Effective and Efficient Entity Search in RDF Data

Roi Blanco[1], Peter Mika[1], and Sebastiano Vigna[2]

[1] Yahoo! Research
Diagonal 177, 08018 Barcelona, Spain
{roi,pmika}@yahoo-inc.com
[2] Università degli Studi di Milano
via Comelico 39/41, I-20135 Milano, Italy
vigna@acm.org

Abstract. Triple stores have long provided RDF storage as well as data access using expressive, formal query languages such as SPARQL. The new end users of the Semantic Web, however, are mostly unaware of SPARQL and overwhelmingly prefer imprecise, informal keyword queries for searching over data. At the same time, the amount of data on the Semantic Web is approaching the limits of the architectures that provide support for the full expressivity of SPARQL. These factors combined have led to an increased interest in semantic search, i.e. access to RDF data using Information Retrieval methods. In this work, we propose a method for effective and efficient entity search over RDF data. We describe an adaptation of the BM25F ranking function for RDF data, and demonstrate that it outperforms other state-of-the-art methods in ranking RDF resources. We also propose a set of new index structures for efficient retrieval and ranking of results. We implement these results using the open-source MG4J framework.

1 Introduction

The amount of data published on the Semantic Web has grown at increasing rates in the past years due to the activities of the Linked Data community and the adoption of RDFa by major web publishers. The amount of data to be managed is stretching the scalability limitations of triple stores that are conventionally used to manage Semantic Web data. At the same time, the Semantic Web is increasingly reaching end users who need efficient and effective access to large subsets of this data. Such end users prefer simple, but ambiguous natural language queries over highly selective, formal graph queries in SPARQL, the query language of triple stores. In a web search scenario, formulating SPARQL queries may not be feasible altogether due to the heterogeneity of data.

These requirements are spurring interest in the field of Semantic Search, in particular the adaptation of Information Retrieval methods to data access. IR-style indexing is efficient in that it scales well with respect to the size of text collections in both index construction and retrieval. The field has also developed a number of methods for effective ranking of documents that match user

L. Aroyo et al. (Eds.): ISWC 2011, Part I, LNCS 7031, pp. 83–97, 2011.

queries. The challenge in Semantic Search is adapting these results in indexing and ranking to exploit the inherent structure and semantics of RDF data, and expanding them to support the user tasks common in RDF retrieval. The most basics of these tasks is entity-search or Ad-hoc Object Retrieval (AOR) as described by [15], i.e. the retrieval of RDF resources that are representation of an entity described in a keyword query.

This problem has direct relevance to the operation of Web search engines, which increasingly incorporate structured data in their search results pages. Figure 1 shows a search result page from Yahoo! Search for the query *vienna, austria*. Besides the ten blue links representing document results, we can see on the left-bar suggestions for points of interest in Vienna. This requires an understanding that this query represents the city of Vienna, and a ranking over the points of interest. Similarly, an information box above the result pages shows relevant travel information such as the current weather and the geographic location of the city. This again requires a decision that travel information might be relevant to this query, and to execute a top-1 query for the most relevant city in the travel database, and to retrieve the location of the city and the current weather.

In this paper, we describe our system for entity search that adapts a state-of-the-art IR ranking model by taking into consideration the structure and semantics of RDF data. We show that this ranking model outperforms in effectiveness all 14 submissions that have been evaluated on the task of entity-search at the Semantic Search workshop in 2010. We also discuss the combination of index structures that allow this system to be efficient even on large and heterogenous datasets collected from the Web.

2 Related Work

Besides the core problem of document retrieval, ranking models from Information Retrieval have been applied in the past to the problem of retrieval over XML [9] and the relational data model [1,8,10]. However, adaptations to the RDF model are relatively new.

The typical way of providing online access to RDF collections is by using triple stores (or quad stores) that implement database-style indexing of the structure of RDF graphs. Triple stores (such as OWLIM[1] and 4store[2]) allow the option to index the text values of literals in an inverted index on the side (e.g. using Lucene), or rely on text-indexing of the underlying DBMSs (such as Oracle[3] and Virtuoso[4]), but these indices are only used for matching (filtering candidate solutions). As SPARQL does not have a built in query language for full text search in literals, this functionality is typically exposed using 'magic predicates'

[1] http://www.ontotext.com/owlim
[2] http://4store.org/
[3] http://www.oracle.com/technetwork/database/options/semantic-tech/index.html
[4] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSIntro

