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, Amsterdam, The Netherlands
{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: no centralized query service exists and the support for natural language access is limited. We present LOTUS: Linked Open Text UnleaShed, a text-based entry point to a massive subset of today's Linked Open Data Cloud. Recognizing the use case dependency of resource retrieval, LOTUS provides an adaptive framework in which a set of matching and ranking algorithms are made available. Researchers and developers are able to tune their own LOTUS index by choosing and combining the matching and ranking algorithms that suit their use case best. In this paper, we explain the LOTUS approach, its implementation and the functionality it provides. We demonstrate the ease with which LOTUS enables text-based resource retrieval at an unprecedented scale in concrete and domain-specific scenarios. Finally, we provide evidence for the scalability of LOTUS with respect to the LOD Laundromat, the largest collection of easily accessible Linked Open Data currently available.

Keywords: Findability · Text indexing · Semantic search · Scalable data management

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 access is limited. A resource is typically 'found' by memorizing its resource-denoting IRI. The lack of a global entry point to resources through a flexible text index is a serious obstacle for Linked Data consumption.

We introduce LOTUS: Linked Open Text UnleaShed,[2] a central text-based entry point to a large subset of today's LOD Cloud. Centralized text search on

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

[2] An early version of LOTUS was presented at COLD 2015 [13]. This paper is a significantly extended and updated version of that work.

H. Sack et al. (Eds.): ESWC 2016, LNCS 9678, pp. 470–485, 2016.
DOI: 10.1007/978-3-319-34129-3_29

the LOD Cloud is not new as Sindice[3] and LOD Cache[4] show. However, LOTUS differs from these previous approaches in three ways: (1) its scale (its index is about 100 times bigger than Sindice's was), (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.

LOTUS indexes every natural language literal from the LOD Laundromat data collection, a cached copy of a large subset of today's LOD Cloud spanning tens of billions of ground statements. The task of resource retrieval is a two-part process consisting of matching and ranking. Since there is no single combination of matching and ranking that is optimal for every use case, LOTUS enables users to customize the resource retrieval to their needs by choosing from a variety of matching and ranking algorithms. LOTUS is not a semantic search engine intended for end users, but a framework for researchers and developers in which semantic search engines can be developed and evaluated. The flexibility of the LOTUS approach towards resource retrieval shifts the adaptation task from the user to the retrieval system, making LOTUS attractive for a wide range of use cases including Information Retrieval, Entity Linking and Network Analysis. Existing Entity Linking systems such as NERD [17]), rely on a single or limited set of knowledge sources, typically DBpedia, and thus suffer from limited coverage. An adaptive linguistic entry point to the LOD Cloud, such as 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 Sect. 2, we detail the problem of performing linguistic search on the LOD Cloud and we formalize it by defining an array of requirements. Section 3 presents relevant previous work on Semantic Web text search. Section 4 describes the LOTUS framework through its model, approach and initial collection of matching and ranking algorithms. The implementation of LOTUS is reported in Sect. 5. Scalability tests and typical usage scenarios of LOTUS are discussed in Sect. 6. We conclude by considering the key strengths, limitations and future plans for LOTUS in Sect. 7.

2 Problem Description

In this section, we detail the requirements for a global text-based entry point to linked open data and the strengths and weaknesses of current findability strategies with respect to these requirements.

2.1 Requirements

The Semantic Web currently relies on four main strategies to find relevant resources: datadumps, IRI dereferencing, Linked Data Fragments (LDF) and SPARQL, but these are not particularly suited to global text-based search.

Since it is difficult to memorize IRIs and structured querying requires prior knowledge of how the data is organized, text-based search for resources is an

[3] http://www.sindice.com/, discontinued in 2014.
[4] http://lod.openlinksw.com/.

important requirement for findability on the Semantic Web (Req1: Text-based). Furthermore, we also require text-based search to be resilient with respect to minor variations such as typos or spelling variations (Req2: Resilience). An important principle of the Semantic Web is that anybody can say anything about any topic (AAA). The findability correlate of this principle is not implemented by existing approaches: only what an authority says or links to explicitly can be easily found. We formulate the correlate requirement as "anybody should be able to find anything that has been said about any topic" (Req3: AFAA).

