Università
Ca'Foscari
Venezia

Corso di Dottorato di ricerca

in Informatica

ciclo 31°

Tesi di Ricerca

Information Retrieval and Extraction from Forums, Complaints and Technical Reviews

SSD: INF/01

Coordinatore del Dottorato
ch. prof. Riccardo Focardi

Supervisore
ch. prof. Salvatore Orlando

Dottorando
Bruno Quintavalle
Matricola 761617

Many thanks to prof. Salvatore Orlando
for the patience and for the precious indications provided,
and to prof. Fidel Cacheda and prof. Renato Fileto
for the corrections and very useful suggested ideas.

Preface

This thesis is about a query language that, in my view, should allow an average computer technician to easily deal with Natural Language Processing tasks, without the understandable fear for its technicalities. In the company I work for I see everyday lots of people naturally playing with complicated SQL queries involving heterogeneous systems, about which I would be scared myself. On the other hand, it is very easy to frighten them with any simple expression containing mathematical symbols or otherwise strange characters. My guess is that a query language like SPARQL should do the job, provided I can manage to fit all the most useful NLP task into simple triple patterns.
Mining complaints is a necessity of many companies nowadays, and to make it simple is one of the main focuses of this research.

Abstract

Complaints and technical reviews often describe complex problems, most of the times in very articulated ways. Over that kind of corpora, we are considering here three classical tasks: Information Retrieval, Text Classification and Information Extraction. In this context however, these tasks should take into special consideration the structure of the sentence, with special attention to verbal phrases, as complaints are usually descriptions of actions that have been performed whilst they shouldn't (or the other way around). We want to leverage results from traditional NLP tasks like Semantic Role Labeling and Dependency Parsing, but also to employ the most recent advances in the field of Word and Sentence Embedding. Moreover, Semantic Web technologies should be employed when background knowledge is required. In order to deal with these three heterogenous approaches, a particular implementation of the SPARQL query language has been developed. It provides a language for template extraction that seamlessly mixes the state of the art of the above-mentioned tasks. Its main difference from SPARQL is the ability to deal with similarity and uncertainty. However, its syntax is strictly a subset of the SPARQL 1.1 one, simplifying the integration with OWL ontologies and allowing its use as an endpoint for other engines in a federated query context. The case studies illustrated here focuses mainly on problems related to telecommunication companies, using publicly available corpora and forums threads extracted from the web. However, the language has been designed to be used in any context that requires extracting information from corpora of complex or technical descriptions.

Table of contents

Table of contents

Chapter 1 – Introduction

When dealing with a dataset of complaints and technical reviews, there are basically three things that we want to be able to do:

- retrieve some of them, according to some information need
- count them, to measure the importance of a specific area of problem
- extract detailed information from them, for example to gain some insights about how to correct the problem

Retrieval of complaints and technical reviews need a special approach. Complaints are usually convoluted descriptions of very complex problems. Moreover, they often implicitly refer to some "obvious" background knowledge that we need to represent. Clearly, in order to correctly retrieve them, the structure of the sentence and of the discourse cannot be disregarded, as instead is normally the case in classical Information Retrieval (IR) approaches. Classical IR systems, having to focus on speed and to deal with huge amounts of documents, makes some necessary simplifications, almost always considering an entire document as an unstructured Bag Of Word (BOW). Here we cannot do the same, and certainly, the burden of employing Natural Language Processing (NLP) techniques will make the things much slower and less scalable. But from the perspective of a company that wants to explore a specific area of interest, speed and scalability may not be a major concern: datasets may be big, but not huge (certainly not the size of the web), and results are not necessarily expected in real time (few hours of computation are easily acceptable).
The second thing that we want to be able to do with a set of complaints is to count the instances of a specific class of problem, within a reasonable margin of error, to measure its entity. This is clearly a classification problem. However, the difficult here is the creation of the training set. It is unlikely that a company is willing to spend the necessary time and effort to collect thousands of examples for each case of interest before having the first results. Also, for most specific problems, these thousands of examples may not exist at all, not even in the web. Therefore, we need to find classification methods that focus on minimizing the number of instances and the user involvement, rather than just considering evaluation metrics like precision and recall.
The third requirement is the ability to perform Information Extraction (IE) on the retrieved documents. The user may want to know things like "*what gets broken*" more often, or "*how much has been payed*" on the average for something. The problem here is that there are obviously many different ways to express all those things in natural language. Fortunately, this issue has been studied from a very long time, and tools are already available off-the-shelf that tackle the problem from different theoretical perspectives (Syntactic Parsing, Semantic Role Labeling, Sentence Similarity). However, none of these tools alone seem to provide a solution that works in all cases. At the moment, the best option from the user perspective is probably an environment that allows the choice of any of them, and possibly also their seamless interaction.

With these three goals in mind, instead of creating a pipeline specifically tailored for the purpose of complaint analysis, we built a flexible tool that may be used by others to tackle similar problems. Interestingly, an area with very similar problems is the field of clinical research, and in particular the analysis of clinical narrative, thanks to the rapid growth in the implementation of electronic health records (EHRs). We report here the words of a very recent article [Zhang et al 2018]: "*psychiatric symptoms often*

consist of subjective and individualized descriptions, which are presented in details of the patient's experience. Instead of a single word or simple noun phrase, psychiatric symptoms have tremendous syntactic and semantic variability. […] Therefore, it is quite challenging for traditional natural language processing (NLP) techniques to automatically extract such diverse mentions of psychiatric symptoms from text."

In the hope to be useful also to this and other applications, we wrote a tool that aim at unifying the state of the art of traditional NLP tools, including the newest Neural Embeddings ones. Ideally, this tool should be intuitive and easy to use, allowing persons without a strong background in NLP to achieve the above-mentioned goals with little coding and very little effort. It is a query language derived from SPARQL (named SPARQL/T, like "SPARQL over Text"), whose syntax is a strict subset of SPARQL 1.1 one, but that works directly on the document's (pre-annotated) text instead of on a set of RDF triples. It allows to seamlessly mix inside the same query clauses that refer to traditional NLP tasks with others that involves uncertainty and similarity measures. For this reason, we had to write its engine from scratch, as existing SPARQL implementations cannot deal very well with uncertainty. However, keeping the syntax exactly the same, besides avoiding the introduction of a dialect, also simplifies its integration with Semantic Web technologies. (for example, SPARQL/T can act as a standard SPARQL endpoint, i.e. it can participate to a federated query initiated by another SPARQL engine).

Chapter 2 – Comparison to Knowledge Extraction tools

The triple patterns of a query in SPARQL/T acts directly on the text and its annotations, extracting knowledge during query execution. An obvious alternative is to extract all possible useful triples from the documents into a triple store, and then employ a standard SPARQL engine. This is exactly what Knowledge Extraction (KE) tools allows to do. This chapter explores the pro and cons of both solutions. In short: at run time pre-extracted knowledge has the advantage of speed. However, the KE task is a very difficult one. Similar documents do not always result in structurally similar graphs, making it difficult to write the queries. Thus, from the user point of view, SPARQL/T similarity-based approach is definitely much easier.

Knowledge Extractors 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 (see chapter 3.1 for more details). It is therefore natural to ask what advantages can be achieved with the SPARQL/T approach, i.e. by working directly on the documents and their NLP annotations, and thus giving up (at least for the moment) the advantages of indexing.
What lacks in SPARQL (but not in general in the KE tools themselves) is the ability to deal with uncertainty. SPARQL queries are crispy, the graphs extracted by KE tools have many different structures, even when extracted from sentences apparently very similar, and thus the user is forced to write very complex queries to achieve even just reasonable recall. SPARQL/T on the other hand relies a lot on the concept of similarity. Words and sentences can be compared using Embeddings, the Join operations consider (among other things) proximity measures and, whenever possible, soft operations that perform a re-ranking of the result are preferred against others with crispy behavior. Obviously, this approach makes SPARQL/T less precise than using SPARQL over KE results, but much more flexible and easier to use.
Figure 1 shows a SPARQL/T triple pattern that employs Universal Sentence Encoder (USE) [Cer at al. 2018]. The `EMB:USE` triple pattern simply extracts from the documents all the snippets of text similar to a given phrase ("*`They increased the price`*").
As can be seen from Table 1 (containing some results of its application to the Comcast Consumer Complaints dataset[1] available on Kaggle.com), sometimes Google USE gives surprisingly good results in finding sentences that are structurally very different from the query ones (albeit with some noise), and its implementation in SPARQL/T requires no effort from the user point of view.

```
?i    EMB:USE    'They increased the price'
```

Figure 1: A triple pattern that extracts snippets similar to the given phrase, using Universal Sentence Embedding

[1] Comcast Consumer Complaints. Public complaints made about Comcast internet and television service. https://www.kaggle.com/archaeocharlie/comcastcomplaints

For comparison, we have extracted the graphs of those sentences using the online version[2] of Pikes [Corcoglioniti et al. 2016], which is a frame-based Knowledge Extraction framework, that produce instances of frames and relations between them in RDF format.
Actually, we have started with three handcrafted very similar sentences, slightly varying the sentence 1 of Table 1, to check if they would be transformed into similar graphs (see Table 2).

Rank Index	Score	Snippet
1	0.895	they raised the prices
7	0.851	they changed the deal
8	0.85	the pricing went up
9	0.846	they implemented the extra charges
21	0.834	They lied about the pricing
22	0.833	They doubled the rates
23	0.833	not only did the prices become exorbitant
24	0.832	They increased the equipment fee
36	0.823	They have consistently offered temporary rates
40	0.822	they have jacked up prices
44	0.82	They simply tick up the cost

Table 1: Some results of the query in Figure 1 from the COMCAST database

1	Telecom Italia has raised the price of the lines.
2	Vodafone has increased the cost of the equipment.
3	British Telecom has doubled the rent expenses.

Table 2: three handcrafted very similar sentences

In fact, as expected, graphs 2 and 3 have very similar structures. Graph 1 instead, maybe because of an error, is quite different from both.
Disregarding thus for the moment graph 1, a possible graph path that encompasses the structures of the other two is depicted in Figure 3: a chain of four nodes (indicated by four variables), linked by some relations (any possible relation is accepted), but with specific constraints on the classes that these nodes belong to. Of course, this is just one of many possibilities. We have chosen here, for the sake of a higher recall, of not caring about the kind of relations, hoping to identify, by watching graphs 2 and 3, four classes of objects to be used as constraints.
Table 3 shows all the classes to which the verbs of the three sentences belong ("*raise*","*increase*" and "*double*"). Here, the choice of the constraint for the `verb` variable seems quite

[2] http://pikes.fbk.eu/

straightforward: all three nodes are instances of the class `sumo:Increasing` (defined in the SUMO ontology, see [Niles & Pease 2001], [Pease 2011]). For the `cost` variable however (Table 4: "*price*", "*cost*" and "*expenses*") the situation is not so lucky: in graphs 2 and 3 the object is an instance of the class `pm:fn15-expensiveness` (frame "Expensiveness" of FrameNet 1.5), but this is not true for graph 1. For the `obj` variable (Table 5: "*lines*", "*equipment*" and "*rent*"), there is no class that includes any two of them and that looks reasonably small to be used as a constraint. Of course, exploring the involved ontologies it may be possible to find a suitable superclass, or we can just manually create one for the purpose. However, this would be time consuming and quite error prone.

