SPARQL/T

A query language with SPARQL's syntax for semantic mining of textual complaints

Bruno Quintavalle†
Università Ca' Foscari
Venezia, Italy
bruno.quintavalle@unive.it

Salvatore Orlando
Università Ca' Foscari
Venezia, Italy
orlando@unive.it

ABSTRACT

Extracting information from complaints, either scraped from the Web or received directly from the client, is a necessity of many companies nowadays. The aim is to find inside them some actionable knowledge. To this purpose, verbal phrases must be analyzed, as many complaints refer to actions improperly performed. The Semantic Roles of the actions (who did what to whom) and the Named Entities involved need to be extracted. Moreover, for the correct interpretation of the claims, the software should be able to deal with some background knowledge (for example, a product's ontology). Although there are already many libraries and out of the shelf tools that allow tackling these problems singularly, it may be hard to find one that includes all the needed tasks. We propose here a query language that adopts the syntax of SPARQL to extracts information from natural language documents, pre-annotated with NLP information. The language provides the user with a simple and uniform interface to the most useful NLP tasks, isolating him or her from the details of the specific implementation. We argue that a query language is much easier and intuitive (from a laymen point of view) than an imperative one. Moreover, the adoption of the SPARQL syntax allows to seamlessly mix, inside the same query, NLP patterns with traditional RDF/OWL ones, simplifying the integration with Semantic Web technologies.

KEYWORDS

SPARQL, Information Extraction, Complaints, Forums, Semantic Web

1. INTRODUCTION

Complaints are usually convoluted descriptions of complex problems. To correctly retrieve them and extract from them all the interesting details, the structure of the sentence cannot be disregarded.

Special attention must be given to verbal phrases, as complaints are usually descriptions of actions that have been performed whilst they shouldn't (or the other way around). From most actions we probably need to extract at least the performer and the object (Semantic Role Labeling), to be tracked along the discourse (Co-reference Resolution). Sometimes we may need to refer to very specific entities (products, services, companies), whose name can be highly ambiguous and hard to detect precisely (Named Entity Recognition). Other times we may want to refer to concepts in general terms, allowing synonyms (WordNet) or trusting some measure of similarity (Word Embedding). Finally, some background knowledge may also need to be considered, like for example ontologies that describe products and services, with their reasonable ranges of prices, performances, delivery times and so on.

It is often hard to find a library or tool that cover all the necessary tasks, and this sometimes forces the user to also employ different programming languages. Moreover, imperative programming languages like Python or Java requires specific skills, which may rule out many potentially interested users. The solution we propose is the query language SPARQL/T (SPARQL over Text) that adopts the syntax of the popular SPARQL query language (avoiding the introduction of another language or dialect), but that acts not only on RDF graphs, but also directly on texts by exploiting their NLP annotations. NLP tasks are available in the form of specific Triple Patterns (TPs), recognizable by the prefix of the predicate function. This provides a level of abstraction from the tools or libraries actually involved in the extraction. Having the same syntax, TP involving NLP task can be seamlessly intermixed with the traditional ones that refer to RDF triples and graphs, thus allowing us to design a SPARQL-like language that supports the so-called Hybrid Queries. Another important feature of SPARQL/T is its ability to deal with uncertainty and similarity. Uncertainty comes from the well-known ambiguity of natural languages, and vector similarity by the exploitation of Word and Sentence Embeddings.

In general, a query language is expected to be more intuitive and easier to use than an imperative one, as it focuses on what needs to be extracted instead on how to extract it. SPARQL/T specifically aims at making Information Retrieval and Extraction from complaints a task achievable by almost anyone, possibly with the aid of a suitable User Interface. Moreover, although we focused on company complaints, indeed SPARQL/T can be used to other form of text documents that present similar degrees of complexity, like for example clinical narratives [Zhang. et al. 2018].

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 the SPARQL/T based on NLP techniques, and Section 4 the Hybrid Queries allowed by SPARQL/T. Section 5 the role of relation algebra to process queries, while Section 6 summarizes the function currently implemented in SPARQL/T. Finally, Section 7 and 8 discuss the system architecture and some performance figures, and Section 9 draws some conclusions and future work.