Decentralized data publishing makes text-based search over multiple data sources difficult. Not all sources have high availability (especially a problem for SPARQL) and results from many different sources have to be integrated on the client side (especially a challenge for IRI dereferencing and Linked Data Fragments). Hence, we set requirements on availability (Req4: Availability) and scalability (Req5: Scalability) for a text-based search service. While LDF provides better serviceability for singular endpoints than the other three approaches, it is still cumbersome to search for resources across many endpoints (Req6: Serviceability). Existing Semantic Web access approaches do not implement IR-level search facilities. In Sect. 3 some systems will be discussed that built some of this functionality on top of standard access methods. However, these systems focus on a single algorithm to work in each and every case. This may be suitable for an end-user search engine but not for a framework in which search engines are build and evaluated, bringing us to our last requirement of customizeability (Req7: customizeability). Below we iterate our requirements:

Req1. Text-based. Resource-denoting IRIs should be findable based on text-based queries that match (parts of) literals that are asserted about that IRI, possibly by multiple sources.

Req2. Resilience. Text-based search should be resilient against typo's and small variations in spelling (i.e., string similarity and fuzzy matching in addition to substring matching).

Req3. AFAA. Authoritative and non-authoritative statements should both be findable.

Req4. Availability Finding resources should not depend on the availability of all the original resource-publishing sources.

Req5. Scalability. Resources should be searchable on a Web scale, spanning tens of billions of ground statements over hundreds of thousands of datasets.

Req6. Serviceability. The search API must be freely available for humans (Web UI) and machines (REST) alike.

Req7. Customizability. Search results should be ranked according to a customizable collection of rankings to support a wide range of use cases.

2.2 Current State-of-the-Art

Datadumps implement a rather simple way of finding resource-denoting terms: one must know the exact Web address of a datadump in order to download and extract resource-denoting IRIs. This means that search is neither text-based

(Req1) nor resilient (Req2). Extraction has to be performed manually by the user resulting in low serviceability (Req6). Datadumps do not link explicitly to assertions about the same resource that are published by other sources (Req3).

IRIs dereference to a set of statements in which that IRI appears in the subject position or, optionally, object position. Which statements belong to the dereference result set 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. Non-authoritative statements about the same IRI cannot be found. Non-authoritative statements can only be found accidentally by navigating the interconnected graph of dereferencing IRIs. As blank nodes do not dereference significant parts of the graph cannot be traversed. This is not only a theoretical problem as 7 % of all RDF terms are blank nodes [11]. In practice, this means that non-authoritative assertions are generally not findable (Req3). Since only IRIs can be dereferenced, text-based access to the Semantic Web cannot be gained at all through dereferencing (Req1). Thus, it is not possible to find a resource-denoting IRI based on words that appear in RDF literals to which it is (directly) related, or based on keywords that bear close similarity to (some of the) literals to which the IRI is related (Req2).

Linked Data Fragments [21] (LDF) significantly increases the serviceability level for single-source Linked Data retrieval (Req6) by returning metadata descriptions about the returned data. E.g., by implementing pagination LDF allows all statements about a given resource to be extracted without enforcing arbitrary limits. This is not guaranteed by IRI dereferencing or SPARQL (both of which lack pagination). An extension to LDF [20] adds efficient substring matching, implementing a limited form of text-based search (Req1). Due to the reduced hardware requirements for running an LDF endpoint it is reasonable to assume that LDF endpoints will have higher availability than SPARQL endpoints (Req4). LDF does not implement resilient matching techniques such as fuzzy matching (Req2) and does not allow non-authoritative statements about a resource to be found (Req3).

SPARQL allows resources to be found based on text search that (partially) matches literal terms (Req1). More advanced matching and ranking approaches such as string similarity or fuzzy matching are generally not available (Req2). As for the findability of non-authoritative statements, SPARQL has largely the same problems as the other three approaches (Req3). There is also no guarantee that all statements are disseminated by some SPARQL endpoint. Endpoints that are not pointed to explicitly cannot be found by automated means. Empirical studies show that many SPARQL endpoints have low availability [4] (Req4).

Federation is implemented by both LDF and SPARQL, allowing queries to be evaluated over multiple endpoints. Federation is currently unable to implement Web-scale resource search (Req5) since every endpoint has to be included explicitly in the query. This requires the user to enter the Web addresses of endpoints, resulting either in low coverage (Req3) or low serviceability (Req6). Other problems are that the slowest endpoint determines the response time of the entire query and results have to be integrated at the client side. Since

Web-wide resource search has to span hundreds of thousands of data sources these problems are not merely theoretical.