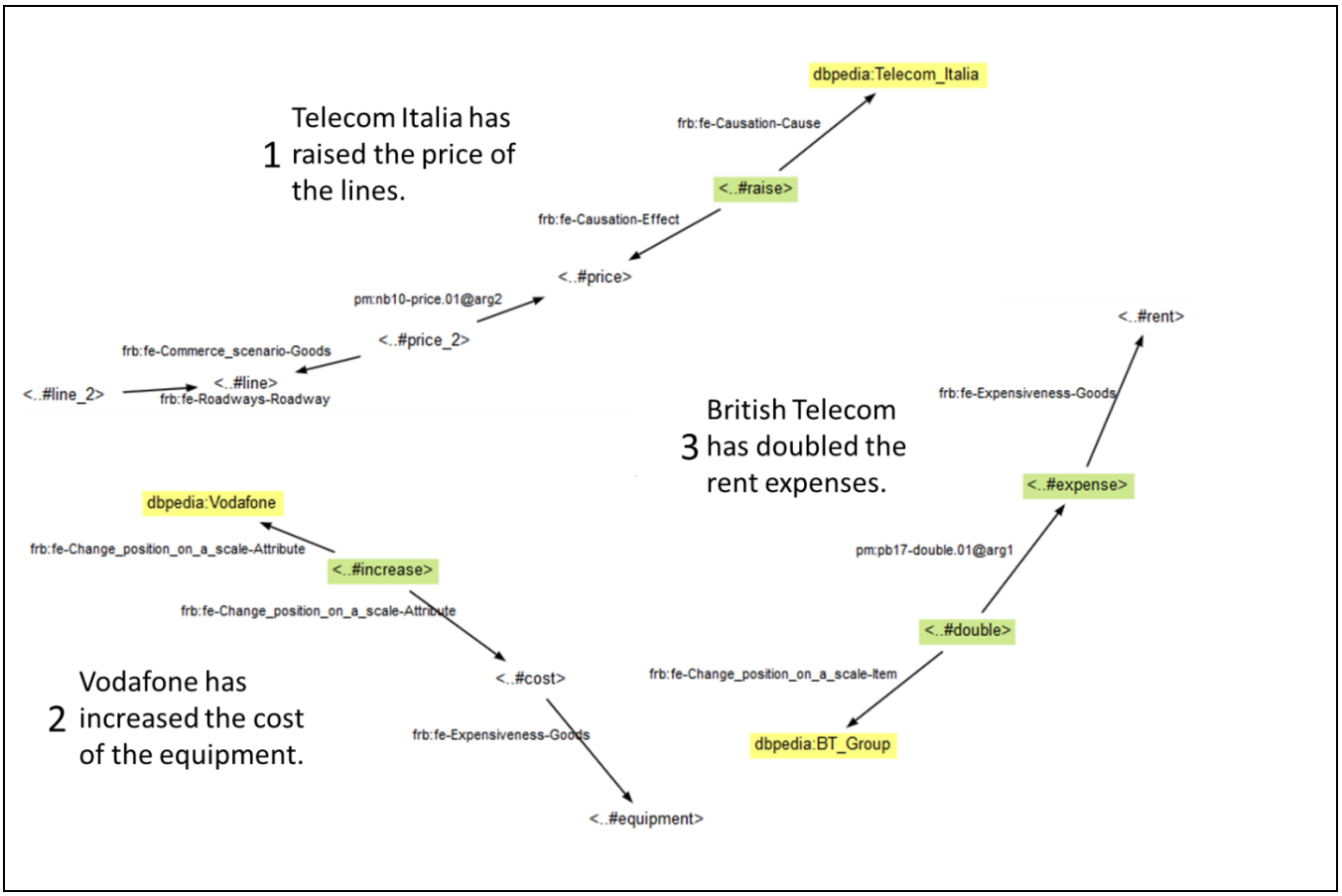


Figure 2: graphs extracted by PIKES from the three handcrafted sentences of Table 2

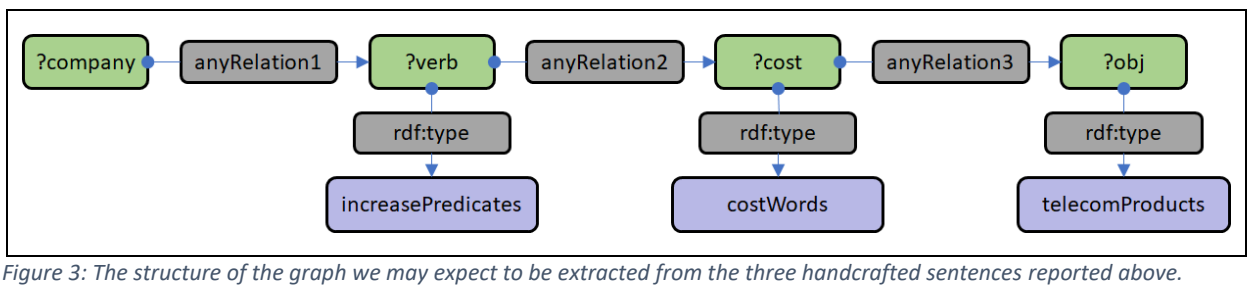


Figure 3: The structure of the graph we may expect to be extracted from the three handcrafted sentences reported above.

Turning now to the other sentences retrieved by the USE pattern (Table 1), they are semantically similar texts that have different syntactic structures. This means that, besides the same difficulties in identifying the proper constraints for the nodes, we are faced with a much larger variety of graph structures to take into account, which makes quite hard to write a SPARQL query that can encompass them all. Figure 4 shows some examples.

In summary, Knowledge Extraction is a difficult problem, and not surprisingly, the resulting graphs are, so to speak, not very easy to recall with a query. To be able to query them easily, pieces of text with similar meaning should result in reasonably similar graphs, and this is often not the case. SPARQL/T on the other hand, leaves to the user the burden of figuring out how the knowledge is expressed in the text. However, the user does not need to be very precise in doing that: the language provides the flexibility necessary to cope with errors and uncertainty.

In general, as just shown, from the user point of view (albeit certainly not from the computational one), SPARQL/T queries on text require much less effort than SPARQL ones on RDF graphs extracted from the same text.

For the sake of fairness, it should also be said that, besides the fact that this test is far to be exhaustive, it can be argued that it is not the data produced by the Knowledge Extraction tool that is difficult to use with SPARQL, but that is the SPARQL language that is not flexible enough for the purpose. A new Conceptual Model of SPARQL/T that introduces a degree of truth also in graph pattern matching is probably worth to be studied.

raised (graph 1)	**increased** (graph 2)	**doubled** (graph 3)
<..#denotedBy> <..#char=19,25> rdf:type frb:frame-Causation-raise.v rdf:type dbyago:Chandelle100169651 rdf:type eso:Decreasing rdf:type eso:Increasing rdf:type eso:Motion rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-causation rdf:type pm:pb17-raise.01 rdf:type pm:vb32-raise-9.4 rdf:type sem:Event rdf:type sumo:Entity rdf:type sumo:Increasing rdf:type sumo:InternalChange rdf:type sumo:Physical rdf:type sumo:Process rdf:type sumo:QuantityChange rdfs:label "raised"	<..#denotedBy> <..#char=13,22> rdf:type frb:frame-Change_position_on_a_scale-grow.v rdf:type eso:Decreasing rdf:type eso:Increasing rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-change_position_on_a_scale rdf:type pm:pb17-increase.01 rdf:type pm:vb32-increase-45.6-1 rdf:type sem:Event rdf:type sumo:Entity rdf:type sumo:Increasing rdf:type sumo:InternalChange rdf:type sumo:Physical rdf:type sumo:Process rdf:type sumo:QuantityChange rdfs:label "increased"	<..#denotedBy> <..#char=20,27> rdf:type frb:frame-Change_position_on_a_scale-double.v rdf:type eso:Creating rdf:type eso:Decreasing rdf:type eso:Increasing rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-change_position_on_a_scale rdf:type pm:pb17-double.01 rdf:type pm:vb32-double-45.4 rdf:type sem:Event rdf:type sumo:Entity rdf:type sumo:Increasing rdf:type sumo:InternalChange rdf:type sumo:Physical rdf:type sumo:Process rdf:type sumo:QuantityChange rdfs:label "doubled"

Table 3: triples associated with the `Verb` nodes in the three graphs. In all three cases the node is an instance of the class sumo:Increasing.

price (graph 1)	**cost** (graph 2)	**expenses** (graph 3)
<..#denotedBy> <..#char=30,35> rdf:type dbyago:Price113303315 rdf:type ks:Entity rdfs:label "price" <..#denotedBy> <..#char=30,35> rdf:type frb:frame-Commerce_scenario-price.n rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-commerce_scenario rdf:type pm:nb10-price.01 rdf:type pm:pb17-price.01 rdf:type pm:vb32-price-54.4 rdf:type sem:Event	<..#denotedBy> <..#char=27,31> rdf:type frb:frame-Expensiveness-cost.n rdf:type dbyago:Price105163807 rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-expensiveness rdf:type pm:nb10-cost.01 rdf:type pm:pb17-cost.01 rdf:type pm:vb32-cost-54.2 rdf:type sem:Event rdf:type sumo:Abstract rdf:type sumo:Attribute rdf:type sumo:Entity rdf:type sumo:NormativeAttribute rdf:type sumo:RelationalAttribute rdf:type sumo:SubjectiveAssessmentAttribute rdfs:label "cost" rdfs:seeAlso dbpedia:Cost	<..#denotedBy> <..#char=37,45> rdf:type frb:frame-Expensiveness-expense.n rdf:type dbyago:Expense113275495 rdf:type eso:IntentionalEvent rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-expensiveness rdf:type pm:nb10-expense.01 rdf:type pm:pb17-expend.01 rdf:type sem:Event rdf:type sumo:ChangeOfPossession rdf:type sumo:DualObjectProcess rdf:type sumo:Entity rdf:type sumo:FinancialTransaction rdf:type sumo:IntentionalProcess rdf:type sumo:Payment rdf:type sumo:Physical rdf:type sumo:Process rdf:type sumo:SocialInteraction rdf:type sumo:Transaction rdfs:label "expenses"

