LOTUS:
Adaptive Text Search for Big Linked Data

Filip Ilievski, Wouter Beek, Marieke van Erp,
Laurens Rietveld and Stefan Schlobach

The Network Institute
VU University Amsterdam
{f.ilievski,w.g.j.beek,marieke.van.erp,
l.j.rietveld,k.s.schlobach}@vu.nl

Abstract. Finding relevant resources on the Semantic Web today is a dirty job: there is no centralized lookup service and the support for natural language lookup is limited. In this paper, we present LOTUS: Linked Open Text UnleaShed, a natural language entry point to a massive subset of today's Linked Open Data Cloud. While a wide array of matching and ranking algorithms have been studied, there is no ultimate combination of matching and ranking that will work for every use case. LOTUS recognizes the case-dependent nature of resource retrieval by allowing users to choose from a wide palette of well-known matching and ranking algorithms. LOTUS is an adaptive toolkit that allows user to easily construct the form of resource retrieval that suits her use case best. In this paper, we explain the LOTUS approach, its implementation and the functionality it provides. We also demonstrate the ease with which LOTUS allows Linked Data to be queried at an unprecedented scale in different concrete and domain-specific scenarios. Finally we demonstrate the scalability of LOTUS with respect to the LOD Laundromat data, the biggest collection of easily accessible Linked Data currently available.

Keywords: Findability, Text Indexing, Semantic Search, Scalable Data Management, User-Driven, LOD Laundromat

1 Introduction

A wealth of information is potentially available from Linked Open Data sources such as those found in the LOD Cloud[1] or LOD Laundromat [3]. However, finding relevant resources on the Semantic Web today is not an easy job: there is no centralized query service and the support for natural language look-up is limited. A resource is typically 'found' by memorizing its resource-denoting IRI. The lack of a Web-wide access point to resources through a flexible text index is a serious obstacle for Linked Data consumption.

[1] http://lod-cloud.net/

In this paper, we present LOTUS: Linked Open Text UnleaShed,[2] a natural language entry point to a large subset of today's Linked Open Data Cloud. Text search on the LOD Cloud is not a new phenomenon as Sindice[3] and LOD Cache[4] show. However, LOTUS differs from these existing approaches in three ways: 1) its scale (its index is about 100x bigger than Sindice's index), 2) the adaptability of its algorithms and data collection, and 3) its integration with a novel Linked Data publishing and consumption ecosystem that does not depend on IRI dereferenceability. sem LOTUS indexes every natural language literal from the LOD Laundromat data collection, a large subset of today's LOD CLoud that spans tens of billions of ground statements. The task of resource retrieval is a two-part process that includes both matching and ranking. Since there is no single combination of matching and ranking that is optimal for every use case, LOTUS enables the user to choose her own combination of matching and ranking. In this paper, we present the LOTUS framework, its API, its approach towards the customization of Semantic Web resource retrieval, and its initial collection of matching and ranking algorithms.

The flexibility of the LOTUS approach towards resource retrieval makes it attractive for a wide range of use cases such as Information Retrieval and Text Analysis. For instance, existing Entity Linking systems such as DBpedia Spotlight [11], Babelfy [12] and NERD tools [15]), rely on a single or limited set of knowledge sources, typically DBpedia, and thus suffer from limited coverage. LOTUS could inspire new research ideas for Entity Linking and facilitate Web of Data-wide search and linking of entities.

The remainder of this paper is structured as follows. In Section 2, we detail the problem of performing linguistic search on the LOD Cloud. In Section 3, related work is presented. In Section 4, we describe the LOTUS framework, followed by its implementation in Section 5. We then present some scalability tests and typical usage scenarios of LOTUS in Section 6. We conclude by discussing the key strengths, limitations and future plans for LOTUS in Section 7.

2 Problem description

How do we find relevant resources on the Semantic Web today? RDF resources can be denoted by blank nodes, IRIs and literals. The Semantic Web currently relies on the following findability strategies: datadumps, IRI dereferencing and SPARQL query endpoints. We discuss these approaches in turn.

Datadumps Datadumps implement a rather simple way of finding resource-denoting terms: if one knows the Web address of a datadump then one can use that address to download the full contents of the file. This exposes all the blank nodes, literals and IRIs that appear in that file (the solid arrows in Figure

[2] An early version of LOTUS was presented at COLD 2015, an unarchived workshop [10]. This paper is a significantly extended and updated version of that work.