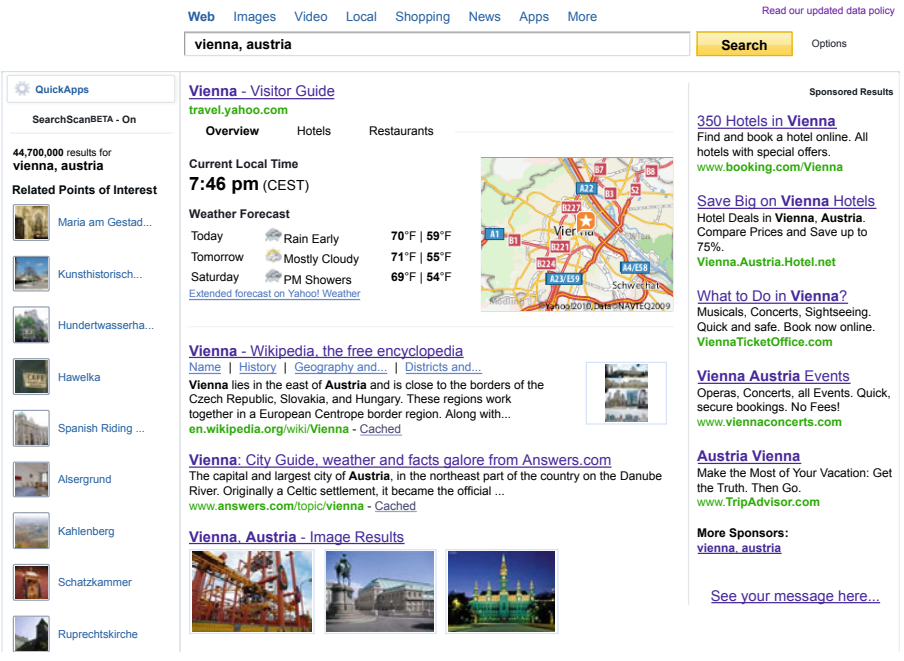

Fig. 1. Examples of structured data in the search result page

that are specific to the triple store. SPARQL 1.0 allows matching using regular expressions, but this is typically not supported by IR engines. Triple stores in general do not perform ranking.

The work of Wang et al. [18] on the Semplore system considers a deeper integration of DB and IR technology, where a set of inverted indices are used for matching limited forms of conjunctive queries, in particular tree-shaped queries with a single target variable at the root of the tree. Such queries may include "keyword concepts", i.e. the set of resources that have a predicate-value that contains a given set of keywords. They show that resolving such queries can be more efficient than the combination of a triple-store with a full-text index on the side. They also propose a simple propagation algorithm to transfer the scores from the keyword matching along the relations back to the root node of the query tree, thereby obtaining a ranking over the results. The relations themselves do not change the scoring.

All of the above systems consider an expert user who is familiar with the structure of the data and is able to denote his information need using a structured query, i.e. providing graph patterns for matching. The scenario we consider in our work is one of ad-hoc retrieval, i.e. retrieval by users who are not assumed to have prior knowledge of the system, including the representation of data. This scenario is typical for open search systems with inexperienced users who

are not aware of the schema of the data, but also for systems that contain heterogeneous collections of data that don't conform to any single schema. As an example, the web dataset considered in this paper consists of over 30,000 unique RDF properties. In such cases, it is impossible to translate the user's information need into a single correct structured representation of the query as suggested by [17].

More specifically, we consider the task of Ad-hoc Object Retrieval (AOR) defined by [15]. Pound et al. point out that over 40% of searches in a typical web search usage are looking for a single object or entity. The task –also commonly called entity search– is thus to provide a ranking over RDF resources in terms of their relevance to an entity that is explicitly named in the query (though the query may contain more information than just the name of the entity). Though this task is basic, it requires solving basic problems in ad-hoc retrieval, in particular, dealing with multiple potential interpretations, and ranking partially matching resources based on their degree of relevance.

This task has been evaluated in a campaign run at the SemSearch 2010 workshop, where six participants entered 14 submissions [7]. The submissions represent a wide range of retrieval approaches, including the ones used in existing Semantic Web search engines such as Sindice [13]. We show that our method outperforms these systems in retrieval effectiveness by as much as 40%.

The method we use is closest in nature to the one proposed by Pérez-Agüera [14]. We also adapt BM25F, a scoring function that is considered state-of-the-art in text retrieval. The index structure used in their system is similar to our horizontal index, but it considers five fields: text (all text from property values), title (words from the URI), object (tokens from the URIs of objects), inlinks (tokens from predicates of 'incoming' triples). The main difference to our work is that BM25F scoring is applied to this five-field structure, while in our work BM25F is used on the vertical index where there is one field per predicate in the data. In other words, while Pérez-Agüera et al. consider all predicates with equal weight, we design a system where it is possible to assign different weights to different predicates. We will show that such weight assignment can improve retrieval performance.