2. RELATED WORK

The related work considered here are of two kinds. First, we explore an obvious alternative to SPARQL/T: extract all possible useful triples from the documents into a triple store, and then employ a standard SPARQL engine. Second, we have a look at other tools that employ SPARQL's syntax for purposes like ours.

2.1 Knowledge Extractors tools

Knowledge Extractors (KE) tools transform Natural Language documents into machine-interpretable formats, often into RDF/OWL graphs than can be stored into standard triple stores (and thus efficiently indexed) and queried in standard SPARQL

FRED [8] automatically generates RDF/OWL ontologies from (multilingual) natural language text. It employs Named Entity Recognition (NER) to link its output to semantic web knowledge and Word Sense Disambiguation (WSD) to align with WordNet and BabelNet. Among FRED points of strength is its ability to represent the structure of the discourse, according to the Discourse Representation Theory [9].

PIKES [6] extracts entities and relations between them by identifying semantic frames, i.e., events and situations describing n-ary relations between entities. In the resulting knowledge graph, each node uniquely identifies an entity of the world, event or situation, and arcs represent relations between them. The PIKES tool implements a rule-based knowledge distillation technique using SPARQL-like rules formulated as SPARQL Update INSERT. . . WHERE. . . statements that are repeatedly executed until a fixed-point is reached.

OpenIE is an Information Extraction paradigm that aims at avoiding human intervention like hand crafted extraction rules or large hand annotated training sets [2]. See [10] for a recent survey of the different implementations. The result of an OpenIE extraction is a set of triples of strings *(subject, predicate, object)*, a textual approximation to an entity-relationship graph called Extraction Graph [5]. The elements of an Extraction Graph are just strings. Many entities and relations may appear in different forms ("Einstein" / "Albert Einstein"). No effort is spent to relate entities to some ontology, nor to put relations into a canonical form (like `invented(X, Y)`). Also, it is accepted that the extractor makes errors, and inconsistent information contained in the source text is not tried to be solved. However, a confidence degree of each triple is calculated based on the number of times it has been extracted from the corpus.

Compared with SPARQL/T approach, pre-extracted knowledge can be indexed, and thus has the advantage of speed. However, the KE task is still a very difficult one. Similar documents do not always result in structurally similar graphs as desired, making it difficult to write queries with reasonable recall. With SPARQL/T similarity-based approach it is much easier to achieve good recall, although precision often suffers. However, KE results, and in particular OpenIE triples, are going to be implemented in the future release of SPARQL/T.

2.2 Tools that employ SPARQL syntax

iDocument [1] is an Ontology Based Information Extraction tool (OBIE) that employs SPARQL syntax in the extraction templates in place of the traditional regular expressions. The annotations, that are pre-extracted by a NLP pipeline, are potentially quite rich, and include Named Entity Recognition, Structured Entity Recognition, Fact Extraction and Scenario Extraction. iDocument is perhaps the tool closer to ours due to the adoption of the SPARQL syntax for Information Extraction using templates.

QLever SPARQL+Text [4] is another tool that employs SPARQL syntax. It allows to efficiently search on text corpus combined with an RDF knowledge base. It only considers Named Entities annotations, that are linked to some Knowledge Bases (Freebase Easy [3], Clue-Web 2012). QLever can mix standard SPARQL triple patterns, referring to the knowledge base, with others that can reference the text and its NE annotations (with two built-in predicates: `ql:contains-entity` and `ql:contains-word`). QLever approach for joining results employs the notion of co-occurrence: the results of each triple pattern are joined when they occur inside the same text segment (i.e. a crispy version of SPARQL/T approach). Different kinds of text segmentations are expected to give different results.

Mìmir [13] is an open-source framework for integrated semantic search over text, document structure, linguistic annotations, and formal semantic knowledge. It allows search constraints against a knowledge base, by accessing at run time a predefined SPARQL endpoint.

All these tools share aims and ideas with SPARQL/T. However, none of them seem to be able to deal with similarity and uncertainty.

3 PURE NLP QUERIES

To illustrate the use of SPARQL/T, we start with a query that addresses only the text and its NLP annotations (i.e. it does not refer to any ontology). Let our information need be to find mentions of the following concept: "*A company has increased the cost of its services without the customer's knowledge*".