Table 4: triples associated with the `Price` nodes in the three graphs. In graphs 2 and 3 the node is an instance of the class pm:fn15-expensiveness, whilst the same node in graph 1 has no (reasonably small) class in common with the other two.

lines (graph 1)	**equipment** (graph 2)	**rent** (graph 3)
<..#denotedBy> <..#char=43,48> rdf:type dbyago:TelephoneLine104402057 rdf:type ks:Entity rdfs:label "lines" rdfs:seeAlso dbpedia:Line_(geometry) <..#denotedBy> <..#char=43,48> rdf:type frb:frame-Roadways-line.n rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-roadways rdf:type sem:Event	<..#denotedBy> <..#char=39,48> rdf:type frb:frame-Gizmo-equipment.n rdf:type dbyago:Equipment103294048 rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-gizmo rdf:type sem:Event rdf:type sumo:Artifact rdf:type sumo:Device rdf:type sumo:Entity rdf:type sumo:Object rdf:type sumo:Physical rdfs:label "equipment"	<..#denotedBy> <..#char=32,36> rdf:type dbyago:Rent113295657 rdf:type ks:Entity rdf:type ks:Predicate rdf:type pm:fn15-cause_to_fragment rdf:type sem:Event rdf:type sumo:Abstract rdf:type sumo:ConstantQuantity rdf:type sumo:CurrencyMeasure rdf:type sumo:Entity rdf:type sumo:PhysicalQuantity rdf:type sumo:Quantity rdfs:label "rent" rdfs:seeAlso dbpedia:Renting

Table 5: triples associated with the `Obj` nodes in the three graphs. There is no reasonably small class in common to be used as a filter for this kind of object.

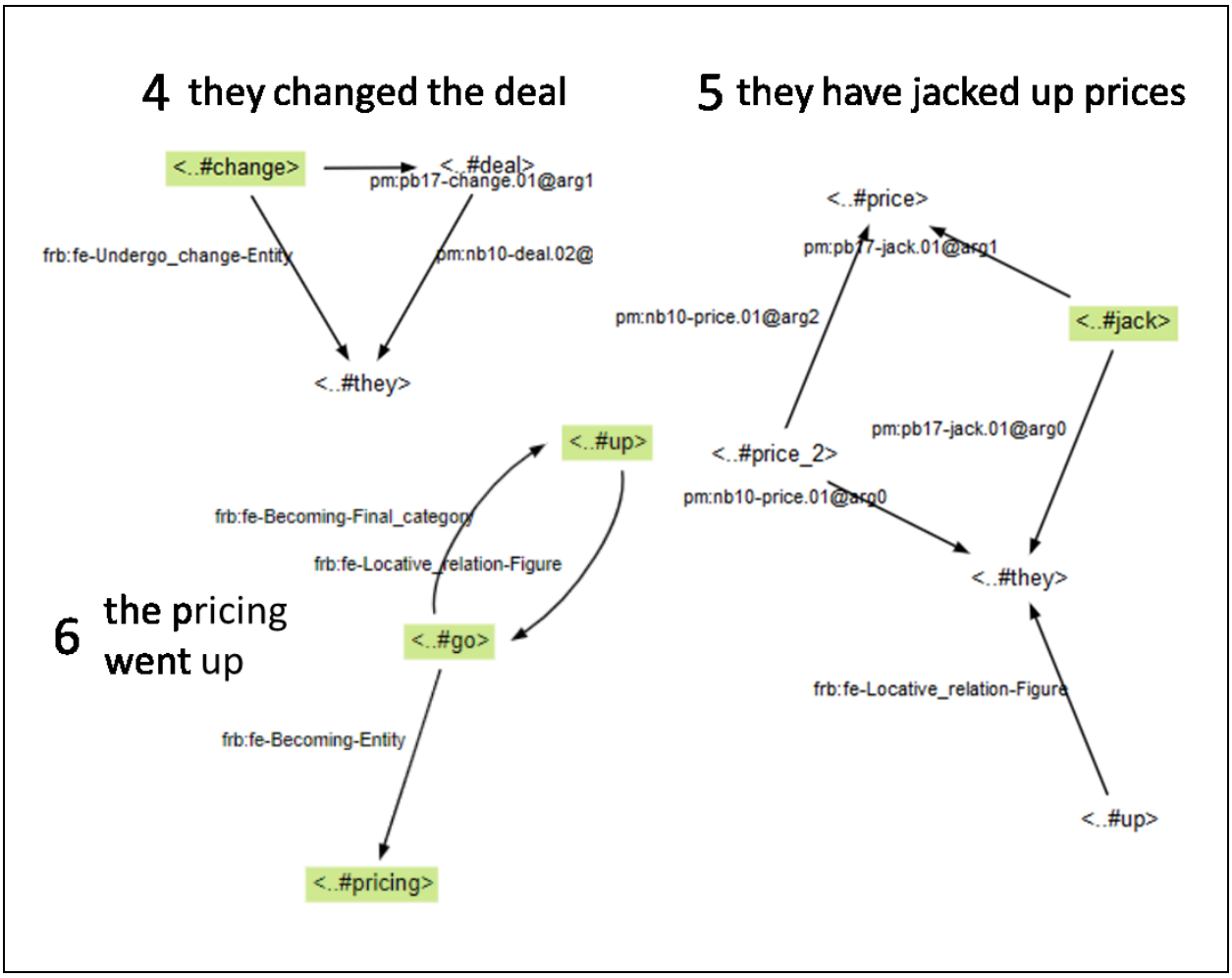


Figure 4: graphs extracted from some other sentences of Table 1

Chapter 3 - Related Work

SPARQL/T is fundamentally a language for both Information Retrieval and Information Extraction from text, whose syntax is a strict subset of the SPARQL 1.1 one. It works directly on the documents and on their NLP (pre-extracted) annotations, can deal with uncertainty and allows the use of Word Embeddings. An alternative approach is provided by Knowledge Extraction tools, described in Chapter 3.1. In Chapter 3.2 we will see some other works that, like SPARQL/T, employ SPARQL syntax for IR/IE tasks. Chapter 3.3 is about the idea of using the Relational Algebra for IE, independently from the syntax employed. Finally, Chapter 3.4 explores the Annotation Ontologies, i.e. some guidelines on how to represent NLP annotations in RDF format.

3.1 Knowledge Extraction tools

Knowledge Extractors (KE) tools transform Natural Language documents into machine-interpretable formats. FRED and Pikes generates RDF/OWL graphs that can be stored into standard triple stores (and thus indexed) and queried with standard SPARQL. This approach can be seen as an alternative to the SPARQL/T one. However, as discussed in Chapter 2, queries that access those graphs are more difficult to write than the SPARQL/T ones. OpenIE generates triple of strings (i.e. not proper graphs), and is going to be implemented into SPARQL/T model (hopefully) soon.

FRED [Gangemi 2017] 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 [Kamp 1981].
PIKES [Corcoglioniti et al 2016] 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 philosophy that aim at avoiding human intervention like hand crafted extraction rules or large hand annotated training sets. Besides being time consuming, these activities become problematic when large heterogeneous corpora are considered. [Banko 2007] proposed a first solution based on a self-supervised approach, using a parser to train an extractor. Over more than ten years of developments, different solutions have been proposed, sometimes with slightly different goals. See [Niklaus 2018] for a recent survey. 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 the Extraction Graph [Cafarella 2006]. 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.

3.2 Tools that employ SPARQL syntax

Andrian [Andrian et al. 2009] introduce **iDocument**, an Ontology Based Information Extraction tool (OBIE) that employs SPARQL syntax in the extraction templates in place of the traditional regular expressions. In iDocument the following query extracts persons and organizations and facts about memberships from text:

```
SELECT * WHERE
{     ?person   rdf:type         foaf:Person.
      ?person   foaf:member      ?org.
      ?org      rdf:type         foaf:Organisation.
}
```

Figure 5: an example of iDocument query

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. Andrian's is perhaps the tool closer to SPARQL/T in terms of using SPARQL syntax for Information Extraction templates. However, the tool does not include Word Embeddings.

QLever SPARQL+Text [Bast & Buchhold 2017] 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 [Bast et al. 2014], Clue-Web[3] 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. Figure 6 gives an example of QLever SPARQL+Text query.

[3] The Lemur Projekt http://lemurproject.org/clueweb12.

```
SELECT ?astronaut ?agency TEXT(?text)
WHERE
{ ?astronaut <is-a>              <Astronaut>.
  ?astronaut <SpaceAgency>       ?agency .
  ?text      ql:contains-entity  ?astronaut .
  ?text      ql:contains-word    "walk*" .
  ?text      ql:contains-word    "moon" .
}
ORDER BY DESC(SCORE(?text))
```

Figure 6: an example of QLever SPARQL+Text query

Mìmir [Tablan et al 2015] 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. Then the following Mìmir semantic query retrieves documents mentioning scientists born in London:

```
{     Person
      sparql="SELECT DISTINCT ?inst WHERE {
            ?inst                                          :birthplace
      <http://dbpedia.org/resource/London>.
            ?inst a :Scientist.
            }"
}
```

Figure 7: an example of Mìmir query

3.3 Annotation Ontologies

Although it isn't strictly a related work, it is useful to point out here that some standards proposals exist that suggest the way in which NLP annotations should be written in RDF stores. OLiA (Ontologies of Linguistic Annotation) is a set of ontologies designed to deal with the heterogeneity of linguistic annotations [Chiarcos 2015]. According to the author, the ontology-based descriptions are comparable across different corpora and/or NLP tools, across different languages, and even across different types of language resources. OLiA defines a Reference Model that specifies a common terminology (like `olia:Determiner` and `olia:Accusative`), and Annotation Models that include individuals (i.e. concrete tags). OLiA Reference Model provides a hierarchy of classes (ex: `DemonstrativeDeterminer` subclassOf `Determiner` subclassOf `PronounOrDeterminer` subclassOf `Morphosyntactic Category`). This makes it easier to combine results of different NLP modules, in case they return tags at different levels of the hierarchy. In fact, as the authors suggest, this may lead to the development of novel ontology based NLP algorithms, for example by simply applying a majority based combination of the different results. OLiA is integrated inside the NLP Interchange Format (NIS), an RDF/OWL-based format that aims to achieve interoperability between different NLP tools, language resources and annotations. In turn, NIS

format has been adopted inside an even broader initiative, the Linguistic Linked Open Data (LLOD) cloud , that focuses on the usage of linked data to represent linguistic resources, like for example DBPedia (in different language versions), WordNet and FrameNet

Chapter 4 - Neural Models

This chapter reviews the state of the art of Word and Sentence Embeddings. Although SPARQL/T employs pre-trained models, the answer the literature provides to some questions motivate some choices taken in SPARQL/T design.