Alternative evaluations exist for other important tasks in Semantic Search. The TREC Entity Track is focusing on entity search over text or hybrid collections (text with metadata).[5] The 1st Workshop on Question Answering over Linked Data (QALD)[6] focused on natural language question-answering over selected RDF datasets, where ranking is not required. The evaluation campaign organized by the European SEALS project focuses on user experience and employs user studies in addition to automated testing [19]. We do not expect that our system could be applied directly in all these scenarios, but some of the techniques described may be useful in designing solutions for them.

[5] http://krisztianbalog.com/files/trec2010-entity-overview.pdf
[6] http://www.sc.cit-ec.uni-bielefeld.de/qald-1

3 Ranking Model

The basis for our ranking is the ranking function BM25F [16] that has been originally developed for text retrieval. It is an extension of the BM25 probabilistic model that weights query terms differently depending on which document *fields* they appear in. Originally, BM25F was employed to weigh occurrences of terms in the *title, body*, or *anchor text* of Web pages, whereas we will break down the description of an RDF resource by the property, and consider as values the literals that appear for each unique datatype-property (see Section 4).

The features that BM25F uses are the field term frequency tf_{si} (number of times term i appears in field s), the field length l_s (number of tokens in the field s) and the field weights v_s. The ranking function does not exploit proximity information or term dependencies.

Using BM25F, a document D is scored against a query Q using a summation over individual scores of query terms $q \in Q$:

$$score^{BM25F}(Q, D) = \sum_{q \in Q} w_i^{BM25F} \tag{1}$$

First, BM25F computes a document length normalization factor as

$$B_s = \left((1 - b_s) + b_s \cdot \frac{l_s}{avl_s}) \right) , \tag{2}$$

where av_l is the average length of field l and b_s is a tunable parameter ($0 \leq b_s \leq 1$) that controls the amount of normalization. Next, BM25F aggregates the weighted term frequencies over all the fields S, normalizing them using B_s as

$$\tilde{tf}_i = \sum_{s=1}^{S} v_s \frac{tf_{si}}{B_s} , \tag{3}$$

and finally these frequencies are normalized using a sigmoid function as

$$w_i^{BM25F} = \frac{\tilde{tf}}{k_1 + \tilde{tf}_i} \cdot w_i^{IDF} , \tag{4}$$

where k_1 is a parameter and w_i^{IDF} is the *inverse document frequency* of term i, calculated as $\log \left(\frac{D - n_i + 0.5}{n_i + 0.5} \right)$ (n_i is the number of documents i occurs in).

The ranking function as described in its most general form requires information of all field lengths (l_s), which is infeasible to index for very large collections. Instead, we use a simplified version of the ranking function where the size of the document D is used as the length of all fields ($l_s = l$). An additional problem of RDF collections is that many objects are very short and are promoted by the normalization component. In order to mitigate this problem, we select a threshold l_{max} so that if $l > l_{max} \rightarrow l = l_{max}$, and set $l_{max} = 10$ for all the experiments.

Standard document retrieval models also allow for incorporating document *query-independent* features, which might come from different sources such as the Web graph. Two examples are the document PageRank values or the number of inlinks that point to a particular Web page. In our case, we classify documents based on their domain into three classes, just like the field weights. We add this document-weights w_D to compute a final retrieval score as [2]:

$$score(Q, D) = w_D \cdot score^{BM25F}(Q, D) \tag{5}$$

4 Indexing

Information Retrieval engines rely on indices for efficient access to the information required for computing scores at query time [11]. Indexing in IR is a basic process of *inversion* (hence the name *inverted index*) in which a document is made accessible by the *term(s)* appearing in the content rather than by some identifier of the document. In more detail, an inverted index provides, for each term that appears in a collection of documents, a *posting list*, that is a list of numbers identifying the documents in which the term appears. The posting lists can be richer, providing, for instance, also the number of occurrences and possibly the exact positions (always expressed as offsets from the start of the document).

Current off-the-shelf retrieval packages allow references to multiple indices or fields within a single query. In addition, state-of-the-art packages provide support for an *alignment* operator. Alignment of queries is useful for *parallel texts*. The need for handling parallel texts comes originally from the area of natural language indexing, e.g. storing part-of-speech information. For example, a text parallel to "Washington won several battles" could be "PERSON VERB - NOUN". Once parallel texts have been indexed, an alignment operator between terms of two different indices returns just the documents in which two terms appear *in the same positions*. For instance, an alignment between "Washington" and "PERSON" would return the document associated to the parallel texts above, but an alignment between "Washington" and "PLACE" would not (even if "Washington" does appear in the document).