This complex concept can be split into two parts:

1. *Increase the price of something*
2. *Not knowing something*

The query in Figure 1 shows a possible approach, with the two parts extracted separately with two groups of triple patterns enclosed in curly braces (i.e.: two Basic Graph Patterns, or BGP in SPARQL terminology). In SPARQL/T everything that is specified inside a BGP must be found inside the same sentence of the document (according to the sentence splitting available in the annotations). The first part is further broken down into four components, extracted into four variables:

1. `who` (a company name)
2. `inc` (a verb like '*increase*' or '*rise*')
3. `pri` (a word like '*price*' or '*cost*', which must be the subject of `inc`)
4. `ser` (the name of the service)

About the second part, we only care about its presence. Moreover, notice that its negation may be stated just implicitly, for example with a sentence like "*I only discovered that in the bill*". For these reasons, for part 2 we rely on Sentence Embedding, using the variable `knw`.

	SPARQL/T Query	NLP Task
1	SELECT ?who ?ser WHERE {	
2	{ ?who NLP:NER "ORGANIZATION" .	NER
3	?inc EMB:ANY "increase raise".	Word Emb.
4	?inc NLP:POS "VERB" .	POS Tag.
5	?inc DEP:DOBJ ?what	Dep. Par.
6	?pri EMB:ANY "price cost" .	Word Emb.
7	?ser NLP:NER "SERVICE" .	NER
8	}	
9	{ ?knw SEN:ANY "I know" } }	Sentence Embedding

Figure 1: A SPARQL/T query that looks for organization that increase the cost of their services without notifying the clients

Notice that, when compared with a traditional SPARQL queries, the query in Figure 1 can be seen as acting on a virtual graph that has partially been extracted into various NLP annotations, but that is still partially embedded inside the text. Figure 2 depicts this idea, i.e.: Text + Annotations ≅ Virtual Graph.

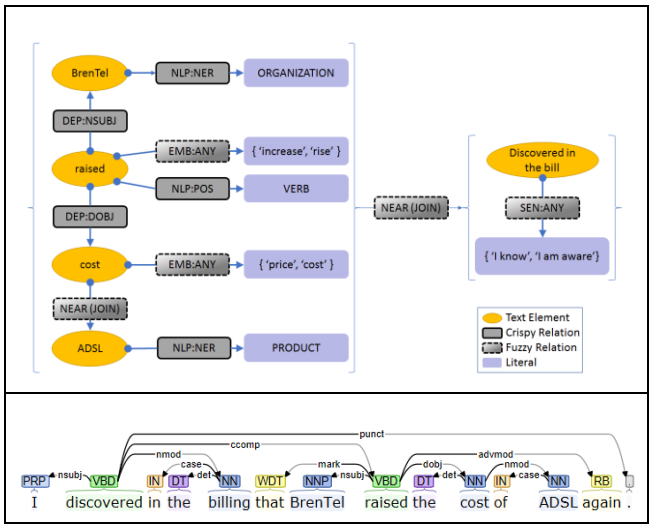

Figure 2: A SPARQL/T query acts at the same time on the text and on its annotations. Together they may be seen as a sort of virtual semantic graph, only partially extracted. The query in Figure 1 for example, if interpreted in SPARQL instead of SPARQL/T, would perfectly match the RDF graph in the upper part of the figure, which can be seen as the virtual graph of the sentence below[1].

4 HYBRID QUERIES

SPARQL/T allows to mix NLP BGP with standard RDF ones. The latter are redirected, exactly as they are, to Apache Jena[2], together with the values of the variables previously extracted. On return, the Jena results are joined to form a single relation. An example of hybrid query is illustrated in Figure 3.

5 RELATIONAL ALGEBRA

SPARQL/T Triple Patterns (TPs) are of two kinds: the NLP ones, that extracts snippets of text and URIs from the document and its annotations, and the RDF ones, that exactly as in the SPARQL case extracts URIs and Literals from an RDF graph. Both kinds of TPs return a relation, i.e., a table of elements that can have up to three columns (one for each variable in the TP). The elements of the columns however, may represent three different things, depending on the kind of TPs: snippets of text (a range S = [begin, end] of tokens extracted), URI U and literals L[3].

```
pref tel: <http://sparqlt.com/ontology/tel>
SELECT ?sen ?off ?pri
WHERE {
 { ?sen LEM:ANY 'love like hate'.
   ?off NLP:EL 'TEL_OFFER'.
 }
 { ?off tel:hasUnlimited tel:social .
   ?off tel:monthlyRate  ?pri .
 }
}
```