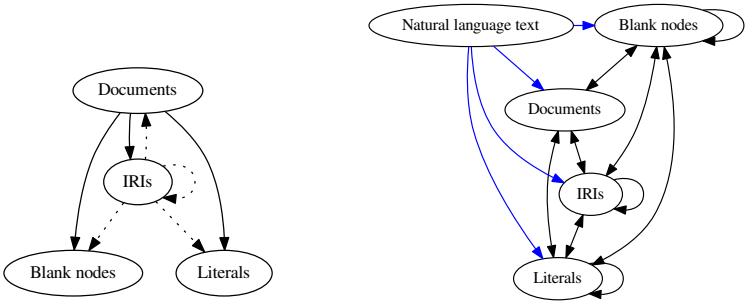
[3] `http://www.sindice.com/`, discontinued in 2014.

[4] `http://lod.openlinksw.com/`

1a). Datadumps do not allow a specific resource to be found and do not link to assertions that are about the same resource but that are published by other sources.

Dereference An IRI dereferences to a set of statements in which that IRI appears in the subject position and, optionally, statements in which that IRI appears in the object position. Which expressions belong to the dereference result set of a given IRI is decided by the *authority* of that IRI, i.e., the person or organization that pays for the domain that appears in the IRI's authority component (the dotted arrows in Figure 1a). Non-authoritative expressions about the resource denoted by that IRI cannot be found directly. This includes statements in which the same IRI appears in the subject position but that are hosted by another authority at another server. In fact, non-authoritative statements can only be found accidentally by navigating the interconnected graph of dereferencing IRIs. Since blank nodes do not dereference (no arrow from blank nodes to IRIs in Figure 1a) significant parts of the graph cannot be traversed. This problem is not merely theoretical since 7% of all RDF terms are blank nodes [9]. In practice this means that non-authoritative assertions are generally not findable.

Since only IRIs can be dereferenced, natural language access to the Semantic Web cannot be gained at all through dereferencing. For instance, it is not possible to find a resource-denoting IRI based on words that appear in RDF literals to which it is related. It is also not possible to search for a resource-denoting IRI based on keywords that only bear close similarity to (some of the) literals to which the IRI is related.



(a) Semantic Web (b) LOTUS+*Frank*

Fig. 1. Graph navigation on the Semantic Web using the standardized notion of IRI dereferencing. A dotted arrow means that only authoritative data can be retrieved. This figure shows the ideal situation in which all IRIs can be dereferences and all dereferencing servers are available. Figure b shows graph navigation on the Semantic Web using LOTUS (blue arrows) in combination with *Frank* (black arrows).

SPARQL endpoints SPARQL [8] is the Semantic Web query language. Compared to IRI dereferencing a SPARQL endpoints provides a far more powerful approach towards finding a resource-denoting term. For instance, SPARQL al-

lows resources to be found based on text search that matches literal terms. As for the findability of non-authoritative expressions about a resource, SPARQL have largely the same problems as the dereferenceability approach. While it is possible to evaluate a SPARQL query over multiple datasets, these datasets have to be included explicitly by using the `SERVICE` keyword [14]. This means that an endpoint that disseminates non-authoritative statements can only be included if its Web address is known beforehand, 'solving' the findability problem by shifting it. The ideal situation would be to always query all SPARQL endpoints. Besides requiring a lenghty list of `SERVICE` keywords this would result in bad performance since the slowest SPARQL endpoint would determine the response time of the entire query. In addition to the unpracticality of finding all non-authoritative endpoints that say something about a resource-denoting term there is also no guarantee that all statements are disseminated by some SPARQL endpoint. Empirical studies show that the number of known SPARQL endpoint with acceptable availability is also rather low (approximately 125) [4].

The SPARQL query language is largely oriented towards matching graph patterns and datatyped values. While the SPARQL specification defines versatile Regular Expression-based operations on the lexical forms of literals, it does not include string similarity matches or other more advanced NLP functionalities.

Requirements Summarizing, the task of finding a resource and statements about it is a big problem on today's Semantic Web. Moreover, this problem will not be solved by implementing existing approaches or standards in a better way, but requires a completely novel approach instead. Based on the above considerations, we specify the following list of requirements for a sufficient solution to the problem of finding Semantic Web resources:

1. Resource findability should not depend on whether an IRI can be dereferenced or on SPARQL endpoint availability.
2. Authoritative and non-authoritative statements should both be findable.
3. The data that can be searched should include tens of billions of ground statements and should include thousands of datasets.
4. Resource-denoting IRIs should be findable based on text-based search that matches (parts of) literals that are asserted about that IRI, possibly by multiple sources.
5. Text based search should include, but should not be limited to, Regular Expression-based matching.
6. The search API must be usable for humans (Web UI) and machines (REST) alike and must be freely available online.
7. Search results should be ranked according to a flexible collection of rankings. Which rankings are used should be customizable in order to support a wide range of use cases.

