RDF Triple Stores and a Custom SPARQL Front-End for Indexing and Searching (Very) Large Semantic Networks

Milen Kouylekov[♣] and Stephan Oepen[♣♡]
[♣] University of Oslo, Department of Informatics
[♡] Potsdam University, Department of Linguistics
{milen|oe}@ifi.uio.no

Abstract

With growing interest in the creation and search of linguistic annotations that form general graphs (in contrast to formally simpler, rooted trees), there also is an increased need for infrastructures that support the exploration of such representations, for example logical-form meaning representations or semantic dependency graphs. In this work, we lean heavily on semantic technologies and in particular the data model of the Resource Description Framework (RDF) to represent, store, and efficiently query very large collections of text annotated with graph-structured representations of sentence meaning. Our full infrastructure is available under open-source licensing, and through this system demonstration we hope to receive feedback on the general approach, explore its application to additional types of meaning representation, and attract new users and possibly co-developers.

1 Motivation: The Problem

Much work in the creation and use of language resources has focused on *tree*-shaped data structures,[1] as are commonly used for the encoding of, for example, syntactic or discourse annotations. Conversely, there has been less focus on supporting general *graphs* until recently, but there is growing interest in graph-structured representations, for example to annotate and process natural language semantics. In this work, we demonstrate how *semantic technologies*, and in particular the data model of the Resource Description Framework (RDF) can be put to use for efficient indexing and search in (very) large-scale collections of semantic graphs.

We develop a mapping to RDF graphs for a variety of semantic representations, ranging from underspecified logical-form meaning representations to 'pure' bi-lexical semantic dependency graphs, as exemplified in Figures 1 and 2 below, respectively. Against this uniform data model, we populate off-the-shelf RDF triple stores with semantic networks comprising between tens of thousands and tens of millions of analyzed sentences. To lower the technological barrier to exploration of our triple stores, we implement a compact 'designer' query language for semantic graphs through on-the-fly expansion into SPARQL. In sum, the combination of standard RDF technologies and specialized query and visualization interfaces yields a versatile and highly scalable infrastructure for search (and in principle limited forms of reasoning) over diverse types of graph-structured representations of sentence meaning.

In our view, there is little scientific innovation in this work, but our approach rather demonstrates substantial design and engineering creativity. Our semantic search infrastructure is built from the combination of industrial-grade standard technologies (Apache Jena, Lucene, and Tomcat) with an open-source application for, among others, format conversion, query processing, and visualization implemented in Java. Thus, the complete tool chain is available freely and across platforms. Its application to additional types of meaning representation (and possibly other graph-structured layers of linguistic analysis) should be relatively straightforward, and we thus believe that our infrastructure can be of immediate value to both providers and consumers of large-scale linguistic annotations that transcend tree structures.

[1] Formally, trees are a restricted form of graphs, where every node is reachable from a distinguished root node by exactly one directed path.

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations, pages 90–94, Dublin, Ireland, August 23-29 2014.

$\langle h_1,$
h_4:_a_q(x_6, h_7, h_5), h_8:_similar_a_to(e_9, x_6), h_8:comp(e_{11}, e_9, _), h_8:_technique_n_1(x_6),
h_2:_almost_a_1(e_{12}, h_{13}), h_{14}:_impossible_a_for(e_3, h_{15}, i_{16}),
h_{17}:_apply_v_to(e_{18}, i_{19}, x_6, x_{20}), h_{21}:udef_q(x_{20}, h_{22}, h_{23}), h_{24}:_other_a_1(e_{25}, x_{20}), h_{24}:_crop_n_1(x_{20}),
h_{24}:_such+as_p(e_{26}, x_{20}, x_{27}), h_{40}:implicit_conj(x_{27}, x_{33}, x_{38}),
h_{31}:udef_q(x_{33}, h_{32}, h_{34}), h_{35}:_cotton_n_1(x_{33}), h_{46}:_and_c(x_{38}, x_{43}, x_{47}),
h_{41}:udef_q(x_{43}, h_{42}, h_{44}), h_{45}:_soybean_u_unknown(x_{43}), h_{48}:udef_q(x_{47}, h_{49}, h_{50}), h_{51}:_rice_n_1(x_{47})
$\{ h_{49} =_q h_{51}, h_{42} =_q h_{45}, h_{32} =_q h_{35}, h_{22} =_q h_{24}, h_{15} =_q h_{17}, h_{13} =_q h_{14}, h_7 =_q h_8, h_1 =_q h_2 \} \rangle$

Figure 1: Example logical form meaning representation (MRS; taken from DeepBank).