Figure 3: SPARQL/T Hybrid Query that returns a table of three columns: a word that express a sentiment (`sen`), the name of a telephone offer (`off`) and its price (`pri`). Here the first two TPs extracts things from the documents and its annotation, whilst the last two refers to an RDF/OWL ontology (not shown). Specifically, the `LEM:ANY` function extracts from the text any word whose lemma is in the list {*love, like, hate*}, whilst `TEL_OFFER` is the id of a class of Named Entities, pre-annotated by an Entity Linking tool and extracted with the `NLP:EL` function. `sen` and `off` are required to be inside the same sentence, and the score is inversely proportional to their distance. The last two TPs are passed to a SPARQL endpoint (Apache Jena), together with the values of `off` found in the text. The distinction between NLP and RDF TPs is made using the prefixes: `LEM` and `NLP` are reserved SPARQL/T prefixes. An URIs with any other prefix, or without prefix at all, is assumed to refer to the RDF/OWL graph.

Moreover, each row of the relations has a score τ that indicates the degree of truth of the extraction. τ is in the range [0,1], and can be for example a *similarity* measure in case of Word Embeddings, a degree of *confidence* in case of Named Entity Recognition or Entity Linking, or be simply equal to 1 for a crispy TP.

[1] Dependency tree obtained with Stanford CoreNLP 3.9.2 https://corenlp.run/
[2] https://jena.apache.org/

[3] In the case of NLP TPs, S is always present, L is the concatenation of the tokens in S, whilst U is only present in case of Named Entity or Entity Linking TPs. In the case of RDF TPs, S is always empty, the URI are stored into U and the literals into L.

Relations extracted by TP are combined using Relational Algebra operators similar to those adopted in SPARQL (see [7] for a theoretical exposition, and the W3C recommendations for the actual approach). The main difference is that in SPARQL/T we need to combine the score τ of the rows of the relations, considering both the scores of the source rows and the way their elements relates to each other (proximity, co-occurrences, …). For example, in combining rows that share variables (NATURAL JOIN operation), we cannot rely on exact match between elements, because snippets of text extracted by different annotation algorithms are unlikely to be exactly the same, and strings of the document may not match exactly the literals in the ontology. A similarity measure is used instead, calculated in a way that depends on the values available. Priority is given to URI: if they are present in both elements a crispy match is performed. Otherwise, when present, snippets of text are compared, in terms of their degree of overlapping. Finally, when comparing RDF literals with text, robustness is sought through an edit distance function (although this practice is discouraged, it allows to search inside the document literals stored in an ontology. NER should be used instead, whenever possible).

The score of the combination of a couple of rows R_1 and R_2 is calculated using a Fuzzy Logic approach: if τ_{R1} and τ_{R2} are the scores of the two rows and (τ_{V1}, … τ_{VN}) are the scores of each couple of elements involved in the join operation (i.e. pertaining to each common variable V), the score of the output row T_O is:

$$\tau_O = \tau_{R1} \otimes \tau_{R2} \otimes \tau_{V1} \otimes \cdots \otimes \tau_{VN}$$

where $\otimes$ is any Fuzzy t-norm, typically the *min()* function (see for example [12])

For the CROSS-PRODUCT operation, i.e. when the two relations share no common variables, the output relation is formed by all the possible couples of rows from the two input ones, scored according to their distance in the document. We thus assume that the closer two concepts are expressed in the documents, the highest are the chances that they are related. However, CROSS PRODUCT operations potentially lead to the exponentially growth of the relations' size, especially in those cases that involves similarity measures (as any word is similar to any other, albeit by a very small amount). This forced the introduction of a memory constrained approach: after each join, the relation is sorted according to the score and cropped to its first best N elements. This beam-search approach to relational algebra, similar to early termination in IR posting list merging, brings linear execution time (proportional to the number of TPs), but also risks losing correct results. So N is an hyperparameter of query processing that needs to be properly tuned on the specific case.