3 Related Work

Several systems have implemented text-based search over Semantic Web data: Swoogle [8], SemSearch [14], Falcons [5], Semplore [22], SWSE [10], Hermes [18], Sindice/Sigma [19]. Swoogle allows keyword-based search of Semantic Web documents. Hermes performs keyword-based matching and ranking for schema resources such as classes and (object) properties. Falcons, Semplore, SWSE and Sindice search for schema and data alike.

Exactly how existing systems extract keywords from RDF data is largely undocumented. Most services use standard NLP approaches such as stemming and stopword removal to improve the keyword index. Furthermore, N-grams are used to take multi-word terms into account (Swoogle), WordNet is used by Hermes enrich keywords with synonyms. Many systems rely on off-the-shelf text indexing tools such as Lucene that perform similar keyword extraction tasks.

Many systems use metrics derived from the Information Retrieval field such as Term Frequency (TF), Inverse Document Frequency (IDF) and PageRank to rank results. Hermes, for example, implements a TF variant called Element Frequency (EF) that quantifies the popularity of classes/properties in terms of the number of instances they have. Sindice calculates a version of IDF by considering the distinctiveness of data sources from which instances originate, as well as the authority of a data sources: resources that appear in a data source that has the same host name are ranked higher than non-authoritative ones. In practice, TF and IDF are often combined in a single TF/IDF-based metric, in order to have a balanced measure of popularity and distinctiveness.

While the basic idea of adapting existing IR metrics such as TF/IDF and PageRank for text-based search of Semantic Web resources is a common ground for existing systems, they all implement this idea in different ways; using different metrics, the same metrics in different ways, or combining the various metrics in a different way. This makes it difficult to compare existing text-based Semantic Web search systems with one another. Also, the fact that the adapted algorithms differ between these systems provides evidence that there is no single retrieval algorithm that fits every use case. Acknowledging that combining existing approaches into a final ranking over end-results is highly application-dependent and is as much an art as a science, LOTUS takes a very different approach. LOTUS provides an environment in which multiple matching and ranking approaches can be developed and combined. This makes it much easier to evaluate the performance of individual rankers on the global end result.

Existing systems operate on data collections of varying size. Sindice, Falcons and Hermes are formally evaluated over hundreds of millions of statements, while Semplore is evaluated over tens of millions of statements. Falcons, Swoogle and Sindice have at some point in time been available as public Web Services for users to query. With Sindice being discontinued in 2014, no text-based Semantic Web search engine is widely available to the Semantic Web community today.

In addition to the work on semantic search engines, there have been multiple attempts to extend existing SPARQL endpoints with more advanced NLP tooling such as fuzzy string matching and ranking over results [9,12,15]. This improves text search for a restricted number of query endpoints but does not allow text-based queries to cover a large number of endpoints, as the problem of integrating ranked results from many endpoints at the client side has not been solved. Virtuoso's LOD Cache[5] provides public access to a text search-enriched SPARQL endpoint. It differs from LOTUS in that it does not allow the matching and ranking algorithms to be changed or combined in arbitrary ways by the user and as a commercial product its specific internals are not public.

4 LOTUS

LOTUS relates unstructured to structured data using RDF as a paradigm to express such structured data. LOTUS is integrated with a central 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 matching. LOTUS currently includes four different matching algorithms and eight ranking algorithms, which leverage both textual features and relational information from the RDF graph.

4.1 Model

Denoted Resources. RDF defines a graph-based data model in which resources can be described in terms of their relations to other resources. An RDF statement expresses that a certain relation holds between a pair of resources.

The textual labels denoting some of these resources provide an opening to relate unstructured to structured data. LOTUS does not allow every resource in the Semantic Web to be found through text search, as some resources are not denoted by a term that appears as a subject of a triple whose object term is a textual label. Fortunately, many Semantic Web resources are denoted by at least one textual label and as the Semantic Web adheres to the Open World Assumption, resources with no textual description today may receive one tomorrow.

RDF Literals. In the context of RDF, textual labels appear 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` [7]. Unfortunately, in practice we find that integers and

[5] http://lod.openlinksw.com/.

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.