3 Related work

The need for easy navigation through the Linked Data Cloud has been early recognized by the community. Swoogle [7] addresses this by performing ranking at different levels of granularity: Semantic Web documents, resources (e.g., RDF class or property) and triples (e.g. interesting RDF graph pattern). While this approach also ranks results according to different query scenarios, it focuses on graph navigation and offers no entry to the literal space of the LOD Cloud. Also, the scale of Swoogle is remarkably smaller than the one of LOTUS.

LOTUS bears much resemblance to Sindice [16], a system that allowed search on Semantic Web documents based on IRIs and keywords that appeared in those documents. Sindice crawled the network of dereferenceable IRIs and queryable SPARQL endpoints to gather data documents. The contents of each document were included in two centralized indices: one for text and one for IRIs. Sindice also semantically interpreted inverse functional relations, e.g. mapping telephone numbers onto individuals. Currently, LOTUS does not perform any type of semantic interpretation, although such functionality could be built on top of it.

There are several differences between LOTUS and Sindice. Some of these are due to the underlying LOD Laundromat architecture and some to the LOTUS system itself. Firstly, Sindice can relate IRIs and keywords to *documents* in which the former occur. LOTUS can relate keywords, IRIs and documents to each other (in all directions). Secondly, Sindice requires data to adhere to the Linked Data principles. Specifically, it requires an IRI to either dereference or be queryable in a SPARQL endpoint. LOTUS is build on top of the LOD Laundromat which accepts any type of Linked Data, e.g. it allows data entered through Dropbox. Thirdly, LOTUS allows incorrect datasets to be partially included due to the cleaning mechanism of the LOD Laundromat. This is an important feature since empirical observations collected over the LOD Laundromat indicate that at least 70% of crawled data documents contain bugs such as syntax errors. Fourthly, since Sindice returns a list of online document links as a result, it relies on the availability of the original data sources. While it has this in common with search engines for the WWW, it is known that data sources on the Semantic Web have much lower availability [9]. LOTUS returns document IRIs that can either be downloaded from their original sources or from a cleaned copy made available through the LOD Laundromat Web Service. Fifthly, LOTUS operates on a much larger scale than Sindice did. Sindice allowed 30M IRIs and 45M literals to be searched while LOTUS allows 3,136M IRIs and 4,335M literals, a difference in scale of factor 100. Finally, Sindice offered a single algorithm for resource retrieval, a characteristic it shares with popular WWW search engines, such as Google[5] and Bing[6]. In such scenario, users have no control over the retrieval process and often no understanding about how the retrieval algorithms work, leaving them with no other option than to adapt to the algorithm or look elsewhere. In the case of LOTUS, the task to adapt is set on the retrieval system

[5] `https://www.google.com/`
[6] `https://www.bing.com/`

instead of on the users. Users of LOTUS can benefit from a high flexibility of retrieval: at the moment we offer four matching and eight ranking algorithms, resulting in 32 retrieval options for the users to choose from. Our approach is user-driven and we strive to expand LOTUS to fit as many use cases as possible.

With Sindice being discontinued in 2014, there is hardly any existing attempt to build a centralized text index over the LOD Cloud or another existing LOD Cloud collection. Virtuoso's LOD Cache[7] is an entity-centric search interface which allows RDF to be searched based on free text or on URIs. LOD Cache's free text queries are translated to a text search-enriched SPARQL language. The retrieval algorithm of LOD Cache combines content-based information (by matching tokens) and relational information (through entity page rank). While LOD Cache shares many characteristics with LOTUS, the main advantage of LOTUS lies in the user flexibility to customize the retrieval criteria.

On the other hand, it is notable that the necessity for text-based access to RDF information has been recognized by most triple stores. Sesame[8], ClioPatria[9], Apache Jena[10] and Virtuoso[11] nowadays offer text search functionality either as a core functionality or an extension.

4 LOTUS

The purpose of LOTUS is to relate unstructured data (natural language text) to structured data using RDF as paradigm to express such structured data. LOTUS has access to an underlying architecture that exposes a large collection of resource-denoting terms and structured descriptions of those terms, all formulated in RDF. It indexes natural text literals that appear in the object position of RDF statements and allows the denoted resources to be findable based on approximate literal matching and, optionally, an associated language tag. The retrieval of relevant resources consists of two consecutive phases: matching and ranking. LOTUS currently includes four different matching algorithms and eight ranking algorithms.