6 FUNCTIONS LIST

Table 1 and Table 2 summarize the *predicates* of SPARQL/T triples that apply to words, lemmas, sentences and their respective embeddings. They have all the same syntax:

```
?var PREFIX:FUNCTION <list of words>.
```

The `PREFIX` of each predicate specifies the unit to consider (word, lemmas, …), whilst the `FUNCTION` code specifies how to use the list of words that follows (words are separated by spaces, n-grams can be specified by joining words with the hyphen char). For example, the first of the following TPs extracts all the words whose lemma is either '*love*' or '*like*' (with score=1), whilst the second extracts (almost) all the words in the document and score them accordingly to the distance from their Word Embedding and the average of the Word Embeddings of '*buy*' and '*rent*'.

```
?x LEM:ANY 'love like' .
?y EMB:AVG 'buy rent' .
```

This first set of predicate function is quite homogeneous, meaning most of the possible combinations of `PREFIX` and `FUNCION` codes are valid, allowing users to quickly experiment different variations. The remaining SPARQL/T functions are listed in Table 3. The Part Of Speech (POS), the Named Entity Recognition (NER) and the Entity Linking (EL) functions extracts the respective annotations, restricted to the kind specified in the literal (object) part. NER accepts the name of a class of entities, which is generally quite broad (persons, organizations, …), and returns snippets of text. EL also accepts the name of a class of entities, but it is normally narrower and context dependent (telephone-offer, mobile-phones, …). Moreover, it associates the entity URI with the snippet of text.

The Dependency Parsing (DEP) and the Semantic Role Labeling (SRL) functions employ two variables and operates on the respective annotation trees. For example, if `x` is already bound to a verb, the following TP binds `y` to its object (according to the dependency tree of the sentence):

```
?x DEP:DOBJ ?y
```

Finally, the Information Retrieval (IR) function allows to restrict the rest of the SPARQL/T query to the results of an initial classical IR query (redirected to Apache Lucene). Its purpose is to speed up the query execution by limiting the data, whenever possible, and it must appear as the first TP of the query itself.

PREFIX	Units of text considered
WRD	Words
LEM	Lemmas
EMB	Word Embedding
EML	Word Embedding of Lemmas
SEN	Sentence

Table 1: Possible prefixes of the homogeneous set of TPs, that apply to sequences of tokens. The prefix indicates the units to consider.

FUNCTION	
ANY	Match any of the listed elements
SEQ	Match the entire sequence
PER	Allows Permutation in the sequence
SUM	Sum of the Word Embedding vectors
AVG	Average of the Word Embedding vectors
REX	Regular Expression

Table 2: Possible functions of the homogeneous set of TPs. They indicate how to use the sequence of tokens specified in the object position of the TPs.

Syntax	NLP Task	Func. & Param.
`?x NLP:POS "lit"`	POS Tagging	`lit∈{verb,noun, …}`
`?x NLP:NER "lit"`	NER	`lit∈{person, location, organization …}`
`?x NLP:EL "lit"`	Entity Linking	`lit=URI|entity_set`
`?x DEP:FUN ?y`	Dependency Parsing	`FUN∈{nsubj, dobj, neg, …}`
`?x SRL:FUN ?y`	SRL	`FUN∈{ARG0,ARG1, …}`
`?x IR:QRY "query"`	Information Retrieval	`query = Lucene query string`

Table 3: Other (non-homogeneous) SPARQL/T Triple Patterns

7 ARCHITECTURE

The current version of the SPARQL/T engine does not focus on speed. The dataset we expect to deal with may be big, but not huge (certainly not the size of the web), and from the perspective of a company that is mining its client problems, results are not necessarily expected in real time (few hours of computation are easily acceptable). The Corpus of documents in SPARQL/T is thus divided into units of relatively small size (called segments) that the query engine considers as a whole. Queries can run on a single or a group of segments. Moreover, a mechanism is provided to extract a small Working Segment from the corpus, using a Lucene query, to be used to test and trim the queries before running the full job. Each segment is made of two parts: one containing the text and its annotations (ANN), serialized in JSON, and a binary one containing the Sentence Embeddings (EMB). SPARQL/T also employs the Apache Jena framework, for the RDF/OWL part, and the Apache Lucene search engine, that is mainly involved in the creation of the Working Segment but that can also be evoked inside a SPARQL/T query.

SPARQL/T is written entirely in Java, using Antlr 4 library[4] [11] to parse query strings.