This technique is implemented in MG4J [5], an open-source engine for text indexing. MG4J provides, for each query, a *minimal-interval semantics*—a set of regions of text satisfying the query which are incomparable by containment (i.e., no region is contained inside another region). The resulting semantics are an extension of the Clarke–Cormack–Burkowski lattice [6] that handles multiple indices (e.g., title and main text) particularly suited to parallel texts. Indeed, the alignment operator can align any set of regions, and since the set of regions associated to a term is exactly given by the positions in which the term appears, we obtain the alignment of parallel texts we described. Other possibilities are also available, such as operators that are weaker than exact alignment.

These functionality allow two main alternatives to implement structured retrieval.

The first option is illustrated in Table 2 using the sample data shown in Table 1. For simplicity, we will call this a *horizontal index* on the basis that RDF resources are represented using only three fields, one field for the tokens from values, one for the properties and one for the tokens from the subject URI. The token and property indices are aligned in that there is a correspondence between the positions in the token and property fields, i.e. the value in the token field at a given position is (part of) the value for the property written to the same position in the property index. (Note that we write the complete predicate in each position of the property field.) The alignment operator is used to align the matches in the token and property fields where the query specifies a token to match in a particular field.

The second option, which we will call a *vertical index* is shown in Table 3 using the same data. Here we create a field for each property occurring in the data. In this case performing matching on particular properties only requires the ability to restrict matches by field. Positions can be still useful, e.g. to make sure the first and last name are matched as consecutive words. Note that structured retrieval can also be implemented using a single field, e.g. by encoding fields as a post-fix of tokens or storing field information as payload. These alternatives are much less appealing. Post-fixing, for example storing terms like *peter_foaf:name*, leads to an explosion in dictionary size, especially when using a large number of fields. On the other hand, encoding fields as payload makes it inefficient to restrict searches to particular fields.

Table 1. Sample RDF data in Turtle format

```
@prefix foo: <http://example.org/ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
foo:peter foaf:name "peter mika" .
foo:peter foaf:age "32" .
foo:peter vcard:location "barcelona" .
```

Table 2. Horizontal index of the data in Table 1

Field	pos1	pos2	pos3	pos4	pos5
token	peter	mika	32	barcelona	
property	foaf:name	foaf:name	foaf:age	vcard:location	
subject	http	example	org	ns	peter

In previous work [12], we have shown that both of these index structures can be efficiently built in a distributed fashion using a single MapReduce job. Since indexing can be efficiently parallelized, the index building time is linear in the

Table 3. Vertical index of the data in Table 1

Field	pos1	pos2	pos3	pos4	pos5
foaf:name	peter	mika			
foaf:age	32				
vcard:location	barcelona				

Table 4. R-vertical index of the data in Table 1

Field	pos1	pos2	pos3	pos4	pos5
w_{imp}	peter	mika	barcelona		
w_{neut}					
w_{uni}	32				

size of the input given the same number of machines in the cluster, and also linear in the number of machines given the same input (up to the natural limit where the cost of distribution outweighs the cost of indexing). The resulting indices are similar in size for the horizontal and vertical case and a small fraction of the size of the input data. Note that the vertical index alone does not contain all the information we need for ranking, in particular only the horizontal index provides direct access to term frequencies and the document sizes that is used in our ranking. Thus in practice we can either use the horizontal index on its own or use a combination of the vertical and the horizontal index, where the vertical index is used for faster matching, but the horizontal index is also accessed when computing resource scores.

In our current work, we propose a third additional index structure for improved performance. For purposes of ranking, we only need to distinguish fields that have different weights assigned. In our ranking function, we will use three different weight levels for important, neutral, and unimportant properties so that we can index all properties with the same weight using only three fields, instead of the much larger number of fields we build for the regular vertical index. We call this reduced version of the vertical index the *r-vertical* index. Table 4 shows how we would index our sample data using this index structure, assuming that we classify *foaf:name* and *vcard:location* as important, and *foaf:age* as unimportant. The disadvantage of the r-vertical index is the loss in functionality: using this index it is not possible any more to issue queries that explicitly restrict matches to particular properties, e.g. to retrieve resources where the word *peter* matches in *foaf:name* and not in other fields. Note that using the r-vertical index instead of the vertical index does not change the way ranking is performed, it merely provides faster access and therefore speeds up the ranking process. We investigate this next. We refer the reader to [12] for more discussion on how we build these indices, the time spent and the distributed methods used to scale up indexing.