4.1 Described resources

RDF defines a graph-based data model in which resources can be described in terms of their relations to other resources. The textual labels denoting some of these resources provide an opening to relate unstructured to structured data.

An RDF statement expresses that a certain relation holds between a pair of resources. We take a **described resource** to be any resource that is denoted by at least one term appearing in the subject position of an RDF statement.

[7] `http://lod.openlinksw.com/`
[8] `http://rdf4j.org/`
[9] `http://www.swi-prolog.org/web/ClioPatria/whitepaper.html`
[10] `https://jena.apache.org/`
[11] `http://virtuoso.openlinksw.com/`

LOTUS does not allow every resource in the Semantic Web to be found through natural language search, as some described resources are not denoted by a term that appears in the subject position of a triple whose object term is a textual label. Fortunately, many Semantic Web resources have at least one textual label linked to them and as the Semantic Web adheres to the Open World Assumption, resources that have no textual description today may receive one tomorrow, as everyone is free to add new content.

4.2 RDF Literals

In the context of RDF, textual labels appear mainly as part of *RDF literals*. We are specifically interested in literals that contain natural language text. However, not all RDF literals express – or are intended to express – natural language text. For instance, there are datatype IRIs that describe a value space of date-time points or polygons. Even though each dataset can define its own datatypes, we observe that the vast majority of RDF literals use RDF or XSD datatypes. This allows us to circumvent the theoretical limitation of not being able to enumerate all textual datatypes and focus on the datatypes `xsd:string` and `rdf:langString` [6]. Unfortunately, in practice we find that integers and dates are also regularly stored under these datatypes. As a simple heuristic filter LOTUS only considers literals with datatype `xsd:string` and `xsd:langString` that contain at least two consecutive alphabetic Unicode characters.

4.3 Linguistic entry point to the LOD Cloud

LOTUS performs offline approximate string matching. Approximate string matching[13] is an alternative to exact string matching, where a given pattern is matched to text while still allowing a number of errors. LOTUS preprocesses text and builds the data index offline, allowing the approximation model to be efficiently enriched with various precomputed metrics.

LOTUS is meant to fit use cases which require access to the LOD Cloud based on free text. In the previous subsection we explained which LOD statements will be indexed in LOTUS, which directly determines the amount of entries which are findable via LOTUS. Next, we need to ensure the user has enough functionality at hand in order to retrieve the data that suits her needs. For this purpose, we focus on two crucial design decisions: matching algorithms and ranking algorithms.

4.4 Matching algorithms

Matching literals can be performed on various levels. Two strings can be compared as full phrases, set of tokens or a set of characters. In LOTUS, we implement four matching functions to cope with the matching diversity:

M1. Phrase matching: Match a phrase in an object string. Terms in each result should occur consecutively and in the same order as in the query.

M2. Disjunctive token matching: Disjunct lookup of set of tokens occurring in the string field of an entry. The set of tokens in the query are connected by logical "OR" operator, expressing that each result should contain at least one of the queried terms. The order of the tokens between the query and the results need not coincide.

M3. Conjunctive token matching: Conjunctive lookup of set of tokens occurring in the string field of an entry. The set of tokens are connected by a logical "AND" operator, which entails that all tokens from a supplied query must match the result. The order of the tokens between the query and the results need not coincide.

M4. Conjunctive token matching with character edit distance: Conjunctive matching (with logical operator "AND") of a set of tokens, where a small Levenshtein-based edit distance on a character level is permitted. This matching algorithm is intended to account for typos and spelling mistakes.

To bring user even closer to her optimal result set, LOTUS allows further filtering to be performed on the language of literals. For this purpose, we index language tags, as explicitly specified by the dataset author or automatically detected by a language detection library. Following BCP 47 semantics[12], a language tag can contain secondary tags, such as country codes. In LOTUS, we focus on the primary language tags which denote the language of a literal and abstract from the complementary tags, such as country or dialect identifiers.

4.5 Ranking algorithms

Ranking algorithms on the Web of data operate on top of a similarity function, which can be literal-based or relational [5][13]. Literal-based (also called content-based) similarity functions exclusively compare the textual content of a query pattern to each matched result. Such comparison can be done on different granularity of text: we distinguish character-based (Levenshtein similarity, Jaro similarity, etc.) and token-based (Jaccard, Dice, Overlap, Cosine similarity, etc.) approaches. The content similarity function can also be information-theoretical, exploiting the probability distributions extracted from data statistics. Relational similarity functions complement the literal similarity by taking the underlying structure of the tree (tree-based similarity) or the graph (graph-based similarity) into account.