8 PERFORMANCE EVALUATION

Concerning the experimental setting, we used a dataset consisting of 20,293 complaint messages downloaded from a single Italian forum on Telephony, mostly regarding the TIM telephone company, and related to the year 2018. We employed Word Embedding vectors with 300 dimensions, downloaded from the fastText[5] web site. The tests were performed on a PC mounting an AMD Athlon X4 880k at 4GHz, with 16 GB of RAM and SSD disk (GPU not used). Finally, we report execution time measured in millisecond, and averaged over 10 trials.

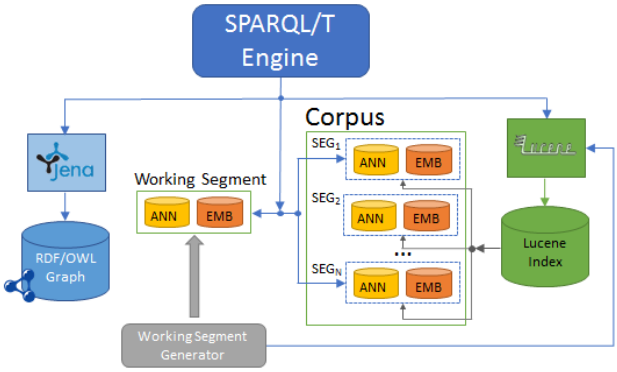

Figure 4: SPARQL/T Architecture

In SPARQL/T most NLP annotations are pre-computed, and Word Embedding comparisons only require a dictionary search and a vector product. The bottleneck, if present, should be the algebra operations, especially the CROSS PRODUCT for large relations. To evaluate this bottleneck, we consider lists of TPs, build a SPARQL/T query incrementally adding one TP at a time, and measure its processing time. Specifically, Table 4 contains the five Word Embedding TPs of our test query, each referring to a distinct variable x_i. The *information need* of the complete query is very simple: find complaints that approximately mean "*yesterday (someone) requested to activate a Vodafone SIM*" (i.e.: all the listed words, or words similar to them, must be present inside the same sentence and be close to each other).

i	Triple Pattern (TP)	Cum. Time (ms)
1	`?x1 EMB:ANY "attivare".`	777
2	`?x2 EMB:ANY "richiesta" .`	1885
3	`?x3 EMB:ANY "sim" .`	2903
4	`?x4 EMB:ANY "vodafone"`	3694
5	`?x5 EMB:ANY "ieri"`	4516

Table 4: A sequence of TPs sharing no common variables. The relations extracted by each TP are combined with a sequence of CROSS-PRODUCT operations. The third column report the execution time of the query (in ms) when the TPs are added one at a time. As can also be seen in Figure 7, time grows linearly with the number of TP.

This particularly simple query is translated into a sequence of CROSS PRODUCT operations. The pseudo-code is depicted in Figure 5. The third column on Table 4 reports the cumulative

[4] https://www.antlr.org/

[5] https://fasttext.cc/

execution time of the TPs, i.e. the time spent to execute the first *i* TPs, for *i = 1…5*. The graph in Figure 7 clearly shows that the trend is linear, thanks to the application of the top-k function that avoids exponential behavior.

```
R=Extract(TP[1])
for i=2 to N
        R=CrossProduct(TopK(R), TopK(Extract(TP[i])))
```

Figure 5: Pseudo-code for the execution of the query in Table 4: for each textual complaint, each TP_i extracts its relation, that is truncated to its top-k results and combined (CROSS_PRODUCT) with the top-k results of the relation at step i-1, to form the relation at step i.

The second set of TPs used in another test query is illustrated in Table 5. Note that in this case the TPs are connected to each other by common variables, and give rise to a sequence of JOIN operations (the pseudo-code is reported in Figure 6. The *information need* of the full query is now: *"find complaints stating that someone activated something Y, where Y is the object of a verb like 'activate' (and thus it is not forced to be similar to anything in particular)".*

i	Triple Pattern (TP)	Cum. Time (ms)
1	`?x LEM:ANY "attivare" .`	109
2	`?x NLP:POS "verb" .`	129
3	`?x DEP:NSUBJ ?y .`	117
4	`?y NLP:POS "noun" .`	118
5	`?y DEP:DET ?z .`	138

Table 5: A sequence of TPs connected by variables. The relations extracted by each TP are combined with a sequence of JOIN operations. The third column report the execution time of the query (in ms) when the TPs are added one at a time.

```
Joiny(  Joiny(  Joinx(  Joinx(Extract(tp1), Extract(tp2)),
                        Extract(tp3)
                ),
                Extract(tp4)
        ),
        Extract(tp5))
)
```

Figure 6: Pseudo-code for the execution of the query in Table 5: the relation extracted by each TP are combined by JOIN operations. In this case the results of all TPs are crispy (there is no similarity measure involved), so they are not truncated by a top-k functions.