2 Technology: Core Components

Our system architecture comprises two core components, viz. (a) the RDF repository, a database storing semantic networks in RDF triple form, and (b) the Web application, an interface for interactive search and visualization over the RDF repository.

Representing Semantic Graphs in RDF The RDF data model is based on statements about resources in the form of *subject–predicate–object* triples. The subject denotes the resource, and the predicate denotes traits or aspects of the resource, thus expressing a relationship between the subject and the object. A database that can store such expression and evaluate queries to them is called a triple store.

In Kouylekov and Oepen (2014), we describe the conversion of different types of semantic structures into RDF graphs. To date, we have addressed three types of meaning representations, viz. (in decreasing complexity) (a) scope-underspecified logical formulas in *Minimal Recursion Semantics* (MRS; Copestake et al., 2005); (b) variable-free *Elementary Dependency Structures* (EDS; Oepen and Lønning, 2006); and (c) bi-lexical dependency graphs as used in Task 8 at SemEval 2014 on *Broad-Coverage Semantic Dependency Parsing* (SDP; Oepen et al., 2014; Ivanova et al., 2012). For all three formats, we draw on (a) gold-standard annotations from DeepBank (Flickinger et al., 2012), a re-annotation of the venerable Penn Treebank WSJ Corpus (Marcus et al., 1993); and on (b) much larger collections of automatically generated analyses over the full English Wikipedia from the WikiWoods Treecache (Flickinger et al., 2010).

To store MRS, EDS, and SDP structures, we created small ontologies for each type of representation, building on a common core of shared ontology elements. In a nutshell, the EDS and SDP ontologies provide a generic representation of directed graphs with (potentially complex) node and edge labels; the dependencies proper, i.e. labeled arcs of the graph, are encoded as RDF object properties. The MRS ontology, on the other hand, distinguishes different types of nodes, corresponding to full predications vs. individual logical variables vs. hierarchically organized sub-properties of variables. Mapping the (medium-complexity) EDS graphs from DeepBank and WikiWoods onto RDF, for example, yields around 12 million and 4.3 billion triples, respectively (for the semantic dependencies of about 37 thousand and 48 million sentences in the two resources).

Web Application The core of our Web application is a search engine that executes SPARQL queries against the RDF repository. SPARQL is an RDF query language to search triple stores, allowing one to retrieve and manipulate RDF data. It is fully standardized and considered one of the key technologies of the Semantic Web. A SPARQL query can consist of triple patterns, conjunctions, disjunctions, and optional filters and functions. The query processor searches for sets of triples that match the patterns expressed in the query, binding variables in the query to the corresponding parts of each triple.

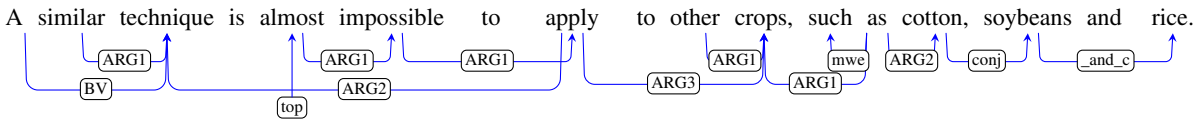

Figure 2: Example bi-lexical semantic dependencies (SDP; taken from DeepBank).

```
PREFIX sdp:<http://wesearch.delph-in.net/rdf/sdp#>
PREFIX dm:<http://wesearch.delph-in.net/rdf/sdp/dm#>
select ?graph
where {
 GRAPH ?graph {
  ?101 sdp:form "quarterly"^^xsd:string .
  ?x dm:lemma "result"^^xsd:string .
  {
   ?100 dm:pos "vbp"^^xsd:string
   UNION ?100 dm:pos "vbg"^^xsd:string
   UNION ...
  }
  ?101 dm:arg1 ?x .
  {
   ?100 dm:arg1 ?x UNION ?100 dm:arg2 ?x
   UNION ?100 dm:arg3 ?x UNION ?100 dm:arg4
  }
  FILTER
  ((!bound(?101) || !bound(?100) || ?101 != ?100)
   && (!bound(?101) || !bound(?x) || ?101 != ?x)
   && (!bound(?100) || !bound(?x) || ?100 != ?x))
 }
}
GROUP BY ?graph
ORDER BY ?graph
```

Figure 3: Core of the auto-generated SPARQL query corresponding to our running example.