We use this classification of similarity algorithms as a starting point for our implementation of three literal-based (R1-R3) and five relational functions (R4-R8) in LOTUS:

R1. Character length normalization: The score of an entry is counter-proportional to the number of characters in its object string.

[12] `https://tools.ietf.org/html/bcp47`

[13] The reader is referred to this book for detailed explanation of similarity functions and references to original publications

R2. Practical scoring function: Token-based scoring function, whose score is a product of three information retrieval metrics: term frequency (TF), inverse-document frequency (IDF) and length normalization (inverse-proportional to the number of tokens)[14]

R3. Phrase proximity: Boosts the score of the entries with a low edit distance to the query phrase.

R4. Terminological richness: Involvement of controlled vocabularies, i.e. classes and properties, in the original document from which the triple stems from.

R5. Semantic richness of the document: Mean graph connectedness degree of the original document.

R6. Recency ranking: The moment in time when the original document was last modified. Triples from recently updated documents have higher score.

R7. Degree popularity: Total graph connectedness degree (indegree + outdegree) of the subject resource.

R8. Appearance popularity: Number of documents in which the subject appears.

5 Implementation

The LOTUS system architecture consists of two components: Index Building (IB) procedure and Public Interface (PI). The role of the IB component is to index strings from LOD Laundromat; the role of PI is to expose the indexed data to users for querying. The two system components are executed sequentially: data is initially indexed, then it can be queried through the exposed public interface.

5.1 System Architecture

As indexing of big data in the range of billions of RDF statements is expensive, we perform offline data indexing. Furthermore, as resources and documents repeat over multiple (potentially many) statements, we need clever ways to compute and access the metadata on documents and resources. Due to this, we initially cache the metadata information on documents needed for the ranking algorithms R4-R6 (Section 4.5). This includes the date when the document was last modified, its mean graph degree and its terminological richness coefficient; and has to be obtained once per dataset through Frank [2] or the Lod Laundromat SPARQL endpoint[15].

The relational information on resource URIs is trickier to obtain and store, as the number of resources in the LOD Laundromat is huge and a resource occurrences are often scattered across many documents, therefore we pre-store the information needed for ranking algorithms R7 and R8 in RocksDB.

[14] This function is the default scoring function in ElasticSearch. Detailed description of its theoretical basis and implementation is available at `https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html`

[15] `http://lodlaundromat.org/sparql/`

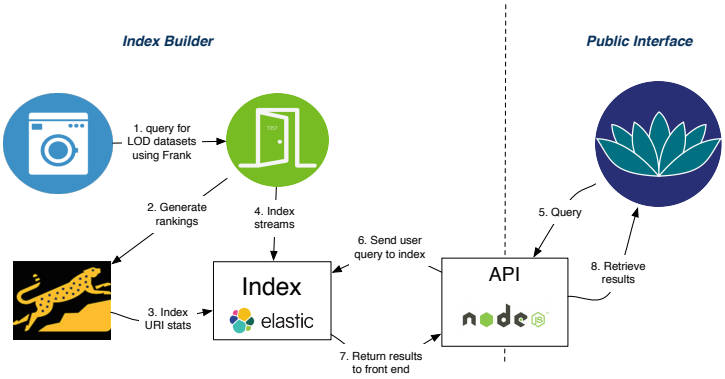


Fig. 2. LOTUS System Architecture

After the relational ranking data, we start the indexing process over all data from LOD Laundromat through a batch loading procedure. This procedure uses LOD Laundromat's query interface, Frank (Step 1 in Figure 2), to enumerate all LOD Laundromat data sets and stream them to a client script. Following the approach described in Section 4, we consider only the statements that contain a natural language literal as an object. The client script parses the received RDF statements and performs a bulk indexing request in ElasticSearch (ES),[16] where the textual index is built (Steps 2, 3 and 4 in Figure 2).

Once the indexing process is finished, we have prepared all the data LOTUS needs to perform retrieval over the LOD Laundromat collection. The index is only incrementally updated when new data appears in LOD Laundromat, this is triggered by an event handler in LOD Laundromat.

Each ElasticSearch entry has the following format:

```
{
  "docid": IRI,
  "langtag": STRING,
  "predicate": IRI,
  "string": STRING,
  "subject": IRI,
  "length": float,
  "docLastModified": int,
  "docTermRichness": float,
  "docSemRichness": float,
  "uriDegree": int,
  "uriNumDocs": int
}
```

The field "string" is preprocessed ("analyzed") by ElasticSearch at indexing time, which allows for approximate lookup of the lexical form from the object literal. The motivation behind analyzing the "string" field comes naturally, as

[16] `https://www.elastic.co/products/elasticsearch`

this contains unstructured text and will rarely be queried for exact match. We also index the language tag ("langtag") of every triple literal, as described in Section 4. For every triple, we also store the subject and predicate URIs, as well as the LOD Laundromat document identifier.