4.2 Linguistic Entry Point to the LOD Cloud

LOD Laundromat. LOTUS is built on top of LOD Laundromat, a Semantic Web crawling and cleaning architecture and centralized data collection. LOD Laundromat spans hundreds of thousands of data documents and tens of billions of ground statements. Inherent to this design decision, LOTUS fulfills three of the requirements we set in Sect. 2: (1) the scale on which LOTUS operates is in range of billions and is 100 times bigger than that of previous semantic search systems that were made generally available for the community to use (Req5); (2) since the LOD Laundromat data is collected centrally, finding both authoritative and non-authoritative RDF statements is straightforward (Req3); (3) as a cached copy of linked data, LOD Laundromat allows its IRIs to be dereferenceable even when the original sources are unavailable (Req4).

Linguistic Entry Point. LOTUS allows RDF statements from the LOD Laundromat collection to be findable through approximate string matching on natural language literals. Approximate string matching [16] is an alternative to exact string matching, where one textual pattern is matched to another while still allowing a number of errors. In LOTUS, query text is approximately matched to existing RDF literals (and their associated documents and IRI resources) (Req1).

In order to support the approximate matching and linguistic access to LD through literals, LOTUS makes use of an inverted index. As indexing of big data in the range of billions of RDF statements is expensive, the inverted index of LOTUS is created offline. This also allows the approximation model to be efficiently enriched with various precomputed retrieval metrics.

4.3 Retrieval

Matching Algorithms. Approximate matching of a query to literals can be performed on various levels: as phrases, as sets of tokens, or as sets of characters. LOTUS implements four matching functions to cope with this diversity (Req2):

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: Match some of the query tokens in a literal. The query tokens are connected by a logical "OR" operator, expressing that each match should contain at least one of the queried tokens. The order of the tokens between the query and the matched literals need not coincide.
M3. Conjunctive token matching: Match all query tokens in a literal. The set of tokens are connected by a logical "AND" operator, which entails that all tokens from the query must be found in a matched literal. The order of the tokens between the query and the matched literals 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 a user even closer to her optimal set of matches, LOTUS facilitates complementary filtering based on the language of the literals, as explicitly specified by the dataset author or automatically detected by a language detection library. While a language tag can contain secondary tags, e.g. to express country codes, LOTUS focuses on the primary language tags which denote the language of a literal and abstracts from the complementary tags.

Ranking Algorithms. Ranking algorithms on the Web of data operate on top of a similarity function, which can be content-based or relational [6].[6] The content-based similarity functions exclusively compare the textual content of a query to each potential result. Such comparison can be done on different granularity of text, leading to 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 content similarity approaches by considering the underlying structure of the tree (tree-based) or the graph (graph-based).

We use this classification of similarity algorithms as a starting point for our implementation of three content-based (R1-R3) and five relational functions (R4-R8) in LOTUS, thus addressing Req7:

R1. Character length normalization: The score of a match is counter-proportional to the number of characters in the lexical form of its literal.

R2. Practical scoring function:[7] The score of a match is a product of three token-based information retrieval metrics: term frequency (TF), inverse-document frequency (IDF) and length normalization (inverse-proportional to the number of tokens).

R3. Phrase proximity: The score of a match is inverse-proportional to its edit distance with respect to the query.

R4. Terminological richness: The score of a match is proportional to the presence of controlled vocabularies, i.e. classes and properties, in the original document from which the RDF statement stems from.

R5. Semantic richness of the document: The score of a match is proportional to the mean graph connectedness degree of the original document.

R6. Recency ranking: The score of a match is proportional to the moment in time when the original document was last modified. Statements from recently updated documents have higher score.

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

[7] 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.

R7. Degree popularity: The score of a match is proportional to the total graph connectedness degree (indegree + outdegree) of its subject resource.

R8. Appearance popularity: The score of a match is proportional to the number of documents in which its subject appears.

5 Implementation

The LOTUS system architecture consists of two main components: the Index Builder (IB) and the Public Interface (PI). The role of the IB is to index strings from LOD Laundromat; the role of the PI is to expose the indexed data to users for querying. The two components are executed sequentially: data is indexed offline, after which it can be queried via the exposed public interface.