5 Evaluation

5.1 Evaluation of Efficiency

To measure the efficiency of these structures, we index the Billion Triples Challenge 2009 dataset.[7] It contains RDF data collected by various Semantic Web crawlers, and as such the data is highly heterogeneous. It contains 2,680,081 classes and 33,164 properties, and therefore it is unlikely that any user could be aware of the complete structure of the data and compose formal queries to match this structure. The collection contains 1.14 billion quads, which is 249GB of data in uncompressed N-Quads format. The usage of predicates is highly skewed, and fits an exponentially decaying distribution (refer to the webpage for other statistics). Note that the scale of the data justifies the use of distributed indexing, i.e. a single-machine setup would have been much slower in indexing this amount of data.

For indexing, we grouped the quads by subject URI, and considered as virtual documents the quads with the same subject. We subdivided each document into fields by considering each unique predicate as a separate field. We only indexed datatype-properties, i.e. quads with literals in the object position. In case of multiple values for the same subject and predicate, we simply considered the concatenation of values. We performed a minimal processing of values at indexing time, namely we removed stop-words using a list of 389 common English terms and lower-cased terms. We also indexed the subject URIs by replacing delimiters with blank spaces and applying the same processing to the resulting string. The version of the BTC 2009 dataset used in the evaluation does not include blank nodes, i.e. all blank node identifiers have been replaced by URIs. We index this data using all three index structures. For the vertical index, we select the top three-hundred most common datatype-properties for indexing.

In the experiments, we measure the efficiency of retrieval, i.e. the time it takes to process queries including matching and ranking, but not result rendering. We consider two execution modes: *AND* where we require all keywords to be present in a document to be scored and *OR* where only a single term is needed. The former execution mode resembles Web search engines whereas the latter is the mode by default in our ranking model. To show the additional cost of structured retrieval, we also include a plain BM25 run using the *token* index.

We sample 150K queries taken from Yahoo!'s query logs with the restriction that they lead to a click in Wikipedia, in order to ensure there is an entity focus in the user intent. Among those queries 68% are unique and the average query length is 2.2 terms. Table 5 presents the average running times, which converge after a couple of thousand queries are being executed. The Table shows results for the baseline BM25 retrieval, all three index configurations (horizontal, vertical and reduced-vertical) and the two query execution modes.

Our tests show that the vertical approach is about eight time faster than the horizontal approach when queries are executed in AND mode, while it is only slightly faster in OR mode.

[7] `http://vmlion25.deri.ie/`

Table 5. Retrieval efficiency using different index structures and execution modes

	AND mode	OR mode
BM25	46 ms	80 ms
Horizontal	819 ms	847 ms
Vertical	97 ms	780 ms
R-Vertical	48 ms	152 ms

In general, AND queries execute faster than OR queries. In AND mode, it is necessary to compute the intersection of two (or more) posting lists; for OR queries, it is necessary to compute the union. Clearly, in the second case we always need to read the full posting list of each term involved. In the AND case, instead, it is often possible to skip over documents that are not necessary using *skip-pointers* [11]. For instance, when computing the AND of a very common and a very rare term, most of the postings of the very common term are not needed to compute the result, as the rare term doesn't appear there.

The difference between AND or OR execution modes is small in the horizontal case, because the alignment operator dominates execution times. Further, we can see that the r-vertical index is almost as fast as the vertical index. This proves that it is possible to trade-off query expressivity for faster execution times and apply our scoring at execution times comparable to the current state-of-the-art in Web search engines.

5.2 Evaluation of Effectiveness

We evaluate the effectiveness of our ranking using the data set, the queries and the relevance assessments that have been made available as part of the Semantic Search Challenge of 2010 [7]. All of the data has been made publicly available for research use.[8]

The collection used in this evaluation is the Billion Triples Challenge 2009 data set that we have described in Section 4. The query set consists of 92 queries with an entity focus selected from the query logs of Microsoft Live Search and Yahoo! Search (see [7] for details.) We use a proprietary, state-of-the-art spell corrector to fix a small number of user mistakes in the queries and apply the same term-processing as on the collection.

For ranking, we use the ranking function described in Section 3. We classify manually the properties into three classes (important, unimportant and neutral) and assign the same v_s for each class. In principle, we could learn or select a different v_s for each field, but in practice this would lead to an excessive number of parameters. Table 6 shows the list of important and unimportant properties.