Our infrastructure supports the definition of families of 'meta' query languages, to address semantic structures in a form that is more compact and much better adapted to the specific target format than SPARQL. An example of such a 'designer' language is the WeSearch Query Language (WQL), which was used in the context of the SemEval 2014 SDP task.[2] By way of informal introduction, consider the following example query:

(1)
```
/v*[ARG* x]
quarterly[ARG1 x]
x:+result
```

This example is comprised of three *predications*, one per line. The following characters have *operator* status: '`/`' (slash), '`*`' (asterisk), '`[`' and '`]`' (left and right square bracket), '`:`' (colon), and '`+`' (plus sign). This is a near-complete list of operator characters in WQL. Each predication can be composed of (i) an *identifier*, followed by a colon if present; (ii) a *form* pattern; (iii) a *lemma* pattern, prefixed by a plus sign, if present; (iv) a *part-of-speech* (PoS) pattern, prefixed by a slash, if present; and (v) a list of *arguments*, enclosed in square brackets, if present. Patterns can make use of Lucene-style wildcards, with the asterisk matching any number of characters, and a question mark ('`?`') to match a single character.

Thus, our example query searches for a verbal predicate (any PoS tag starting with '`v`'), that takes any form of the lemma '`result`' as its argument (in the range $\texttt{ARG}_1 \ldots \texttt{ARG}_n$), where this argument is further required to be the $\texttt{ARG}_1$ of a node labeled '`quaterly`'.

The auto-generated SPARQL expression that corresponds to this example query is shown in Figure 3. The query generator replaces the wildcarded PoS pattern by the union of all matching tags (that start with '`v`', e.g. '`?100 dm:pos "vbp" UNION ...`' Likewise, the underspecified argument relation of this predication is replaced by the union of all possible argument types. Finally, the query processor ensures a one-to-one correspondence between query elements and matching graph elements, i.e. multiple distinct query components cannot match against the same target (graph component), or vice versa. This is accomplished in SPARQL through the filter expressions towards the end of the generated query.

[2] See `http://wesearch.delph-in.net/sdp/` for an on-line demonstration and additional documentation.