The remaining six fields in our ElasticSearch index structure are collected for the ranking algorithms in LOTUS: "length" is based on the number of characters in the "string" field (ranking R1); "docLastModified", "docTermRichness" and "docSemRichness" are document-oriented meta numbers (used in rankings R4-R6); "uriDegree" and"uriDocs" save relational information about the subject URI of the ElasticSearch entry (ranking algorithms R7 and R8).

5.2 Distributed Architecture

In our implementation, we leverage the distributed features of ElasticSearch and scale LOTUS horizontally over 5 servers. Each server has 128 GB of RAM, 6 core CPU with 2.40GHz and 3 SSD hard disks with 440 GB of storage each. We enable data replication to ensure high runtime availability of the system.

5.3 API

Users can access the underlying data through an API. The usual query flow is described in steps 5-8 of Figure 2. Our NodeJs[17] interface to the indexed data exposes a single query endpoint[18]. Through this endpoint, the user can supply the query pattern, indicate the desired matching and ranking algorithms, and optionally supply additional requirements, such as language tag or number of results to retrieve. The basic query parameters are[19]:

- **string:** A natural language string to match in LOTUS
- **match:** Choice of a matching algorithm, one of *phrase, terms, conjunct, fuzzy-conjunct*
- **rank:** Choice of a ranking algorithm, one of *lengthnorm, psf, proximity, termrichness, semrichness, recency, degree, appearance*
- **size:** Number of best scoring results to be included in the response
- **langtag:** Two-letter language identifier

LOTUS is also available as a web interface at `http://lotus.lodlaundromat.org/` for simple exploration of the data. Code of the API functions and data from our experiments can be found on github.[20]. The code used to create the LOTUS index is also publicly available[21]

[17] `https://nodejs.org`
[18] `http://lotus.lodlaundromat.org/retrieval`
[19] See `http://lotus.lodlaundromat.org/docs` for additional parameters and more detailed information
[20] `https://github.com/filipdbrsk/LOTUS\_Search/`
[21] `https://github.com/filipdbrsk/LOTUS\_Indexer/`

Table 1. Statistics on the indexed data

total # literals encountered	12,380,443,617
#xsd:string literals	6,205,754,116
#xsd:langString literals	2,608,809,608
# indexed entries in ES	4,334,672,073
# distinct sources in ES	493,181
# hours to create the ES index	67
disk space used for the ES index	509.81 GB
# in_degree entries in RocksDB	1,875,886,294
# out_degree entries in RocksDB	3,136,272,749
disk space used for the RocksDB index	46.09 MB

6 Performance Statistics and User Scenarios

As the LOTUS framework does not pertain to provide a one-size-fits-all solution, we present some performance statistics and scenarios in this section. We test LOTUS on a series of queries and we see that the different matching and ranking algorithms implemented in LOTUS come with very different performance characteristics.

6.1 Performance Statistics

Statistics over the indexed data are presented in Table 1. We encountered over 12 billion literals in LOD Laundromat, 8.81 billion of which are defined as a natural language string (with xsd:string or xsd:langString datatype). According to our approach, 4.33 billion of all literals (around 35%) express natural language strings. The initial LOTUS index was created in 67 hours and consumes 509.81 GB of disk space storage. The current index consists of 4.33 billion entries, stemming from 493,181 distinct datasets.[22]

In order to give an indication of the performance of LOTUS from a client perspective we preformed 324,000 text queries. For this we extracted the 6,000 most frequent bigrams, trigrams and quadgrams (18,000 N-grams in total) from the source of *A Semantic Web Primer*[1]. Non-alphabetic characters were first removed and case normalization was applied. For each N-gram we performed a text query using one of three matchers in combination with one of six rankers. The results are shown in Figure 3. We observe certain patterns in this Figure. Matching by disjunctive lookup of terms (M2) is strictly more expensive than the other two matching algorithms. We also notice that bigrams are more costly to retrieve than trigrams and quadrams. Finally, we observe that there is no difference between the response time of the relational rankings which is expected, because these rank results in the same manner, through sorting pre-stored integers in a decreasing order. We note that we experienced no problems with the availability of LOTUS.

[22] The number of different source documents in LOTUS is lower than the overall number of sources in LOD Laundromat, as not every source document contains string literals.