4.1 - Word Embeddings

The peculiarity of SPARQL/T, if compared with other languages for Information Extraction, is the ability to employ Word Embeddings, and to smoothly mix them with traditional NLP techniques and Semantic Web technologies. Word Embeddings provide a measure of semantic similarity between words, and can be used in a query to (fuzzy) specify a set of words to consider (instead of using a crisp list of words or lemmas, or in place of a WordNet synset).
In this chapter we are not focusing on how to extract Word Embedding from corpora, as many off the shelf good ones are already available online. In fact, being the quality of a Word Embedding largely dependent on the size of the corpora employed to make it (Chapter 4.1.6), it is quite unlikely that a "home-made" Word Embedding can outperform the ones provided (for free) for example by Google. However, a possibility remains that a domain specific Word Embedding, extracted from a particular corpus, may be better suited for some specific purposes, or that it can be employed somehow to improve a general purpose one (Chapter 4.1.7).
Interesting issues considered here are also how to evaluate the quality of a Word Embedding (4.1.2), in which way they can be employed in downstream applications (text classification, clustering) and, perhaps even more importantly, how to use them with units of text larger than the single word (Chapter 4.1.4).

4.1.1 Introduction

Word Embeddings are vector representations of words. More specifically, Word Embeddings associate to each word in a vocabulary a vector v such that the similarity between two words can be measured as a function of their respective vectors.
Word Embedding vectors are relatively small, typically in the order of 50 to 300 dimensions, as opposite to the one-hot representations that employ vectors of the size of the dictionary (typically hundreds of thousands of elements), where all elements are zeros except for the one in the position associated to the word. In NLP, Word Embeddings have various applications. They can simply be used directly to assess the similarity of words or larger units of text (by comparing the average vectors of the two units). They can be employed in classical machine learning algorithms, for example for clustering or classification of documents, and they are largely employed as input layers in Deep Neural Networks.
Word Embeddings are extracted from very large corpora by algorithms that, in a way or another, leverage Harris distributional hypothesis ([Harris 1954], [Sahlgren 2008]), i.e. that similar words

occur in similar contexts. The context of a word can be something as simple as a fixed window of words around it, or it can be a syntactic structure like a phrase or a sentence.
According to Mikolov [Mikolov et al. 2013], the idea of encoding words in dense vector representations may be dated back at least to 1986 [Hinton et al. 1986].
Nowadays there are several different algorithms to extract Word Embeddings from a corpus. Aggarwal [Aggarwal 2018] identifies three different categories: Kernel Methods, Distributional Semantic models and Neural Networks models (actually Aggarwal lists four categories, separating Recurrent NN from the other types of NN).
Baroni [Baroni et al 2014] has shown than Neural Embeddings outperforms Distributional Semantic algorithms in many task (Chapter 4.1.2). The most popular and most well studied Neural Embedding algorithm is probably Mikolov's word2vec [Mikolov et al. 2013], but many other Neural Networks model have been proposed for the purpose (see [White 2018] for a recent survey), and it is probably safe to say that any Neural Network that has been successfully trained on a specific natural language task needs to produce some form of compact vector representation of the words in one of its internal layers, which can be extracted to generate a valid Word Embedding.

4.1.2 Embedding Evaluation

As expressed by Schnabel in [Schnabel et al.2015]: "*A good embedding provides vector representations of words such that the relationship between two vectors mirrors the linguistic relationship between the two words*."
Baroni in [Baroni et al 2014], in order to compare the performances of the traditional Distributional Semantic Models (based on word counts) with those of the Neural Word Embeddings, defined six benchmarks:

- Semantic similarity or relatedness. Similarity and relatedness are two different concepts. The word "coffee" for example is likely to be found in the Embedding as similar to "tea", as they probably often occur in the same context, and for the same reason they will also be found both related to the word "cup". However, we cannot say that coffee is an object similar to a cup. The datasets to assess the performance of an Embedding on this task are necessarily manually created by asking human subjects to rate the degree of semantic similarity or relatedness between two words on a numerical scale. The correlation between the average scores that subjects assigned to the pairs and the cosines between the corresponding vectors in the model space can be used as a measure of performance.
- Synonym detection: The suggested datasets contains a set of records with a target term and an associated set of synonym candidates. The test obviously consists in comparing the vectors of the term with those of the candidates, picking the closest one. Performance is evaluated in terms of accuracy.
- Concept categorization: nominal concepts are clustered together. Here we expect them to group into natural categories (e.g., helicopters and motorcycles should go to the vehicle class, dogs and elephants into the mammal class). A set of gold categories are then employed to measure the purity of each cluster (the extent to which each cluster contains concepts from a single category)
- Selectional preferences: in NLP selectional preference (also known as selectional restrictions) indicates a word's tendency to co-occur with words that belong to certain lexical sets. For

example, the verb marry prefers subjects and objects that denote humans. Selectional preference is used for example in the Word Sense Disambiguation (WSD) task [Resnik 1997]. The datasets suggested for this benchmark contain verb-noun pairs that were rated by human subjects for the typicality of the noun as a subject (or object) of the verb (e.g., people received a high average score as subject of to eat, and a low score as object of the same verb). For each verb we also need a gold set of possible subjects and objects, possibly extracted from a corpus. The vectors of those subjects are averaged to obtain a "prototype" subject of the verb. The cosine similarity between the prototype and the subject in the pair is then compared with the human rating.

- Analogy: a proportional analogy holds between two word pairs: a:a* :: b:b* (a is to a* as b is to b*) For example, Tokyo is to Japan as Paris is to France. Mikolov [Mikolov et al. 2013] found that such analogy could be solved with simple algebraic operations: let X=vector(a)-vector(a*)+vector(b). b* is found by searching in the Embedding space the word closest to X (according to cosine similarity). Given a dataset of 4-tuple of words (a, a*, b, b*), the task is to find one given the other three, using the above algebraic expression. Systems are evaluated in terms of proportion of correct answers.
  A list of datasets available for the purpose is given for example in the Wiki of the Association for Computational Linguistics[4]

In that article Baroni compared word2vec results with those created with classic DSM algorithms, extracted from the same corpus of about 2.8 billion, concluding that Neural Word Embeddings beat the current state of the art in most cases, and approached it in many more. (One exception was the selectional preference tasks, where the two methods achieved comparable results.)
Schnabel [Schnabel et al.2015] further formalized the task of Word Embedding evaluation by identifying two major categories: intrinsic and extrinsic ones. In extrinsic evaluation downstream tasks like POS tagging or Named Entity Recognition are used to compare the performances of different Word Embeddings. Intrinsic evaluations on the other hand directly test for syntactic or semantic relationships between words, using a predefined set of query terms called query inventory (which is what Baroni did in the above-mentioned work). Schnabel also introduced two new tasks, both involving the judgment of a human over the results of Word Embeddings. In what he calls Comparative Intrinsic evaluation, datasets consist of query words only. The words most similar to the query ones are extracted according to different Word Embeddings, and users (from Amazon Mechanical Turk in Schnabel's work) are asked to choose the best matches, according to their perceptions. The Coherence task on the other hand try to assess the coherence of groups of words in a small neighborhood in the embedding space. Intuitively, good embeddings should have coherent neighborhoods for each word (again, according to human perception), so he presented Turkers with four words, three of which are close neighbors and one of which was the "intruder" the human had to spot.
Schnabel observation was that extrinsic evaluation are not consistent across tasks, i.e. different tasks tends to favor different embeddings. Moreover, they are often not consistent with intrinsic evaluations. For these reasons, albeit recognizing the ability of extrinsic tasks to give insight into the information encoded in the embeddings, he discouraged their use as a measure for abstract quality.

[4] https://aclweb.org/aclwiki/Analogy_(State_of_the_art), fetched September 27, 2018

4.1.3 Meta Embeddings / Domain Adaption

Meta Embedding can be defined as the task of creating new embeddings by combining existing ones [Bollegala et al. 2017]. As noticed for example by Schnabel [Schnabel et al.2015] (see Chapter 4.1.2) performances of different Word Embeddings vary significantly across different tasks, suggesting that different methods may capture complementary aspects of lexical semantics. This of course may depend both on the corpus and on the algorithm used to produce them.

Meta embedding can also be seen as a Domain Adaption problem: assuming a corpus of a certain domain of interest is available, there could be reasons to believe it may not be enough for the purpose (typically, for Named Entity Recognition), for example because of its limited size or for the possible different sub-language employed. This is frequently the case in healthcare NLP tasks, where for privacy reasons extensive corpora may not be easily available, or that may use sub-languages inappropriate for the problem (medical doctors tend to write telegraphic reports full of medical terms, whilst people in forums express themselves in totally different ways). As an example, reported in [Zhang et al 2018]: having to deal with medical reports, which corpus should be employed? A relatively small one solely made of such reports or a much bigger general one extracted from Wikipedia? Or can we adapt in some way the bigger one to the domain of the smaller?

The most obvious solution to merge Word Embedding coming from different corpora is probably to merge the corpora and then retrain the whole. However, this may be impractical, mainly for two reasons: some of the source corpora may not be publicly available and the computational effort may be too high. And besides that, we may still want to try to get “the best” of embeddings coming from different algorithms.

Among the challenges in meta embedding construction there is obviously the facts that both the vocabulary and the dimensionality of the source embeddings may be different. Dimensionality differences in particular often rules out the possibility of a trivial sum or average. Vector concatenation on the other hand can be a simple baseline, albeit with the drawback of drastically increasing the dimensionality of the meta embedding. Bollegala reduced such dimensionality by employing Singular Value Decomposition (SVD) to reduce the size of the matrix obtained with the concatenation of the source embeddings ones (using vectors of zeros for words that did not belong to one of the vocabularies).

Yin and Schütze [Yin and Schütze 2016] introduced three neural networks algorithms for Meta Embeddings: 1toN, 1toN$^+$ and MutualLearning, the last two of which are designed to works also with vocabularies that do not overlap, i.e. in case of Out Of Vocabulary (OOV) words. Remarkably, the last one may be used to generate vectors for OOV words of the source embeddings. The network for the 1toN$^+$ is shown in Figure 8, which is trained, on all the words of the intersection of the source vocabulary, to generate the (known) vectors from the meta embedded ones. M_1 to M_5 are the projection matrices from the meta embedding space to each of the source ones. Although, at least in principle, after training them they could also be used directly to recover source embedding OOV vectors from the meta embedding ones, this is not what is described in the article. Their MutualLearning algorithm calculates instead projections between each of the source embeddings, and averages the results for OOV words.