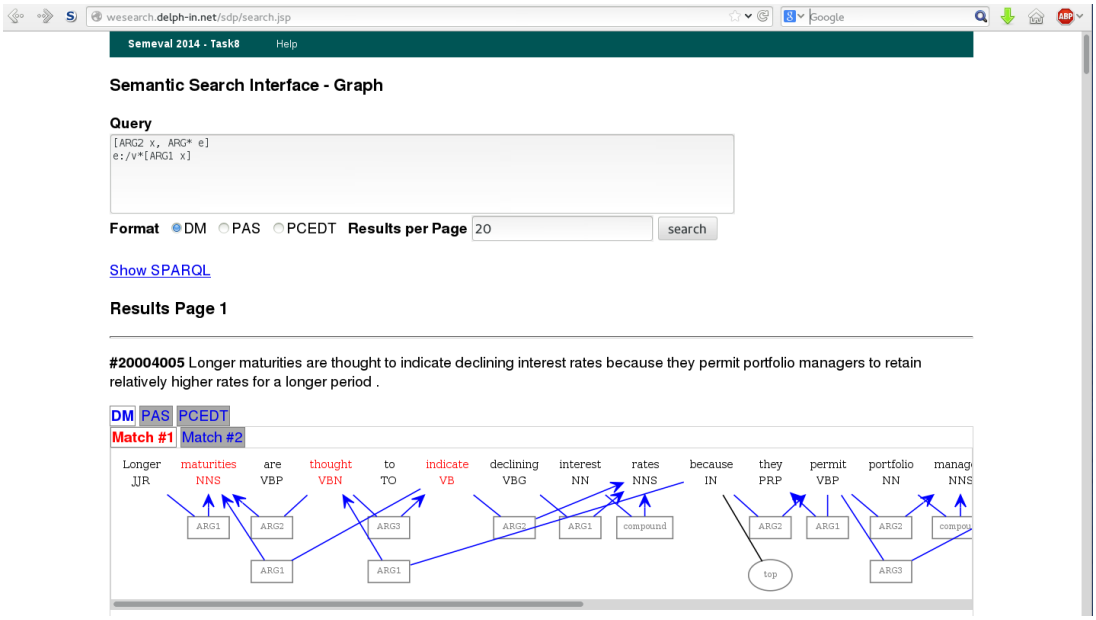

Figure 4: Screenshot of the interactive search interface, querying (semantic) object control structures.

Figure 4 shows a screenshot from the SemEval SDP user interface, demonstrating how WQL facilitates concise (and reasonably transparent) search for semantic 'object' control, i.e. a configuration involving two predicates sharing an argument in a specific assignment of roles.

Within the capabilities of the SPARQL back-end, different dialects of the meta query language can be implemented in a modular fashion, for example distinguishing different types of nodes and introducing additional node properties, as in the more complex MRS universe. Our query front-end transforms 'meta' queries into equivalent SPARQL expressions, and the search interface allows users to inspect the result of this transformation (and matching results), to possibly refine the search incrementally either at the 'meta' query layer or directly in SPARQL.

3 Demonstration: Indexing and Search

Our proposed interactive demonstration will seek to highlight (a) the flexibility of our infrastructure, i.e. walk through a series of queries of increasing complexity against different target formats; (b) its scalability, by comparing response times for different types of queries and different target formats over the large DeepBank and the vast WikiWoods indexes; and (c) the ease of 'behind the scenes' functionality, showing how additional semantic annotations in various formats can be ingested into the index. As part of this latter aspect of the demonstration, we will optionally discuss how we apply string-level indexing (in Apache Lucene) and basic frequency statistics in query interpretation and optimization, which jointly with parallelization over 'striped' RDF triple stores can yield greatly reduced response times for common types of queries to the WikiWoods index. We envision that parts of the demonstration can be organized in an audience-driven manner, for example taking as input the (possibly informal) characterization of a semantic configuration, collectively transforming it into a query against our DeepBank or WikiWoods stores, observing linguistic or technical properties of matching results, and refining the search incrementally.

Our software infrastructure is entirely open-source and (increasingly) modularized and parameterized to facilitate adaptation to additional types of annotation. Please see the project web page for licensing and access information, as well as for pointers to a variety of existing on-line demonstrations:

`http://wesearch.delph-in.net/`

References

Copestake, A., Flickinger, D., Pollard, C., and Sag, I. A. (2005). Minimal Recursion Semantics. An introduction. *Research on Language and Computation*, *3*(4), 281 – 332.

Flickinger, D., Oepen, S., and Ytrestøl, G. (2010). WikiWoods. Syntacto-semantic annotation for English Wikipedia. In *Proceedings of the 7th International Conference on Language Resources and Evaluation.* Valletta, Malta.

Flickinger, D., Zhang, Y., and Kordoni, V. (2012). DeepBank. A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories* (p. 85 – 96). Lisbon, Portugal: Edições Colibri.

Ivanova, A., Oepen, S., Øvrelid, L., and Flickinger, D. (2012). Who did what to whom? A contrastive study of syntacto-semantic dependencies. In *Proceedings of the Sixth Linguistic Annotation Workshop* (p. 2 – 11). Jeju, Republic of Korea.

Kouylekov, M., and Oepen, S. (2014). Semantic technologies for querying linguistic annotations. An experiment focusing on graph-structured data. In *Proceedings of the 9th International Conference on Language Resources and Evaluation.* Reykjavik, Iceland.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpora of English: The Penn Treebank. *Computational Linguistics*, *19*, 313 – 330.

Oepen, S., Kuhlmann, M., Miyao, Y., Zeman, D., Flickinger, D., Hajič, J., ... Zhang, Y. (2014). SemEval 2014 Task 8. Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation.* Dublin, Ireland.

Oepen, S., and Lønning, J. T. (2006). Discriminant-based MRS banking. In *Proceedings of the 5th International Conference on Language Resources and Evaluation* (p. 1250 – 1255). Genoa, Italy.