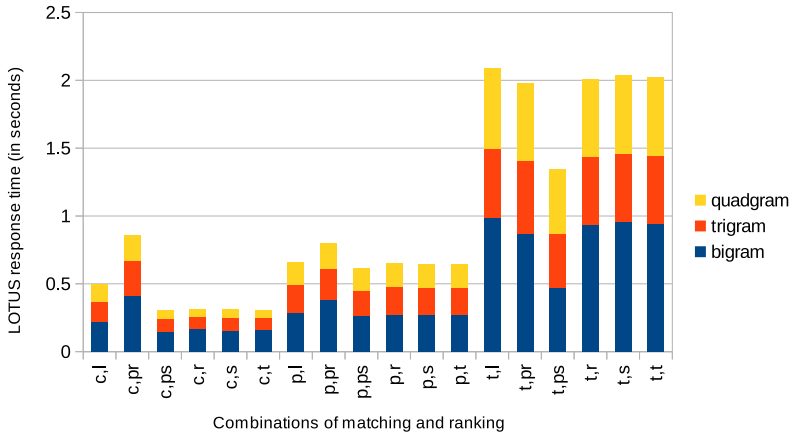


Fig. 3. LOTUS average response times in seconds for bi- tri- and quadgram requests. The horizontal axis shows 18 combinations of a matcher and a ranker. The matchers are *conjunct* (c), *phrase* (p) and *terms* (t). The rankers are *length normalization* (l), *proximity* (pr), *psf* (ps), *recency* (r), *semantic richness* (s) and *term richness* (t). The bar chart is cumulative per match+rank combination. For instance, the first bar indicates that the combination of conjunct matching and length normalization takes 0.20 seconds for bigrams, 0.15 seconds for trigrams, 0.15 seconds for quadgrams and 0.5 seconds for all three combined. The slowest query is for bigrams with terms matching and length normalization, which takes 1.0 seconds on average.

6.2 Usage Scenarios

To demonstrate the flexibility and the potential of the LOTUS framework, we performed retrieval on the query "graph pattern". For this query, we chose the phrase-based matching option and iterated through the different ranking algorithms. The top 8 results obtained with each of the different ranking modes are presented in Figure 4.

In our results, the literal-based ranking algorithms all put the same result on the top position, but there is a fair amount of variation in its top 8 results. The relational similarity scorers perform quite differently and all present entirely new top 8 results. This allows a user who is, for example, interested in analyzing the latest changes in a dataset to select the Recency ranking algorithm and retrieve statements from the most recently updated datasets first. A user who is more interested in linguistic features of a query can use the length normalization ranking to explore resources which match the query as exact as possible. Use-cases interested in multiple occurrences of informative phrases would benefit from the practical scoring function. When working in use cases concerning popularity of resources, the degree-based rankings can be useful.

Users can also vary the matching dimension. Suppose one is interested to explore resources with typos or spelling variation: fuzzy conjunctive matching would be the appropriate matching algorithm to apply.