Finally, although the dimensionality of the source embeddings is not guaranteed to be the same, some dimensions are very common in the pre-trained ones (usually 50,100,200 and 300). In these cases, simple averaging can give quite good results, as Coates & Bollegala show in what they call "Frustratingly easy meta-embedding" [Coates & Bollegala 2018].

To summarize the results of the above-mentioned articles, almost all approaches to Meta Embedding are reported to achieve a certain improvement over the use of a single Word Embedding. However, it has been noticed that adding more and more Word Embedding does not always help (see also Chapter 4.1.6 on corpus size). In Yin & Schütze words: "*Whether an embedding set helps, depends on the complementarity among the sets as well as how we measure the ensemble results*".

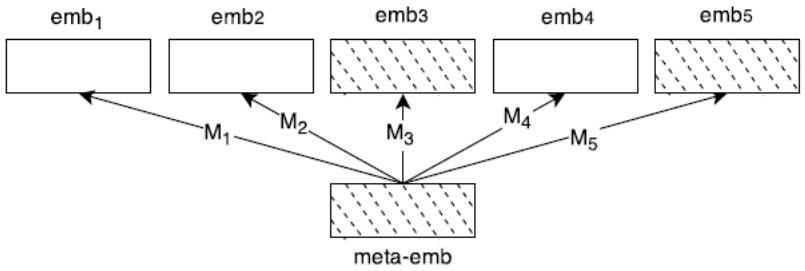


Figure 8: Yin and Schütze 1toN+ Meta Embedding algorithm

4.1.4 N-grams and Collocations

An n-gram is simply a sequence of words extracted from the text. A collocation is an n-gram that correspond to some conventional way of saying things. They can be for example:

- Noun phrases ("*strong tea*", "*weapon of mass destruction*")
- Phrasal verbs ("*to make up*")
- Stock phrases ("*the rich and the powerful*")

Collocation are characterized by limited compositionality, i.e. the meaning of the collocation is not derivable from the meaning of its parts, or at least not completely. "Boston Globe" is a newspaper, and so it is not a natural combination of the meanings of "Boston" and "Globe". More subtly, the meaning of the expression "*international best practice*", although derivable from the meaning of its three components, is usually intended as "*administrative efficiency*", i.e. there is a sort of "extra" meaning added to the sum of the parts. Another characteristic of collocation is that it is not possible to substitute its words with other with similar or same meaning, or to grammatical transform them, for example from singular to plural forms. The classical example is the expression "strong tea": even if "powerful" has a meaning very similar to "strong", the expression "powerful tea" is uncommon and considered strange (although, interestingly, "strong drug" and "powerful drug" are both acceptable).

Collocation are often called in different ways, like "terms", "idiomatic phrases" or "quality phrases", and the task of automatically discover them from a corpus is usually called "automatic term recognition" or "phrase mining". See for example [Liu et al 2017] for a recent survey on the topic. Very briefly, a statistical approach to the problem may consists in identifying in the corpus words that occur together more often than chance (a property called concordance). However,

this test alone gives quite poor results. Some other quality measures must be taken into account, like popularity (sufficient frequency), informativeness ("this paper" is not a very informative n-gram) and completeness ("vector machine" is not considered completed, as the more common phrase is "support vector machine").
But how do we deal with collocations in Word Embeddings? A suggestion come from the very beginning of the Neural Word Embedding history, in the Mikolov article [Mikolov et al. 2013]: first we identify a large number of phrases using a data-driven approach, and then we treat the phrases as individual tokens during the training. The obvious problem with this approach is that it does not consider OOV collocations. Another possible approach is to simply combine the vectors of the collocation's words by summing or averaging it. Poliak [Poliak et al 2017] suggest a Neural Network approach inspired by Mikolov's word2vec that allows to deal with OOV collocations by first creating some components, named Skip-Embeddings, and then combining them in a way that preserve order sensitive information, to form the embedding of the collocation.
Li [Li et al 2017] proposed a different Neural Network approach, valid for any n-grams. Vectors are generated for every possible bi-gram and tri-gram in the corpus (experimenting with different tasks, like trying to predict the sentiment or a certain label). The resulting n-gram embeddings, in the view of the author, is intended for the more general purpose of Text Representation, i.e. to code large chunks of texts (not just collocations) into a single vector.

4.1.5 Lemmatization

Is lemmatization beneficial to Word Embedding extraction? (Maybe especially when the corpus is small). A lemma is the canonical form used to represent a set of words. The lemma of a verb for example is its infinite form. Considering lemmas instead of all the possible inflections of a word is a technique often used in NLP to cope with data sparsity. However, transforming all the words in a corpus to their respective lemma (lemmatization) is an expensive task. A cheaper alternative is to try to remove the derivational affixes employing some heuristic algorithm. However, this second choice is not always applicable (understemming, for example: irregular verb) or may lead to collisions (overstemming, for example {"universal","university","universe"} → "univers").
Fares [Fares et al 2017] presented a pilot experiment to empirically evaluate the impact of text pre-processing on word embeddings. Two corpora were considered (a Wikipedia dump[5] and the Gigaword Fifth Edition[6]). Intrinsic similarity evaluations were made on two datasets (SimLex-999 [Hill et al 2015] and Google Analogy Dataset [Mikolov, Chen et al 2013]). According to Fares, the models trained on the lemmatized corpora are consistently better than the full-form models.

4.1.6 Effect of the corpus size on the embedding quality

Is it true that, given a certain training algorithm, the bigger is the corpus the better is the embeddings? Zhu [Zhu et al 2017] investigated word2vec ability in deriving semantic relatedness and similarity between biomedical terms from large publication data. Medical terms with three semantic types were first selected from a corpus (disorders, symptoms, and drugs). The vectors

[5] https://meta.wikimedia.org/wiki/Data_dumps
[6] https://catalog.ldc.upenn.edu/LDC2011T07

extracted by word2vec were used to compare couples of such term, and the result was compared against a gold standard built on human judgment. So, in fact, two different things were measured: the ability of their model to identify the medical terms and the correlation between the similarity calculated with the Word Embeddings and the one perceived by the doctors. Ten models were trained separately by increasing the size of the dataset from 10% to 100%. The result was that the increasing size of the corpus was always beneficial to the term identification, whilst the correlation with human judgment had a peek (at about 3.3 million distinct vocabularies) and then slowly deteriorate.
Fares [Fares et al 2017] made a similar study and came to exactly the same conclusion: more data is not necessarily better.
Zhu also noticed that the model trained on abstracts produced better results on the correlation task, whilst the one trained on the body was better in identifying the terms. The possible explanation is that authors tend not to mention brand names of drugs in abstracts, whilst bodies of articles contains much more terms, including irrelevant ones that may degrade the performances. No matter why, this is a clear example of how the use of different sub-languages may influence the result.

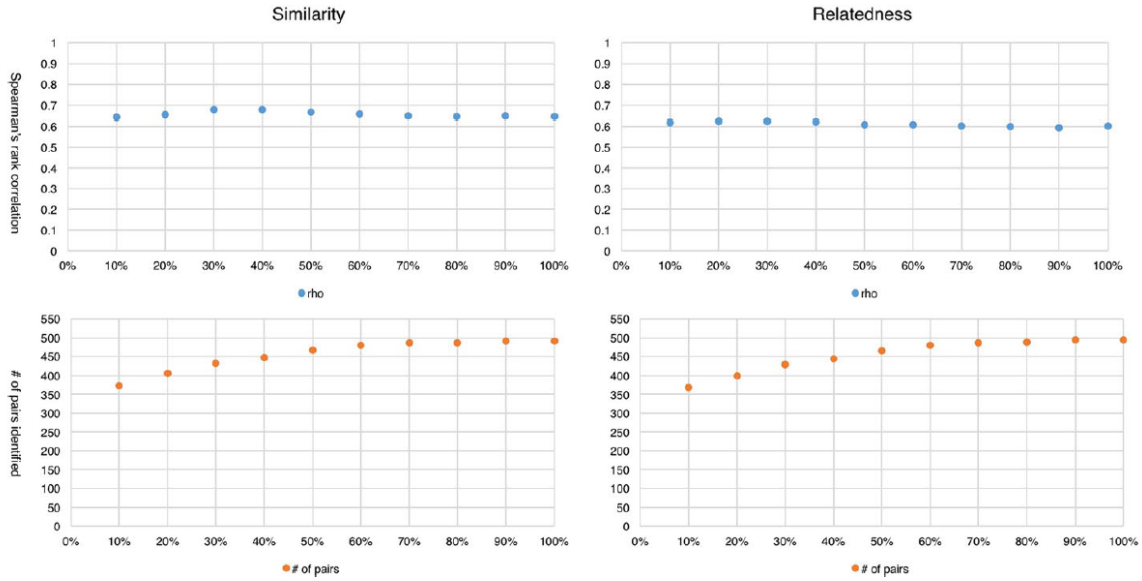


Figure 9: [Zhu et al 2017]: correlation with the human perception and number of identified terms as a function of the size of the corpus employed to train their model.

4.1.7 Custom embeddings

Is a word embedding built with a specific corpus better than a one learnt on more general data? Clearly, the corpus needs to have the information necessary for the task. A generic corpus for example may not have enough examples to extract relations among medical terms. However, in general, this may not always be the case. Wang [Wang et al. 2018] compared the Word Embeddings extracted from two medical corpora (HER, containing clinical notes, and MedLit, containing biomedical literature) with two general pre-trained Word Embeddings (Google News and GloVe). The intrinsic evaluation was a comparison with a gold standard of 165 medical term pairs whose similarity was assessed by professionals, finding the two domain Word Embeddings

performing much better than the general ones. Table 6 shows the neighborhoods of the symptom "*sore throat*", according to the four Word Embeddings, partially explaining such finding. However, extrinsic evaluations gave different results. Three downstream tasks involving the four Word Embedding was tested: a Text Classification problem, an Information Retrieval problem and a Relation Extraction one. The conclusion in this case was that there might be no significant difference when word embeddings trained from an out-domain corpus are employed for a biomedical NLP application.

Table 6: Most similar word to the symptom "sore throat", according to 4 different Word Embeddings, from [Wang et al. 2018]

EHR	MedLit	GloVe	Google News
scratchy	runny	shoulder	soreness
thoat	rhinorrhea	stomach	bruised
cough	myalgia	nose	inflammed
runny	swab fecal	chest	contusion
thraot	nose	neck	sore triceps

4.1.8 Chapter Conclusions

The most important conclusion coming from this literature analysis is probably that we cannot expect a single Word Embedding to suit all purposes. Restricted domain Word Embedding are clearly “more focused” than the ones extracted from huge general-purpose corpora, but inflating a corpus as much as possible by simply adding stuff to it without cognition does not lead to good results. From the point of view of SPARQL/T language development, this is a sharp indication that the user should be allowed to use different Word Embeddings inside the same query.