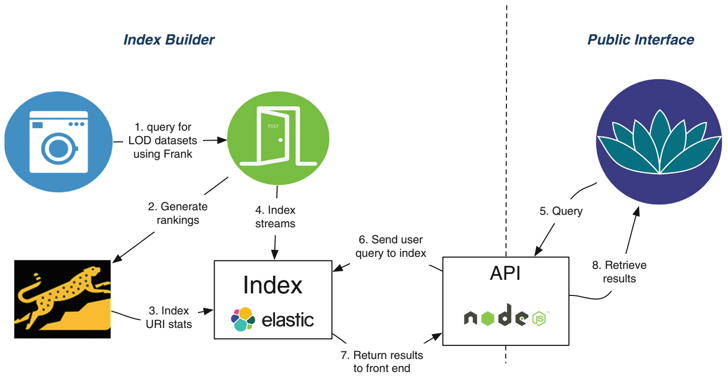


Fig. 1. LOTUS system architecture

5.1 System Architecture

Since our rankings rely on metadata about documents and resources which is reused across statements, we need clever ways to compute and access this metadata. For this purpose, we pre-store the document metadata needed for the ranking algorithms R4-R6, which includes the last modification date, the mean graph degree and the terminological richness coefficient of each LOD Laundromat document. We use the LOD Laundromat access tools (Frank [2] and the SPARQL endpoint[8]) to obtain these. The rankings R7 and R8 use metadata about resource IRIs. Computing, storage and access to this information is more challenging, as the number of resources in the LOD Laundromat is huge and their occurrences are scattered across documents. To resolve this, we store the graph degree of a resource and number of documents where it appears in RocksDB.[9]

[8] http://lodlaundromat.org/sparql/.

[9] http://rocksdb.org/.

Once the relational ranking data is cached, 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 Fig. 1), to list all LOD Laundromat documents and stream them to a client script. Following the approach described in Sect. 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),[10] where the textual index is built (Steps 2, 3 and 4 in Fig. 1).

As soon as the indexing process is finished, LOTUS contains the data it needs to perform text-based retrieval over the LOD Laundromat collection. Its index is only incrementally updated when new data is added in LOD Laundromat.[11]

For each RDF statement from the LOD Laundromat, we index: (1) *Information from the statement itself:* subject IRI, predicate IRI, lexical form of the literal ("string"), length of the "string" field (in number of characters), language tag of the literal and document ID; (2) *Metadata about the source document:* last modification date, terminological richness coefficient and semantic richness coefficient; (3) *Metadata about the subject resource:* graph degree and number of documents in which the resource appears.

We store the metadata for (2) and (3) in a numeric format to enable their straightforward usage as ranking scores by ElasticSearch.

5.2 Implementation of the Matching and Ranking Algorithms

While the matching algorithms we introduce are mainly adaptations of off-the-shelf ElasticSearch string matching functions, we allow them to be combined with relational information found in the RDF statement to improve the effectiveness of the matching process. The approximate matching algorithms M1-M4 operate on the content of the "string" field, storing the lexical form of a literal. This field is preprocessed ("analyzed") by ElasticSearch at index time, thus allowing existing ElasticSearch string matching functionalities to be put into practice for matching. We allow users to further restrict the matching process by specifying relational criteria: language tag of the literal ("langtag"), associated subject and predicate as well as LOD Laundromat document identifier.

Similarly to the matching algorithms, our ranking algorithms rely on both ElasticSearch functionality and relational information extracted from LOD Laundromat. Concretely, our rankings are based on: (1) *Scoring functions from ElasticSearch* (R1-R3); (2) *Document-level scores:* last modification date (R4), terminological richness (R5) and semantic richness (R6); (3) *Resource-level scores:* graph degree (R7) and number of documents that contain the resource (R8).

5.3 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,

[10] https://www.elastic.co/products/elasticsearch.
[11] This procedure is triggered by an event handler in the LOD Laundromat itself.

6 core CPU with 2.40 GHz and 3 SSD hard disks with 440 GB of storage each. We enable data replication to ensure high runtime availability of the system.

5.4 API

Users can access the underlying data through an API. The usual query flow is described in steps 5–8 of Fig. 1. We expose a single query endpoint,[12] through which the user can supply a query, choose a combination of matching and ranking algorithms, and optionally provide additional requirements, such as language tag or number of results to retrieve. The basic query parameters are:[13]

- **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 human-friendly exploration of the data, thus fulfilling Req6 on usefulness for both humans and machines. Code of the API functions and data from our experiments can be found on github.[14] The code used to create the LOTUS index is also publicly available.[15]

6 Performance Statistics and User Scenarios

As LOTUS does not 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 show the impact of different matching and ranking algorithms.

6.1 Performance Statistics

Statistics over the indexed data are given in Table 1. LOD Laundromat contains over 12 billion literals, 8.81 billion of which are defined as a natural language string (`xsd:string` or `xsd:langString` datatype). According to our approach, 4.33 billion of these ($\sim$35 %) express natural language strings. The initial LOTUS index was created in 67 h, consuming 509.81 GB of disk space. The current index consists of 4.33 billion entries, stemming from 493,181 distinct datasets.[16]

[12] http://lotus.lodlaundromat.org/retrieval.