Similarly, we do not assign a weight w_D individually to each document, but manually classify a small number of domains into the three classes. Table 7 shows the list of important and unimportant domains, while all other domains are considered neutral. We then set w_D to w_D^i, w_D^u, w_D^n for documents coming

[8] `http://km.aifb.kit.edu/ws/semsearch10/`

from domains classified as important, unimportant and neutral respectively. It is future work to look at how we could automatically learn these lists, i.e. based on the likelihood of the fields matching in relevant documents or domains vs. the likelihood of matching in irrelevant documents or domains. Similarly, we use a single b parameter for all b_s. We choose a separate weight for the *subject* field, which plays a special role as the identifier of the resource. We score the documents after matching in OR execution mode.

Table 6. Manually selected list of important and unimportant properties. URIs are abbreviated using known prefixes provided by the prefix.cc web service

important	dbp:abstract, rdfs:label, rdfs:comment, rss:description, rss:title, skos:prefLabel, akt:family-name, wn:lexicalForm, nie:title
unimportant	dc:date, dc:identifier, dc:language, dc:issued, dc:type, dc:rights, rss:pubDate, dbp:imagesize, dbp:coorDmsProperty, dbo:birthdate, foaf:dateOfBirth,foaf:nick, foaf:aimChatID, foaf:openid, foaf:yahooChatID, georss:point, wgs84:lat, wgs84:long

We use the official relevance assessments for evaluation, which were gathered using Amazon Mechanical Turk and used a three-scale grading for excellent results, fair results and irrelevant results [3]. We report the retrieval performance using Mean Average Precision (MAP) [11] which is more robust to noise perturbations than the P@10 measure [4] and check for statistical significant differences against the baseline using Wilcoxon's signed rank test (significance level set to 0.01).

Table 7. Manually selected list of important and unimportant domains

important	dbpedia.org, netflix.com
unimportant	www.flickr.com, www.vox.com, ex.plode.us

Our results are shown in Table 8. We perform two rounds of parameter tuning, in each round using a linear search over the individual parameter spaces. First, we select a default configuration for the parameters and tune the performance of each one of the features individually and report on their individual contribution to the increase in performance. Next, given the parameter list ordered as displayed in the table, we report the performance increase when adding a new parameter to the model, one at a time. This allows us to determine what is the benefit of adding each parameter over the best configuration found for the model so far.

We report the contribution of each of the features described Section 3. We start with the plain BM25 function with no structure ($v_s = 1$). We then investigate the effect of tuning BM25's b parameter. We then look at the result of assigning field weights other the default $v_s = 1$, in particular the effect of finding an optimal weight for the subject field (v_{sjc}), and for important and unimportant

fields according to Table 6. Last, we look at changing document weights to other than the default $w_D^n = 1$. In particular, we assign a higher weight to documents that are from important domains as given by Table 7, and then decrease the weights of documents from unimportant domains. We omit the results for the k_1 parameter as it has little effect in retrieval performance.

Table 8. Feature importance measured with MAP. Improvements are statistically significant against plain BM25 using Wilcoxon's pairwise sign rank test (p-value < 0.01). The *Individual Features* column computes the improvement of each feature independently, on top of the untuned baseline, whereas the *Combination* column shows cumulative gain as we add features in the listed order, one at a time.

Feature	Description	Individual Features	Combination
BM25	BM25	0.1805	0.1805
b	BM25's b parameter	0.2450 (+35.7%)	0.2450 (+35.7%)
v_{sjc}	weighting for the subject field	0.2279 (+26.26%)	0.2512 (+2.5%)
v_{imp}	weighting for *important* properties	0.2261 (+25.25%)	0.2565(+2.1%)
v_{uni}	weighting for *unimportant* properties	0.2160 (+19.72%)	0.2590 (+1%)
w_D^i	weighting for *important* domains	0.2229 (+23.49%)	0.2730 (+5.4%)
w_D^u	weighting for *unimportant* domains	0.2319 (+28.47%)	0.2754 (+1%)

The first column of results shows that all features are able to improve significantly the baseline, even adding them individually. It is interesting to note that property field weighting (v_{sjc}, v_{imp}, v_{uni}) is able to improve the MAP score by more than 20%. This is a promising result given that we only took a few properties into account, and potentially adding more parameters to the ranking function could boost the performance by a larger margin. Adding query-independent domain-based weights (w_D^i, w_D^u) is also beneficial, despite the fact that we only included a limited number of site domains. This indicates that there is still room for improvement, given enough training data available and further analysis of which fields and properties should be weighted differently.