Future work will examine the Word Embedding literature in search for answers that may lead to the implementation of useful operators. Is it possible to distinguish between semantic and syntactic similarities, in order to consider only one of the two (typically the former)? Can we detect modalities (necessity, possibility, …) and quantifiers? What about negations? In fact, we believe that the next most challenging task, but also the most important for many applications, will be exactly the one of identifying and correctly deal with the many form of negations.

Chapter 5 - SPARQL/T Language Introduction

In this chapter we introduce the SPARQL/T query language with an example. The exposition is intended to be informal, easy understandable by a layman computer user, largely unaware of most NLP technicalities. Part of the purpose of this research is exactly this, to "make it easy". In the next chapter a more formal description of SPARQL/T algebra will be given.

5.1 Example 1

Let's say for example that, after skimming a few documents in our corpus of complaints[7], we notice a recurrent case: clients are noticing increases of cost, and stating that they were not aware of them and of their reason. So, the concepts we want to identify in the text are two: <cost increase of something> and <lack of awareness of something>. Moreover, we are interested in those cases where the two <something> refer to the same object.

We can start with a simple dictionary search, looking inside the text for any word that may be synonym of 'price'. Assuming we have a list in mind (like '*cost*', '*bill*' and '*charge*'), we can use the following triple pattern to assign the result to the variable p:

```
?p wrd:any 'price cost bill charge' .
```

However, enumerating all the possible synonyms of a word is obviously cumbersome and error prone. Something is going to be left out for sure. We have two other options[8]:

1. We can use WordNet synsets. The following triple pattern searches in the document (and assign to the variable p) all the words in the text that, according to WordNet[9], are synonyms of '*price*':

   ```
   ?p wn:synonym  'price'
   ```

2. We can also use a Word Embedding. A Word Embedding is simply a dictionary that assigns a vector of numbers to each word of a vocabulary. However, these numbers in a certain way encode the meaning of the words, and can be used to measure the similarity between any two words. For example, the following triple pattern assign to the variable p all the words in the text that are similar to the word '*price*'.

   ```
   ?p emb:any     'price'
   ```

Both solutions have some drawbacks. The point is that, for practical purposes, we are not normally looking for synonyms, we are looking for equivalent concepts, which strongly depends on the context. Think for example at the word '*house*'. Is '*flat*' equivalent to '*house*'? And what about a '*villa*'? They are all buildings, and from the point of view of connecting them to the fiber optic network, they are certainly equivalent. But from a real estate point of view, i.e. from the perspective of someone that want to buy them, they are certainly not. WordNet can be a good

[7] Or by employing some form of topic mining.

[8] Actually three, when OWL ontologies will be implemented.

[9] Returning any word of any synset which also contains the specified word. Word Sense Disambiguation is at the moment not implemented.

option in some cases, but not in general. Word Embedding on the other hand are trained to return a high degree of similarity for words that frequently appear in the same context, which is exactly what we are looking for[10]. However, when working with similarity instead of crispy equality tests, a new situation arises. The result is not a set of exact results, as we may desire, but a list of results ranked (ordered) by their similarity with the given term. In other words, we are going to have in return a few good results on top, followed by a long list of useless stuff. This is expected: it is due to the fact that Word Embeddings use vectors to measure similarity, so any word is semantically similar to any other, at least to some little extent. Later we are going to see how SPARQL/T deals with this problem.
Now let's use Word Embedding also to find the verb, that should be a synonym of '*increase*'.

```
?i emb:any 'increase raise' .
?i nlp:pos 'verb' .
```

Let's notice two things:

- In the first clause, we suggested two words: '*increase*' and '*raise*'. The effect is that all the words of the text will be compared with both, and the best results will be returned in the variable i. In principle this would not be necessary, as we expect '*increase*' and '*rise*' to be similar. However, in general we cannot know this for sure. It depends on the Word Embedding employed (which, by the way, may also miss some words). With this trick, we just try to make the query more robust.
- The second clause share the same variable `i` of the first. It actually behaves like a filter, restricting the words resulting from the first clause to the verbs[11].

What happen when we place the three clauses together? The result is a table with two columns, one for each variable declared (`p` and `i`), and the rows of the table are all the possible combinations of the rows of `p` with those of `i`. This is very similar to what happens in relational databases when we perform a cross-product, i.e. when we join two tables together without specifying a key. However, in this case we are dealing with natural language, not with structured data, and we have the following desire: that `p` (the price) is the object of `i` (the increment). How can we make sure of that? As we will see in the next examples, there are techniques that look into the syntax of the sentence[12] and of the discourse[13]. However, a quick and dirty method that often gives surprisingly good result is to simply look at the distance (number of words) between the two. Intuitively, the more far apart two words are in a document, the less likely is that they are related to each other. Again, SPARQL/T translate this idea into a ranking, an ordered table where the (hopefully) correct results are placed on top. So now, the order of our set of couples of words (`p`,`i`) will be a function of three things:

- How well `p` represents the concept of price
- How well `i` represent the concept of increment
- How close is `p` to `i`.

[10] Provided the corpus on which the Word Embedding has been trained is rich of the context considered
[11] Technically, imposing that the Part Of Speech (POS) tag associated to the token must be a verb.
[12] At the moment, Dependency Parsing and Semantic Role Labeling
[13] To be implemented: Anaphora Resolution

Actually, what we have requested here is that `p` and `i`, besides being as close as possible, have belong to the same sentence. This kind of requirement is so common that in SPARQL/T it is the default behavior. However, we will see soon how to avoid it, just in case.
Proceeding in the same way we just did for the first concept (<cost increase of something>) let's now write the two clauses that will allow us to extract the second concept (<lack of awareness of something>):

```
    ?k elm:any     'know notice aware' .
    ?n wrd:any     'not never without' .
```

Here the first row uses a Word Embedding to extract from the document words whose meaning is similar to '*know*', '*notice*' or '*aware*', whilst the second one looks for (exact) words that may express a negation. Notice the prefix `elm:`, employed now in place of `emb:`. What we are asking here is to consider the lemmas instead of the words. The lemma of a word is a sort of root from which other forms are derived. It is the word that typically appears on the dictionary. For verbs for example, it is its infinite form. If we are interested in comparing the meaning of two words, using their lemma usually leads to better results, avoiding the Word Embedding to focus on syntactic similarities. About the negation, just looking for a 'not' in proximity of our verb is certainly not a safe procedure. The 'not' we find may easily refer to something else, or the negation may be expressed in other ways. However, it often works! (But again, we will see in the next examples how considering the structure of the sentence may help on this).
Now, let's place together the five clauses we have written up to now, in the way that follows:

```
{   ?i elm:any     'increase rise' .
    ?i nlp:pos     'verb' .
    ?p elm:any     'price cost bill charge' .
}
{   ?k elm:any     'know notice aware' .
    ?n wrd:any     'not never without' .
}
```

Notice the use of the curly braces. As mentioned before, the default behavior of a set of clauses[14] in SPARQL/T is to force everything inside them to belong to the same sentence. Conversely, if we want to allow two things to belong to different sentences, all we have to do is enclose them in different blocks, i.e. in different pairs of curly braces. But how are the results of two blocks joined together?
As stated in the beginning, we aim at documents where the two concepts (<cost increase> and <lack of awareness>) are related. And again, we resort to proximity as a measure of relatedness between concepts. Now, the result is going to be a table with four columns (`i`,`p`,`k`,`n`), and the order of the rows will depend on three things:

- How well a couple (`i`,`p`) expresses the concept of <cost increase>
- How well a couple (`k`,`n`) expresses the concept of <lack of awareness>
- How close the two concepts are expressed in the document

[14] Technically, of a Basic Graph Pattern (BGP)

To conclude our example, let's say that we want to catch a totally different way of expressing both concepts together: <hidden fees>. In SPARQL/T the UNION clause has exactly this purpose: allow syntactic alternatives of the same meaning. The two clauses expressing the <hidden fee> concept are reported at the bottom of the full query in Figure 10 , and at this point they should need no explanation.

```
SELECT ?i ?p ?k ?n
WHERE
{  {  {   ?i elm:any     'increase rise' .
          ?i nlp:pos     'verb' .
          ?p elm:any     'price cost bill charge' .
     }
     {    ?k elm:any     'know notice aware' .
          ?n wrd:any     'not never without' .
     }
    }
    UNION
    {     ?k emb:equ  'hide' .
      ?p emb:equ  'fee' .
    }
}
```

Figure 10: an example of SPARQL-T query

Notice instead the list of four variables (i,p,k,n) included in the SELECT clause. They are the output columns of the query and they will normally contain the information we really intended to extract from the documents. In this case, they might have been for example "what exactly is costing more", and "how much".

One final consideration remains to be done: how many results is the query going to return? SPARQL/T examines one document at a time. For each document, at each step of execution[15] it keeps only a limited number of results, the top N of the table. When combining two tables[16], if the results are more than N, the best N are kept and the other discarded[17]. At the end of the search, SPARQL/T returns only one record per document, the one on the top of each document's results, and then again sorts the output and keeps only the best ones. This kind of behavior is necessary in order to avoid exponential growth and intractable queries, but unfortunately is made at the risk of missing some good results. So, the number N of results kept at each step should be considered with care, as both speed and efficacy highly depend on it.

[15] For each triple pattern.

[16] For each join operation.

[17] Beam search.

Chapter 6 - SPARQL/T Conceptual Model

This chapter formally describes the SPARQL/T language, as opposite to the previous chapter that introduced it with some examples. The actual formal description is given in chapter 6.5, which is preceded by some subchapters that explain some issues and motivate some choices.
Chapter 6.1 briefly discusses the idea of Conceptual Model itself, trying to establish what it should contain and what not. Chapter 6.2 gives an overview of the main principles of operation of the language, and briefly exposes the main differences from the SPARQL for RDF ones. Chapter 6.3 introduces the concept of uncertainty, which is not present in SPARQL, the Fuzzy Set approach adopted in SPARQL/T and some issues that are still debatable. Chapter 6.4 highlight some more issues and differences between the Text and the RDF models. Then chapter 6.5 formally illustrates the SPARQL/T conceptual model, exposing it in a side-by-side fashion with the model of SPARQL (as reported by the W3C), allowing thus the reader to easily compare the two. Finally, chapter 6.6 exposes the parts that are peculiar to the SPARQL/T Model, that finds no counterpart in the SPARQL for RDF one.