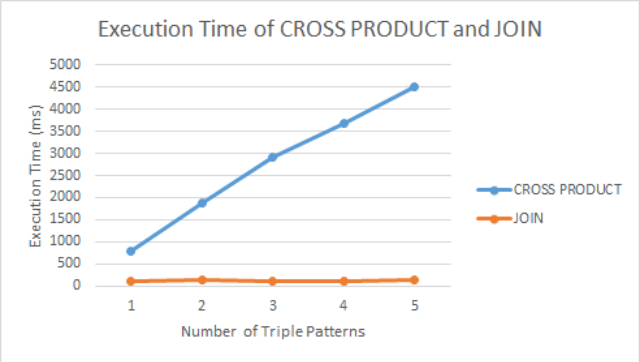

Figure 7: Cumulative execution time of a sequence of TPs in the case of CROSS PRODUCT (Table 4) and JOIN (Table 2)

9 CONCLUSIONS AND FUTURE WORK

We discussed the main features of SPARQL/T and the current prototype engine. We plan to include in the language many other NLP tasks. Among the most urgent ones, there is the detection of the negations in texts, as most complaints are about things that do not work or have not been done. Sentiment Analysis would also help in identifying the issues raised by customers, which are probably stated in proximity of negative sentiment expressions.

REFERENCES

[1] B. Adrian, J. Hees, L. van Elst and A. Dengel, iDocument: using ontologies for extracting and annotating information from unstructured text. In: Proceedings of the 32nd Annual German Conference on AI, (Springer-Verlag, Heidelberg, 2009).

[2] M. Banko, M.J. Cafarella, S. Soderland, M. Broadhead, O. Etzioni, Open information extraction from the Web, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007, pp. 2670–2676.

[3] H. Bast, F. Baurle, B. Buchhold, and E. Haußmann. 2014. Easy access to the Freebase dataset. In WWW. 95–98

[4] Bast, H., Buchhold, B.: Qlever: A query engine for efficient sparql+text search. In:CIKM. pp. 647–656. ACM (2017)

[5] Cafarella, M. J., Banko, M., & Etzioni, O. (2006). Relational web search. Tech. rep., University of Washington, Department of Computer Science and Engineering. Technical Report 2006-04-02.

[6] Corcoglioniti, F., Rospocher, M., Palmero Aprosio, A.: A 2-phase frame-based knowledge extraction framework. In: Proc. of ACM Symposium on Applied Computing (SAC'16)

[7] Cyganiak, R. 2005. A relational algebra for sparql. Tech. rep. HPL-2005-170, HP-Labs. http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html

[8] A. Gangemi, V. Presutti, D. R. Recupero, A. G. Nuzzolese, F. Draicchio, and M. Mongiovì. Semantic web machine reading with FRED. Semantic Web, 8(6) 2017.

[9] H. Kamp. A theory of truth and semantic representation. In J. A. G. Groenendijk, T. M. V. Janssen, and M. B. J. Stokhof, editors, Formal Methods in the Study of Language, Part I, pages 277–322. Mathematisch Centrum, 1981.

[10] Niklaus C, Cetto M, Freitas A, Handschuh S, A Survey on Open Information Extraction, arXiv:1806.05599

[11] T. Parr, The Definitive ANTLR 4 Reference, 2nd ed. Pragmatic Bookshelf,2013.

[12] U. Straccia, Foundations of Fuzzy Logic and Semantic Web Languages, CRC Studies in Informatics Series, Chapman & Hall, 2013.

[13] Tablan, V., K. Bontcheva, I. Roberts, and H. Cunningham (2015). Mímir: An open-source semantic search framework for interactive information seeking and discovery. In: J. Web Sem. 30, pp. 52–68.