The second column of the table shows the accumulated improvement when we introduce one parameter at a time in the model. The total improvement using this one-step linear tuning of features is around 53% over the untuned baseline. 35% of the improvement is due to the b parameter, and on top of that, the field and site features are able to boost the performance another 18%, which is an encouraging result. The fact that the document normalization component plays an important role in the performance (controlled by b) goes accordingly to results in document retrieval. This indicates that the model is able to incorporate many different signals and boost up the performance significantly by combining them in a suitable way.

Next, we perform a 2-fold cross validation splitting the query set in two halves in order to determine the performance of the combination of features and what would be the effectiveness of the system in a real search environment, with

Table 9. Cross-validated results comparing our ranking function against the BM25 baseline and the best performing submission at SemSearch 2010 (percentage improvements are relative to SemSearch 2010)

Method	MAP	NDCG
SemSearch'10	0.1909	0.3134
BM25	0.1805 (-6%)	0.3869 (+23%)
BM25F	0.2705 (+42 %)	0.4800 (+52%)

limited training data available. We tune the parameters performance with a linear search and the promising directions algorithm [16] on each one of the halves separately. The algorithm starts with an initial set of parameter values, and performs one independent linear search over each parameter. Then, it selects the vector going from the initial set of parameter values and the best found values, which defines a *promising direction* in the parameter space. The algorithm explores the parameter space over this vector and repeats the whole process until convergence to a local minimum or when a maximum number of iterations is reached. We report the results averaged over the two halves in Table 9 using both the MAP and the NDCG metric, where the latter exploits graded relevance judgments. Our method improves 50% over the BM25 baseline, and 42% over the best run submitted to SemSearch 2010 using MAP [7]. These results are extremely significant and would necessarily translate to a qualitative jump in user experience.

Looking at the results in more detail, we could conclude that we did poorly on long queries such as *the morning call lehigh valley pa*. We also did poorly on queries with only one relevant result that we didn't find such as *kaz vaporizer*, in this case because the single result came from ex.plode.us domain which we marked as unimportant due to poor quality data (a flat list of tags).[9] We also performed low on the query *hospice of cincinnati*, which is a long-term care provider in Cincinnati that has no directly relevant resource in the BTC dataset. In this case, our system favored blog posts from RSS feeds that mentioned all three words and in general talked about hospice care in Cincinnati. However, the assessors marked as fair results other institutes in Cincinnati, such as the University of Cincinnati, the Hyde Park in Cincinnati and the Cincinnati Police Department. Conversely, we did well on queries that were short but highly selective such as *mst3000*, which stands for Mystery Science Theater 3000, an American cult television comedy. We also did well on queries where there was only one relevant result that we did manage to find, e.g. *fitzgerald auto mall chambersburg pa*. This auto mall has no relevant information in the BTC dataset, but the City of Chambersburg, Pennsylvania was accepted as a fair result by the assessors. All other queries fell in between these extremes, and typically had more than one relevant result.

[9] `http://ex.plode.us` is a social aggregator that is not in service any more.

6 Conclusions

Ad-hoc object retrieval is one of the most basic tasks in semantic search and it has direct applications in search engines that incorporate structured data in their result pages. In this paper, we have proposed an adaptation of the BM25F ranking function to the RDF data model that incorporates both field weights, document priors and a separate field for the subject URIs. We have shown that each of these features contributes to effectiveness on its own and in combination with other features. In cross-validation, the combination of these features outperforms in effectiveness the baseline BM25 method that ignores RDF structure and semantics by 50% in MAP score. It also improves on other state-of-the-art methods on the ad-hoc object retrieval task by 42% in MAP and 52% in NDCG scores.

We have shown two basic index structures, which we called the horizontal and vertical indices, for efficient retrieval of the information required for scoring. Both provide the same query expressivity, but represent different trade-offs in effectiveness. The vertical index becomes ineffective as the number of properties grow, while the horizontal index is able to capture all our data, but requires the slower alignment operator to resolve queries. We also proposed a modified version of the vertical index, which groups properties with the same weight, and thereby trades off query expressivity for a performance that is comparable to retrieval over text. In previous work, we have shown that both basic structures can be efficiently built using MapReduce.

In future work, we plan to explore the combination of retrieval with data integration to reduce the redundancy in current object search results. For this, we need to find co-referent objects in search results and integrate the information that different sources provide. A second problem we would like to address is the ranking of information that is provided about each object. As some objects may have several hundreds of triples associated with them, it is necessary to select only those triples for display that are most descriptive of the object and at the same time pertinent to the user query.