6.1 About Conceptual Models

This chapter presents the Conceptual Model of SPARQL/T, highlighting for clarity the differences from its SPARQL for RDF counterpart. But how exactly should its model look like? What should be included, and what instead should better be excluded to get an elegant conceptualization of the language? Some guidelines are found in the literature. In a general sense, according to Brambilla et al. [Brambilla et al 2017], a model can be informally defined as "*a simplified or partial representation of the reality, defined in order to accomplish a task or to reach an agreement on a topic*". More specifically, in the field of Information Systems, Griethuysen introduced in 1982 the "principle of conceptualization", [Griethuysen 1982] stating that (as reported by Olivé [Olivé 2007]) "*A conceptual model should only include conceptually relevant aspects […] of the universe of discourse, thus excluding all aspects of […] data representation, […] message formats, data structures, etc.* "
The Object Management Group (OMG) is even more specific, suggesting also that modeling should be organized into three levels:

- Computation Independent Model (CIM)
- Platform Independent Model (PIM)
- Platform Specific Model (PSM)

The first level (CIM) should represent what the solution is expected to do, hiding all IT-related specification. The last level (PSM) should include all of the details necessary to produce an implementation, and the middle level (PIM) has been introduced to deal with different platforms.

Because SPARQL/T current implementation is in pure Java, and thus already (almost) platform independent, and because all the annotations are made offline[18], platform is not an issue and we will only need to focus here on the Computation Independent part of the model.
Therefore, according with the above-mentioned principle of conceptualization, all the nuisances related to the specific NLP algorithms and tools have been left out of the model and considered implementation details. The user should only be concerned with the high-level notions of the different NLP tasks, available in the form of triple patterns, and thus isolated from any variation, imposed for example by the language. This approach, besides leading to a simpler model, has the advantage of simplifying the improvements of the product as long as new, more effective algorithms become available. However, there are issues from which the user should not be kept apart. It's a matter of fact, when dealing with Natural Language, that uncertainty is present at every corner in multiple forms, and it is my opinion that such uncertainty must be representable in the model, and actually be a prominent part of it, as otherwise the ability of the model to represent reality becomes seriously compromised. The user should be well aware of all the situations that may arise uncertainty, and also of the fact that it propagates in some way all along the query execution, up to the final result. But again, we should also be careful to avoid to commit to a too much specific way to treat it, as it may be hard to change it in case a more effective one is found.

6.2 Principles of Operation

SPARQL/T is basically a query language for template extraction from text that aims at presenting the user an interface to the most useful NLP task, in an easy to write, declarative fashion. In other words, it allows to specify what has to be extracted, hiding how it is actually done. Its syntax is a subset of the SPARQL 1.1 one, but the semantic is slightly different. Triple patterns are evaluated against an NLP annotated document[19], and not against an RDF graph like in SPARQL. A SPARQL/T triple pattern expresses an NLP annotation to look for in the text, not a pattern to match in a graph. For homogeneity with the SPARQL for RDF model, we are still going to call them triple-patterns, even though extraction-patterns would probably be more appropriate. Figure 11 gives an example of such triple patterns.

```
?n    wrd:any     'not never without' .
```

Figure 11: An example of triple pattern: 'wrd' is the NLP function group, 'any' is the subfunction, 'not never without' is the argument of the function

Like in SPARQL, variables and literals are allowed. The formers are bound with snippets of texts during query evaluations, are used in the join operations and becomes the output of the query. The latter are used mainly as constant parameters of NLP functions.
Despite the different contexts and the presence of the uncertainty that characterize Natural Language, the modus operandi of the two languages and the queries results look very similar (see Figure 12). In a sense, we can think that SPARQL/T evaluates NLP triple patterns against a **virtual**

[18] In a Linux environment, generating a (temporarily) proprietary format JSON file.
[19] Annotation should normally be performed offline before query execution, but of course this is more an implementation consideration, not a part of the Conceptual Model.

graph that is still embedded in the text (consisting of the text and its annotations), extracting from it a table of results or some RDF subgraphs, exactly as its standard counterpart.
The SPARQL/T language has been designed to resemble as much as possible its RDF counterpart, not only in the syntax but also in the semantic, at least as much as the different context allows. The use of most constructs should be intuitive to the user familiar with SPARQL. For example, the `UNION` keyword in the RDF case has been provided as "*a means of combining graph patterns so that one of several alternative graph patterns may match.*"[20]. In SPARQL/T the things that the `UNION` clause allows to combine are not graph patterns, but different ways to express a concept in Natural Language words, which is anyway a very similar idea. However, due to the need to deal with similarity instead of with exact matches, some differences from the RDF case are unavoidable also at conceptual level. For example, the `GROUP BY` keyword may involve a clustering of the results (instead of a simple aggregation), with the number of desired clusters indicated with the `LIMIT` keyword, and the `AVG` function calculating the average of the Word Embeddings and returning the result closed to that average.

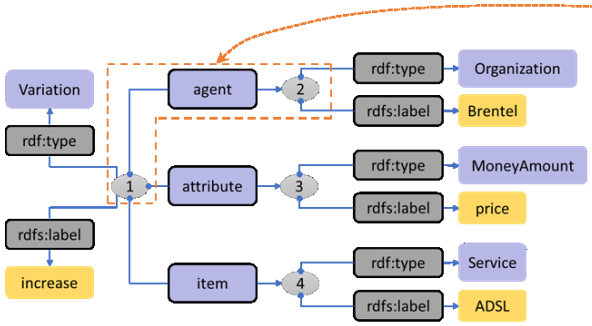
Like in SPARQL, triple patterns in SPARQL/T generates relations, i.e. tables of up to three columns (one for each variable) containing the results extracted from the text. Relations of different triple patterns are combined by the Relational Algebra operations to form new ones, until the final result set of the query is obtained. Figure 14 gives an idea of how the three major operations of the SPARQL/T algebra, the `JOIN`, `UNION` and `MINUS`, work in the text case. The four relations in Table 8 are hypothetical extractions from the documents in Table 7.
All three operations are performed in three phases:

- The tuples of two relations are matched to form the output ones. The match is performed on the columns (variables) shared by the two input tuples (in this case, always the column `food`). The weights of the output tuples are calculated according to the input ones[21]. The approximate formula (that do not take into account the degree of confidence in the match), are reported in the three cases. Assuming the standard Fuzzy norms and negator:
    - The Join takes the minimum
    - The Union takes the maximum
    - The Minus decreases the left by the amount of the right
- The output results are sorted according to the weight
- The tail of the results is removed (see Chapter 6.4.6 about Memory Constraints)

[20] https://www.w3.org/TR/rdf-sparql-query, chapter 7 Matching Alternatives
[21] And to the degree of confidence in the match, not reported here for simplicity

RDF Case

RDF Graph

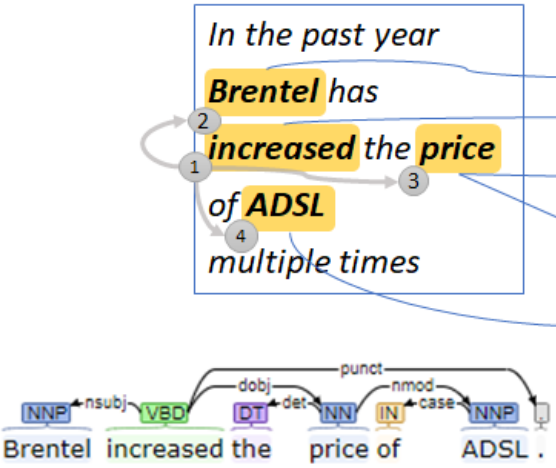


SPARQL Query

```
SELECT ?Org ?Var ?What ?Item
WHERE
{    _:N1  rdf:type        :Variation
     _:N1  rdfs:label      ?Var
     _:N1  p:Agent         _:N2
     _:N2  rdf:type        :Organization
     _:N2  rdfs:label      ?Org
     _:N1  p:Attribute     _:N3
     _:N3  rdfs:label      ?What
     _:N1  p:Item          _:N4
     _:N4  rdfs:label      ?Item
}
```

Text Case

Text+Annotations ≅ Graph



SPARQL/T Query

```
SELECT ?Org ?Var ?What ?Item
WHERE {
?Org   NLP:NER 'organization'
?Var   ELM:ANY 'increase rise' .
?Var   NLP:POS 'verb' .
?Var   DEP:dobj ?What
?What  ELM:ANY 'price cost' .
?What  DEP:nmod ?Item .
}
```

Results

RDF-SPARQL

?Org	?Var	?What	?Item
Brentel	increase	price	ADSL

Text-SPARQL/T

?Org	?Var	?What	?Item
Brentel	increased	price	ADSL

Figure 12: Representation of the idea of Virtual Graph embedded in the text. The queries are different, but the syntax of the two languages is the same. The logic adopted by SPARQL/T queries to extract elements of the text is, to the user, very similar to the one that SPARQL use to extract RDF subgraphs. Despite the very different scenario and the uncertainty of Natural Language, very similar results are expected to be achievable.

idDoc	Document
1	For lunch I'm going to eat a sandwich with chicken and tomatoes
2	Can I have chips with my steak please?
3	Let's take a slice of that apple pie.
4	Take the street in front of the restaurant
5	What about fish and chips for dinner?
6	I'm not used to vegetarian meals

Table 7: documents for the example in Figure 14

Eat - Food			
idDoc	W_{Eat}	?eat	?food
1	0.8	eat	sandwich
2	0.7	have	steak
3	0.7	take	pie
4	0.6	take	restaurant

Food - Ingredients			
idDoc	$W_{Ingredients}$	?food	?ingredient
3	0.8	pie	apple
1	0.7	sandwich	chicken and tomatos
2	0.6	steak	chips

Meal - Food			
idDoc	W_{Meal}	?meal	?food
1	0.8	lunch	sandwich
4	0.8	dinner	fish and chips
5	0.6	meal	vegetarian