[13] See http://lotus.lodlaundromat.org/docs for additional parameters and more detailed information.

[14] https://github.com/filipdbrsk/LOTUS_Search/.

[15] https://github.com/filipdbrsk/LOTUS_Indexer/.

[16] The number of distinct sources in LOTUS is lower than the number of documents in LOD Laundromat, as not every document contains natural language literals.

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

To indicate the performance of LOTUS from a client perspective we preformed 324,000 text queries. 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 Fig. 2. We observe certain patterns in this Figure. Matching disjunctive 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 quadgrams. 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.

The performance and scalability of LOTUS is largely due to the use of ElasticSearch, which justifies our rationale in choosing ElasticSearch as one of our two main building blocks: it allows billions of entries to be queried within reasonable time constraints, even running on an academic hardware infrastructure.

6.2 Usage Scenarios

To demonstrate the flexibility and the potential of the LOTUS framework, we performed retrieval on the query "graph pattern". We matched this query as a phrase (M1) and iterated through the different ranking algorithms. The top results obtained with two of the different ranking modes are presented in Fig. 3.

The different ranking algorithms allow a user to customize her results. For example, if a user is interested in analyzing the latest changes in a dataset, the Recency ranking algorithm will 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 that match the query as precisely as possible. Users interested in multiple occurrences of informative phrases could benefit from the practical scoring function. When the popularity of resources is important, the degree-based rankings can be useful.

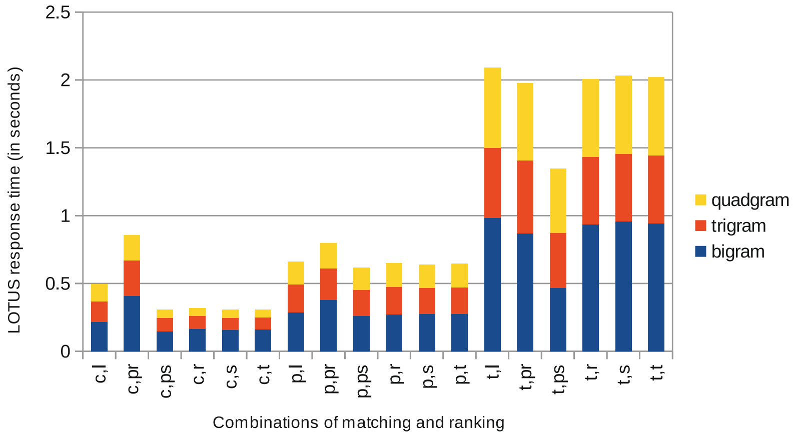
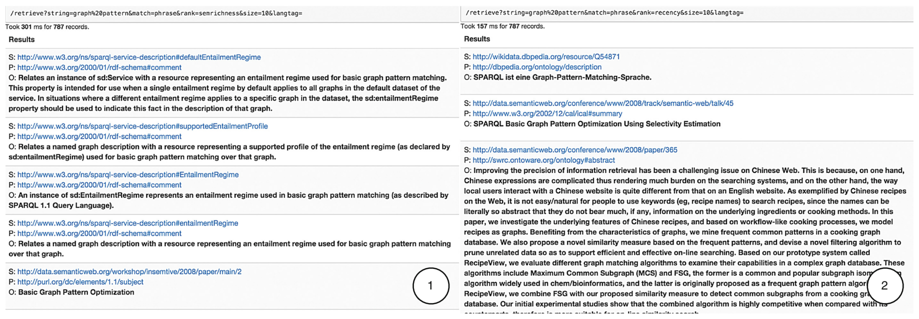


Fig. 2. LOTUS average response times in seconds for bi- tri- and quadgram requests. The horizontal axis shows 18 combinations of a matcher and a ranker. The matchers are *conjunct* (c), *phrase* (p) and *terms* (t). The rankers are *length normalization* (l), *proximity* (pr), *psf* (ps), *recency* (r), *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 s for bigrams, 0.15 s for trigrams, 0.15 s for quadgrams and 0.5 s for all three combined. The slowest query is for bigrams with terms matching and length normalization, which takes 1.0 s on average.



```
/retrieve?string=graph%20pattern&match=phrase&rank=semrichness&size=10&langtag=
```

Took **301** ms for **787** records.