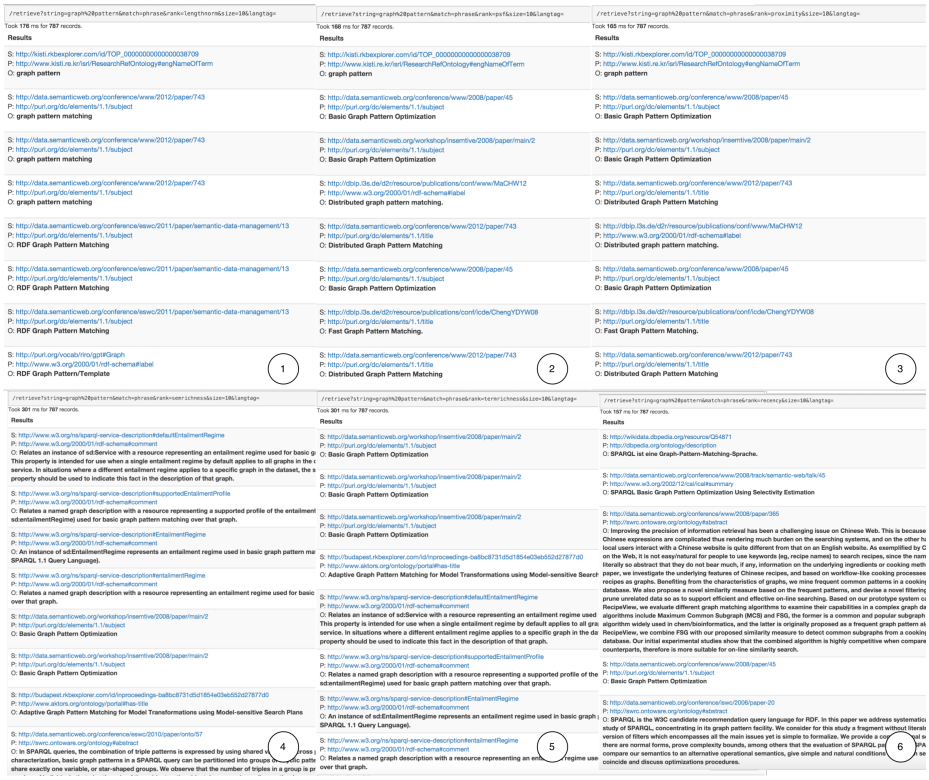


Fig. 4. Results of query "graph pattern" with terms-based matching and different rankings: 1) Length Normalization, 2) Practical scoring function, 3) Phrase proximity, 4) Semantic richness, 5) Terminological richness, and 6) Recency.

7 Discussion and Conclusions

In this paper, we presented LOTUS, a full-text entry point to the LOD collection in LOD Laundromat. We detailed the specific difficulties in accessing textual content in the LOD cloud today and the approach taken by LOTUS to address these. LOTUS allows its users to customize their own retrieval method by exposing analytically well-understood matching and ranking algorithms.

LOTUS performs retrieval by taking into account both textual similarity and certain structural properties of the underlying data. In the current version of LOTUS we focused on context-free[23] ranking of results and indicated the versatility of LOTUS by measuring its performance and showing how the ranking algorithms affect the search results. A context-dependent ranking mechanism, could make use of additional context coming from the query in order to re-score and improve the order of the results. Context-dependent functionality could (to

[23] By "context-free", we mean that the retrieval process can not be directly influenced by additional restrictions or related information.

some extent) be built into LOTUS on a single RDF statement level. However, graph-wide integration with structured data would require a different approach and implementation, potentially based on a fulltext-enabled triple store (e.g. Virtuoso).

Although further optimization is always possible, the current version of LOTUS performs indexing and querying in an efficient and scalable manner. This is largely thanks to the underlying distributed architecture. Future work will evaluate the precision and recall of LOTUS on concrete applications, such as Entity Linking.

References

1. Antoniou, G., Groth, P., van Harmelen, F., Hoekstra, R.: A Semantic Web Primer. The MIT Press, 3rd edition edn. (2012)
2. Beek, W., Rietveld, L.: Frank: Algorithmic Access to the LOD Cloud. Proceedings of the ESWC Developers Workshop 2015 (2015)
3. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: Lod laundromat: a uniform way of publishing other peoples dirty data. In: ISWC 2014, pp. 213–228 (2014)
4. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: SPARQL web-querying infrastructure: Ready for action? In: ISWC 2013 (2013)
5. Christophides, V., Efthymiou, V., Stefanidis, K.: Entity Resolution in the Web of Data. Morgan & Claypool Publishers (2015)
6. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax (2014)
7. Ding, L., Pan, R., Finin, T., Joshi, A., Peng, Y., Kolari, P.: Finding and ranking knowledge on the semantic web. In: The Semantic Web–ISWC 2005, pp. 156–170. Springer (2005)
8. Harris, S., Seaborne, A., Prudhommeaux, E.: Sparql 1.1 query language. W3C Recommendation 21 (2013)
9. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An Empirical Survey of Linked Data Conformance. Web Semantics: Science, Services and Agents on the World Wide Web 14, 14–44 (2012)
10. Ilievski, F., Beek, W., van Erp, M., Rietveld, L., Schlobach, S.: Lotus: Linked open text unleashed. In: COLD workshop, ISWC (2015)
11. Mendes, P.N., Jakob, M., García-Silva, A., Bizer, C.: DBpedia Spotlight: Shedding Light on the Web of Documents. pp. 1–8. 7th International Conference on Semantic Systems. ACM (2011)
12. Moro, A., Raganato, A., Navigli, R.: Entity linking meets word sense disambiguation: a unified approach. Transactions of the Association for Computational Linguistics 2, 231–244 (2014)
13. Navarro, G.: A guided tour to approximate string matching. ACM computing surveys (CSUR) 33(1), 31–88 (2001)
14. Prud'hommeaux, E., Buil-Aranda, C.: SPARQL 1.1 Federated Query (2013), `http://www.w3.org/TR/sparql11-federated-query/`
15. Rizzo, G., Troncy, R.: Nerd: a framework for unifying named entity recognition and disambiguation extraction tools. In: Proceedings of EACL 2012. pp. 73–76 (2012)
16. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the open linked data. In: Proceedings ISWC'07/ASWC'07. pp. 552–565 (2007)