of this kind.

O4. For every word or phrase in the corpus that refers to *Tony Blair*, the entity recognizer will add the artificial word person:tony_blair to the index, at the same position of the referring word or phrase.

Now assume that a user has typed our example query audience pope politici . Then, right after the last keystroke, the following prefix search and join queries are launched in the background.

Q1. audience pope politici* The result of this prefix query is a ranked list of all word-in-document pairs (w, d), where w starts with politici and d is a match for audience pope (which itself was launched as a prefix query). This query provides the content of the upper box in Figure 1. There is nothing semantic about this query.

Q2. baseclass:politici* This prefix query tells us whether politici is the prefix of some base class. If more than one base class matched, the lower box from Figure 1 would display the alternatives. For our example query, exactly one base class matches, namely *person*.

Q3. audience pope person:* This prefix query finds all occurrences of *persons* (as found by the entity recognizer) in documents that already matched the query audience pope.

Q4. class:politician - is_a - person:* This prefix query provides all documents that correspond to occurrences of *persons* that are actually *politicians*. Here - is a negative proximity operator, namely x - y matches if y precedes x.

Q5. Finally, the lists of word-in-document pairs from Q3 and Q4 are *joined* over the word component, providing a list of all occurrences of *politicians* in documents that matched the query audience pope.

The astute reader will wonder why we took the detour of mapping the class *politician* to the class *person* instead of looking for occurrences of *politicians* directly. The procedure, as described by example above, is actually key to ESTER's efficiency in both query processing time and index space. If we annotated each occurrence of an entity in our given text collection by all the classes it belongs to, our index would blow up by one or two orders of magnitude. The other extreme would be to mark each recognized entity merely as belonging to class *entity*, and only once in its canonical document list all the classes and relations to which it belongs. Then, however, for Q3 and Q4 above, we would have to complete the prefix entity:* which would be inherently slow, since a constant fraction of all words in the index are completions of that prefix. ESTER therefore chooses the middle path of having a few intermediate base classes such each individual entity belongs to only few of these base classes, and at the same time the number of occurrences of entities from each base class is reasonably bounded. For details, see [1].

3 Supported queries and user interface

It can be shown that arbitrary SPARQL queries can be reduced to few of ESTER's basic prefix search and join operation along the lines of the example above. Thus, it would in principle be possible to provide a SPARQL user interface. The problem with SPARQL, however, is that, like for all SQL derivates (and also most XML query languages), even simple queries demand a non-trivial effort from the user to learn at least the basics of the query language. ESTER therefore provides its functionality in a very lightweight and intuitive user interface that is modelled after the standard search engine input field. That is, the user enters a keyword query just as she would for an ordinary search engine, and the system interactively offers plausible interpretations of this query along with hits. Here is a list of features provided by ESTER in this fashion.

1. **Syntactic completions:** ESTER shows all prefix completions of the last query word in the upper box from Figure 1. While in the example, only single word completions appear, this box can also show words that contain the query word as a subword, as well as phrases, synonyms, and spelling variants.

2. **Semantic completions:** When the last word of the query is the prefix of multiple semantic classes, ESTER displays all possible classes in the lower box from Figure 1. If exactly one semantic class matches, a selection of *instances* from that class is displayed instead, as shown in Figure 1. This is the case at which we had a closer look in the previous section.

For both syntactic and semantic completions, the user can click on a displayed completion and the search will be refined accordingly. There is also an option to view completions of any other of the query words, for example pope from our example query. However, since the engine interactively updates its display after each keystroke, and since queries are typed from left to right, the default mode is to show completions only for the last query word.

3. **Relations:** If the last query word is an entity, ESTER also provides a third box, not shown in Figure 1, that displays information on all semantic relations for that entity. If the entity has exactly one relation or if the last query word matches a relation, a selection of the corresponding objects (from the matching fact triples) is displayed. For example, if in the situation from Figure 1 we clicked on Tony

Blair, the third box would display selected facts with him as a subject, such as `born in 1953` and `politician of United Kingdom`.

3. **Explicit joins:** In Q5, joins were used as part of the procedure to compute the semantic completions, but they are also of use as explicit queries. For example, assume we are looking only for politicians that had an audience with the pope and were involved in some scandal. In ESTER, we could type this as

```
politician[audience pope, scandal]
```

and the result (which would be shown in the lower box of Figure 1) would be a list of politicians which lead to a match with `audience pope` as well as with `scandal`, but *not necessarily in the same document!* Note that this assembly of information across documents is something which neither an ordinary full-text search engine nor an XPath engine can do.

4 Demonstration

The demonstration will allow arbitrary interactive queries on the full text of the English Wikipedia (about 3 million documents) combined with the Yago ontology (about 3 million facts), for which the index consists of about 1.5 billion postings, which are stored in compressed form in no more than 4.1 GB.

All features listed in the previous section are possible. The response time is below 100 milliseconds for most queries, thus giving a true search-as-you-type feeling.

The system is implemented as a Web application, with three components: JavaScript Code running on the client, PHP code running on a web server, and the actual query engine, implemented in C++, that answers the basic prefix search and join queries. All three components can reside on different machines. We will demonstrate a purely local setup running on a standard notebook (under Windows), as well as a setup that works via the Internet with an arbitrary client, and query engine and web server on a dedicated server (under Linux).

5 Difference compared to [1]

This demo is based on [1], which describes the design behind ESTER, analyzes how this leads to a space-efficient index and fast query times, and proves how ESTER can, in principle, answer (almost) arbitrary SPARQL queries.

For this demo, we fully implemented and integrated all the components described in [1], that is: our own fully functional, scalable entity recognizer, an indexer that given a document collection and an ontology produces the special ESTER index, and a fully functional web-based user interface. Our system is built in a way that hides as much as possible of its internal complexity, thus making it very easy to use for indexing and then querying an arbitrary given text collection and ontology. (We will give a live demonstration of the complete index building tool chain.) Getting the user interface to work in a completely seamless and intuitive fashion, as hinted at by the screenshot of Figure 1, was, as usual, an incredible amount of detail work.

Acknowledgments

This work was partially supported by the German Science Foundation (DFG), Priority Programme 1307 (Algorithm Engineering).

References

[1] H. Bast, A. Chitea, F. M. Suchanek, and I. Weber. ESTER: efficient search on text, entities, and relations. In *30th Conference on Research and Development in Information Retrieval (SIGIR'07)*, pages 671–678, 2007.

[2] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML documents via XML fragments. In *26th Annual Conference on Research and Development in Information Retrieval (SIGIR'03)*, pages 151–158, 2003.

[3] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar semantic search: a database approach to information retrieval. In *Conference on Management of Data (SIGMOD'06)*, pages 790–792, 2006.

[4] R. Schenkel, F. M. Suchanek, and G. Kasneci. YAWN: A semantically annotated Wikipedia XML corpus. In *12. Symposium on Database Systems for Business, Technology and the Web of the German Socienty for Computer science (BTW 2007)*, 2007.