Results

S: http://www.w3.org/ns/sparql-service-description#defaultEntailmentRegime
P: http://www.w3.org/2000/01/rdf-schema#comment
O: **Relates an instance of sd:Service with a resource representing an entailment regime used for basic graph pattern matching. This property is intended for use when a single entailment regime by default applies to all graphs in the default dataset of the service. In situations where a different entailment regime applies to a specific graph in the dataset, the sd:entailmentRegime property should be used to indicate this fact in the description of that graph.**

S: http://www.w3.org/ns/sparql-service-description#supportedEntailmentProfile
P: http://www.w3.org/2000/01/rdf-schema#comment
O: **Relates a named graph description with a resource representing a supported profile of the entailment regime (as declared by sd:entailmentRegime) used for basic graph pattern matching over that graph.**

S: http://www.w3.org/ns/sparql-service-description#EntailmentRegime
P: http://www.w3.org/2000/01/rdf-schema#comment
O: **An instance of sd:EntailmentRegime represents an entailment regime used in basic graph pattern matching (as described by SPARQL 1.1 Query Language).**

S: http://www.w3.org/ns/sparql-service-description#entailmentRegime
P: http://www.w3.org/2000/01/rdf-schema#comment
O: **Relates a named graph description with a resource representing an entailment regime used for basic graph pattern matching over that graph.**

S: http://data.semanticweb.org/workshop/insemtive/2008/paper/main/2
P: http://purl.org/dc/elements/1.1/subject
O: **Basic Graph Pattern Optimization**

1

```
/retrieve?string=graph%20pattern&match=phrase&rank=recency&size=10&langtag=
```

Took **157** ms for **787** records.

Results

S: http://wikidata.dbpedia.org/resource/Q54871
P: http://dbpedia.org/ontology/description
O: **SPARQL ist eine Graph-Pattern-Matching-Sprache.**

S: http://data.semanticweb.org/conference/www/2008/track/semantic-web/talk/45
P: http://www.w3.org/2002/12/cal/ical#summary
O: **SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation**

S: http://data.semanticweb.org/conference/www/2008/paper/365
P: http://swrc.ontoware.org/ontology#abstract
O: **Improving the precision of information retrieval has been a challenging issue on Chinese Web. This is because, on one hand, Chinese expressions are complicated thus rendering much burden on the searching systems, and on the other hand, the way local users interact with a Chinese website is quite different from that on an English website. As exemplified by Chinese recipes on the Web, it is not easy/natural for people to use keywords (eg, recipe names) to search recipes, since the names can be literally so abstract that they do not bear much, if any, information on the underlying ingredients or cooking methods. In this paper, we investigate the underlying features of Chinese recipes, and based on workflow-like cooking processes, we model recipes as graphs. Benefiting from the characteristics of graphs, we mine frequent common patterns in a cooking graph database. We also propose a novel similarity measure based on the frequent patterns, and devise a novel filtering algorithm to prune unrelated data so as to support efficient and effective on-line searching. Based on our prototype system called RecipeView, we evaluate different graph matching algorithms to examine their capabilities in a complex graph database. These algorithms include Maximum Common Subgraph (MCS) and FSG, the former is a common and popular subgraph isom algorithm widely used in chem/bioinformatics, and the latter is originally proposed as a frequent graph pattern algor RecipeView, we combine FSG with our proposed similarity measure to detect common subgraphs from a cooking gra database. Our initial experimental studies show that the combined algorithm is highly competitive when compared with**

2

Fig. 3. Results of query "graph pattern" with terms-based matching and different rankings: (1) Semantic richness, (2) Recency.

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.

7 Discussion and Conclusions

In this paper, we presented LOTUS, a full-text entry point to the centralized LOD Laundromat collection. 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, taking into account both textual similarity and certain structural properties of the underlying data. LOTUS currently provides 32 retrieval options to be used in different use cases. Even though LOTUS is to some extent “connecting the dots” between the underlying infrastructures of LOD Laundromat and ElasticSearch, it is connecting a lot of these dots and due to the billions of statements that we are dealing with some of the connections are non-trivial.

In the current version of LOTUS we focus on context-free[17] ranking of results and demonstrate 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. To some extent, context-dependent functionality could be built into LOTUS. However, graph-wide integration with structured data would require a different approach, potentially based on a fulltext-enabled triplestore (e.g. Virtuoso).