References

1. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword Searching and Browsing in Databases using BANKS. In: ICDE, pp. 431–440 (2002)
2. Blanco, R., Barreiro, Á.: Probabilistic Document Length Priors for Language Models. In: Macdonald, C., Ounis, I., Plachouras, V., Ruthven, I., White, R.W. (eds.) ECIR 2008. LNCS, vol. 4956, pp. 394–405. Springer, Heidelberg (2008), http://portal.acm.org/citation.cfm?id=1793274.1793322
3. Blanco, R., Halpin, H., Herzig, D.M., Mika, P., Pound, J., Thompson, H.S., Tran, D.T.: Repeatable and reliable search system evaluation using crowdsourcing. In: Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR, ACM (2011)
4. Blanco, R., Zaragoza, H.: Beware of relatively large but meaningless improvements. Yahoo! Research Technical Report (2011)

5. Boldi, P., Vigna, S.: MG4J at TREC 2005. In: Voorhees, E.M., Buckland, L.P. (eds.) The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings. No. SP 500-266 in Special Publications, NIST (2005), http://mg4j.dsi.unimi.it/
6. Clarke, C.L.A., Cormack, G.V., Burkowski, F.J.: An algebra for structured text search and a framework for its implementation. The Computer Journal 38(1), 43–56 (1995), http://comjnl.oxfordjournals.org/content/38/1/43.abstract
7. Halpin, H., Herzig, D., Mika, P., Blanco, R., Pound, J., Thompon, H., Duc, T.T.: Evaluating ad-hoc object retrieval. In: Proceedings of IWEST (2010)
8. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword Search in Relational Databases. In: VLDB, pp. 670–681 (2002)
9. Kamps, J., Geva, S., Trotman, A., Woodley, A., Koolen, M.: Overview of the Inex 2008 Ad Hoc Track. In: Geva, S., Kamps, J., Trotman, A. (eds.) INEX 2008. LNCS, vol. 5631, pp. 1–28. Springer, Heidelberg (2009)
10. Luo, Y., Wang, W., Lin, X.: SPARK: A Keyword Search Engine on Relational Databases. In: ICDE, pp. 1552–1555 (2008)
11. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
12. Mika, P.: Distributed indexing for semantic search. In: SEMSEARCH 2010 Proceedings of the 3rd International Semantic Search Workshop, pp. 1–4. ACM (2010), http://portal.acm.org/citation.cfm?id=1863879.1863882
13. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: {A} Document-oriented Lookup Index for Open Linked Data. International Journal of Metadata, Semantics and Ontologies 3(1) (2008), http://www.sindice.com/pdf/sindice-ijmso2008.pdf
14. Pérez-Agüera, J.R., Arroyo, J., Greenberg, J., Iglesias, J.P., Fresno, V.: Using BM25F for semantic search. In: Proceedings of the 3rd International Semantic Search Workshop on - SEMSEARCH 2010, pp. 1–8. ACM Press, New York (2010), http://portal.acm.org/citation.cfm?doid=1863879.1863881, http://km.aifb.kit.edu/ws/semsearch10/Files/bm25f.pdf
15. Pound, J., Mika, P., Zaragoza, H.: Ad-hoc Object Ranking in the Web of Data. In: Proceedings of the WWW, pp. 771–780. Raleigh, USA (2010)
16. Robertson, S., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond, foundations and trends in information retrieval. Foundations and Trends in Information Retrieval 3(4), 333–389 (2009), http://dx.doi.org/10.1561/1500000019
17. Tran, T., Wang, H., Haase, P.: Hermes: Data Web search on a pay-as-you-go integration infrastructure. Web Semantics: Science, Services and Agents on the World Wide Web 7(3), 189–203 (2009), http://linkinghub.elsevier.com/retrieve/pii/S1570826809000213
18. Wang, H., Liu, Q., Penin, T., Fu, L., Zhang, L., Tran, T., Yu, Y., Pan, Y.: Semplore: A scalable IR approach to search the Web of Data. Web Semantics: Science, Services and Agents on the World Wide Web 7(3), 177–188 (2009), http://www.sciencedirect.com/science/article/B758F-X1SBDK-1/2/8efe2a494e75791c8b333a1abdfc4188
19. Wrigley, S.N., Reinhard, D., Elbedweihy, K., Bernstein, A., Ciravegna, F.: Methodology and campaign design for the evaluation of semantic search tools. In: Proceedings of the 3rd International Semantic Search Workshop on - SEMSEARCH 2010, pp. 1–10. ACM Press, New York (2010), http://portal.acm.org/citation.cfm?doid=1863879.1863889