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

Acknowledgments. The research for this paper was supported by the European Union’s 7th Framework Programme via the NewsReader Project (ICT-316404) and the Netherlands Organisation for Scientific Research (NWO) via the Spinoza fund.

References

1. Antoniou, G., Groth, P., van Harmelen, F., Hoekstra, R.: A Semantic Web Primer, 3rd edn. The MIT Press, Cambridge (2012)
2. Beek, W., Rietveld, L.: Frank: algorithmic access to the LOD cloud. In: Proceedings of the ESWC Developers Workshop (2015)
3. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: Lod laundromat: a uniform way of publishing other peoples dirty data. ISWC **2014**, 213–228 (2014)
4. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.-Y.: SPARQL web-querying infrastructure: ready for action? In: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J.X., Aroyo, L., Noy, N., Welty, C., Janowicz, K. (eds.) ISWC 2013, Part II. LNCS, vol. 8219, pp. 277–293. Springer, Heidelberg (2013)
5. Cheng, G., Ge, W., Qu, Y.: Falcons: searching and browsing entities on the semantic web. In: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, NY, USA, pp. 1101–1102 (2008). http://doi.acm.org/10.1145/1367497.1367676

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

6. Christophides, V., Efthymiou, V., Stefanidis, K.: Entity Resolution in the Web of Data. Morgan and Claypool Publishers, San Rafael (2015)
7. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax (2014)
8. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. In: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management, CIKM 2004, NY, USA, pp. 652–659 (2004). http://doi.acm.org/10.1145/1031171.1031289
9. Feyznia, A., Kahani, M., Zarrinkalam, F.: Colina: a method for ranking sparql query results through content and link analysis. In: Proceedings of the 2014 International Conference on Posters & Demonstrations Track, ISWC-PD 2014, CEUR-WS.org, Aachen, Germany, vol. 1272, pp. 273–276 (2014). http://dl.acm.org/citation.cfm?id=2878453.2878522
10. Hogan, A., Harth, A., Umbrich, J., Kinsella, S., Polleres, A., Decker, S.: Searching and browsing linked data with swse: the semantic web search engine. Web Semant. Sci. Serv. Agents World Wide Web **9**(4), 365–401 (2011). JWS special issue on Semantic Search. www.sciencedirect.com/science/article/pii/S1570826811000473
11. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An empirical survey of linked data conformance. Web Semant. Sci. Serv. Agents World Wide Web **14**, 14–44 (2012)
12. Ichinose, S., Kobayashi, I., Iwazume, M., Tanaka, K.: Ranking the results of DBpedia retrieval with SPARQL query. In: Kim, W., Ding, Y., Kim, H.-G. (eds.) JIST 2013. LNCS, vol. 8388, pp. 306–319. Springer, Heidelberg (2014)
13. Ilievski, F., Beek, W., van Erp, M., Rietveld, L., Schlobach, S.: Lotus: linked open text unleashed. In: COLD workshop, ISWC (2015)
14. Lei, Y., Uren, V.S., Motta, E.: SemSearch: a search engine for the semantic web. In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS (LNAI), vol. 4248, pp. 238–245. Springer, Heidelberg (2006)
15. Mulay, K., Kumar, P.S.: Spring: ranking the results of sparql queries on linked data. In: Proceedings of the 17th International Conference on Management of Data, COMAD 2011, Computer Society of India, Mumbai, India, pp. 12:1–12:10 (2011). http://dl.acm.org/citation.cfm?id=2591338.2591350
16. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. (CSUR) **33**(1), 31–88 (2001)
17. 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)
18. Tran, T., Wang, H., Haase, P.: Hermes: data web search on a pay-as-you-go integration infrastructure. Web Semant. Sci. Serv. Agents World Wide Web **7**(3), 189–203 (2009). www.sciencedirect.com/science/article/pii/S1570826809000213. The Web of Data
19. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: weaving the open linked data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 552–565. Springer, Heidelberg (2007)
20. Van Herwegen, J., De Vocht, L., Verborgh, R., Mannens, E., Van de Walle, R.: Substring filtering for low-cost linked data interfaces. In: Arenas, M. (ed.) ISWC 2015. LNCS, pp. 128–143. Springer, Switzerland (2015)

21. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the web with high availability. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 180–196. Springer, Heidelberg (2014)
22. 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). www.sciencedirect.com/science/article/pii/S1570826809000262. The Web of Data