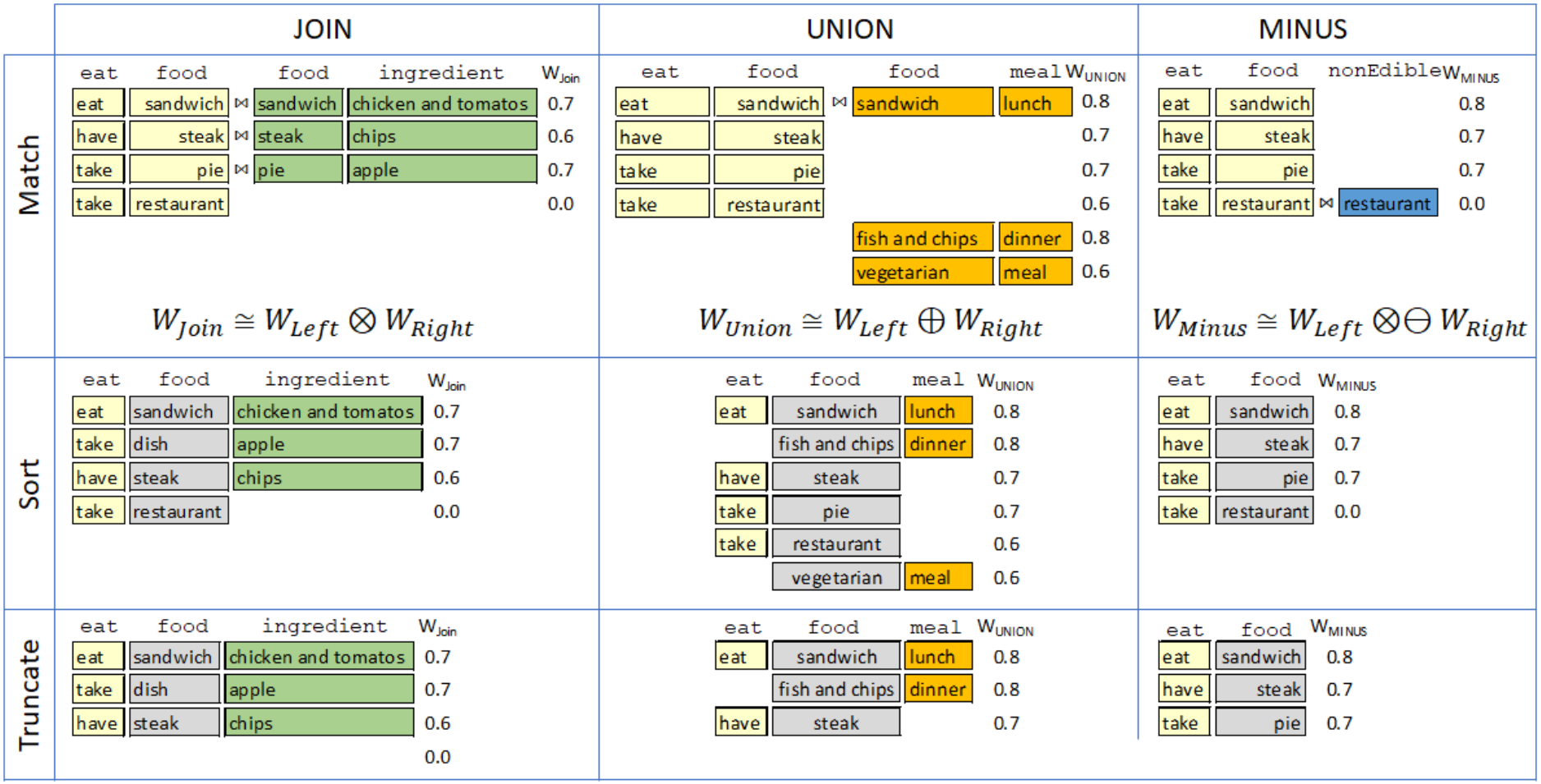
Non Edible		
idDoc	$W_{NonEdible}$	?nonEdible
4	0.8	restaurant

Table 8: Relation extracted from the documents in Table 7 and combined by the algebra operations in Figure 14. For the purpose of explaining the algebra operations, the actual triple patterns that can be employed to extract them are not meaningful. However, just to give the idea, Figure 13 reports a query to extract the first one (Eat-Food).

```
?eat   ELM:ANY     "eat"
?eat   NLP:POS     "verb"
?food  ELM:ANY     "food"
```

Figure 13: possible query for the extraction of the relation "Eat-Food" of Table 8. (Results depend on the specific Word Embedding employed)

Figure 14: Example of how the JOIN, UNION and MINUS Relational Algebra operation work, applied to the relations in Table 8, extracted from the documents in Table 7



6.3 Ambiguity of language and Uncertainty of results

It is a very well-known fact that Natural Language is highly ambiguous. Words very often possess different meanings, co-references between words are uncertain, sentences have different interpretations that depend on the context, and so on. Ambiguity is part of the reality of Natural Language, and not just the unavoidable byproduct of still imprecise algorithms. As such, in my opinion, ambiguity should be represented in the model, and not forcefully solved as soon as possible with a flip of a coin by choosing the most likely option. Without a way of representing ambiguities, the model will simply be not enough expressive to represent reality. Being SPARQL/T a query language and not a modeling one, our concerns are limited to take into consideration ambiguity in the results, ranking them accordingly. However, we must be careful not to fall into what Dubois & Prade call "*the unfortunate confusion between degrees of belief and what logicians call degrees of truth*" [Dubois & Prade 2001]. An example that is often used to illustrate the differences between the two is the one of the bottle [Bezdek & Pal 1996]. A bottle can be full, empty, half-full and so on. If we consider the sentence "the bottle is full", we can attach to it a degree of truth between 0 and 1, say for example 0.7, to state that the bottle is not really full. However, such degree of truth reflects the amount of liquid in the bottle, not the degree of belief in the fact that the bottle is full. It does not mean, in other words, that the probability of finding the bottle full is 0.7. There is no uncertainty about the amount of liquid the bottle contains, it is more a matter of definition of what we mean for "full". Logics that consider more than two values of truth are called Many-valued logics. Fuzzy Logic, among these, allows infinite ones, expressed with a real number in the range [0,1]. It seems appropriate thus, to employ Fuzzy Logic to deal with language ambiguity and NLP tools uncertainty. But representing such things is not just a simple matter of attaching a weight to a result (for example, the degree of confidence in an annotation). We also need to define how such weights can be calculated when results are combined (i.e., during the join operations). In other words, **compositionality is necessary**. Formally, a weighted logic or inference system is said to be compositional if and only if the weight of a complex formula can be calculated by combining the weights of its atomic constituents. Unfortunately, as shown by Dubois & Prade [Dubois & Prade 2001], this may not always be possible, or at least not with a good theoretical foundation. According to them, only partial true can be represented in forms that allow compositionality, whilst partial belief cannot. To say it in their own words: "[…] *not only the full compositionality of any uncertainty calculus is not possible, but retaining this property as much as mathematical consistency allows, only leads to a very crude, almost deterministic representation of belief*".

To briefly illustrate this point (see their article for a complete demonstration), let's show that it is not possible for example to compute the degree of belief of a disjunction from the degrees of belief in its components.

Let $N(p) \in \{0, 1\}$ denote the (Boolean) degree of belief of p. By convention, let $N(p) = 1$ when a certain Belief Base K proves p, and 0 otherwise. Let's say that $N(p) = 0$ and $N(q) = 0$, i.e. an hypothetical agent knows nothing about the values of p and q, and try to determine $N(p \vee q)$. Let's consider two special cases:

- If p = q and then N(p ∨ q) = N(p ∨ p) = N(p) = 0 .
- If q = ¬p the agent must believe p ∨ ¬p (since it is a tautology), hence N(p ∨ q) = N(p ∨ ¬p) = 1

Hence, the degree of belief in p ∨ q when N(p) = 0 and N(q) = 0 can be 0 or 1, i.e. it cannot be calculated.

All that said, what do we need to represent in SPARQL/T? Truth values or degree of confidence? Let's consider the most prominent cases:

1. Annotations are subject to error, and sometimes (although not so often) a measure of confidence may be provided by the annotation algorithm.
2. For different reasons (typically for joins), we may want to compare two snippets of text to check if they are "almost the same" (as we do not expect them to be exactly the same). In these cases, a measure of overlapping between the two is employed.
3. Let's say we want to find two related words in the documents, like a verb and its subject. In absence of a better indicator (like a parse tree) a noisy but often good enough measure of relatedness is their relative distance (the number of words in between them).
4. Triple pattern involving Word Embeddings compares the Word Embeddings of the words in the document with those listed in the triple pattern. The best matches (according for example to the cosine similarity measure) are ranked higher.

The situation appears not so sharp. Confidence in annotation (example 1) is clearly more related to uncertainty (lack of complete knowledge), whilst Word Embeddings, as a measure of similarity between concepts, is more related to the concept of truth, or better, in Fuzzy Logic terms, to the idea of belonging to a certain set. The two middle cases are more debatable. However, it seems acceptable to see both of them as expressions of fuzziness (similarity in case 2 and relatedness in case 3) instead of as a lack of knowledge. Thus, giving up the idea of keeping track of the quality of the annotations, let's embrace Fuzzy Logic and its notation to represent Natural Language ambiguity.

What follows defines the three operations used in the framework of Fuzzy Logic, as defined for example in [Beg & Ashraf 2009]

Definition (t-norm and t-conorm)
A triangular norm (t-norm) $\otimes$ and a triangular co-norm (t-conorm) $\oplus$ are increasing, associative, commutative and $[0,1] \times [0,1] \to [0,1]$ mappings satisfying $x \otimes 1 = x$ and $x \oplus 0 = x$ for all $x \in [0,1]$

Definition (negator)
A negator $\ominus$ is an order reversing $[0,1] \to [0,1]$ mapping such that $\ominus 0 = 1$ and $\ominus 1 = 0$

The standard t-norm, t-conorm and negator are defined as follows:

Standard (Łukasiewicz) negator	$\ominus x = 1 - x$
Standard (Gödel) t-norm	$x \otimes y = \min(x, y)$
Standard (Gödel) t-conorm	$x \oplus y = \max(x, y)$

Other possibilities, as described for example in [Straccia 2013], are:

Gödel negator	$\ominus_G x = \begin{cases} 1 & if\ x = 0 \\ 0 & otherwise \end{cases}$
Bounded difference or Łukasiewicz t-norm	$x \otimes_L y = \max(0, x + y - 1)$
Algebraic product or product t-norm	$x \otimes_P y = x\ y$
Drastic product	$x \otimes_D y = \begin{cases} 0 & if\ (x,y) \in [0,1[\times [0,1[\\ \min(x,y) & otherwise \end{cases}$
Bounded sum or Łukasiewicz t-conorm	$x \oplus_L y = \min(1, x + y)$
Algebraic sum or product t-conorm	$x \oplus_P y = x + y - xy$
Drastic sum	$x \oplus_D y = \begin{cases} 1 & if\ (x,y) \in]0,1] \times]0,1] \\ \max(x,y) & otherwise \end{cases}$

T-norms generalize the Boolean operation of conjunction, and will be used here in the Join operation. Intuitively, the Join operation implements a sort of conjunction of the results of two triple patterns, so the degree of truth of the result, even without considering the fuzziness introduced by the join itself, should not be higher than the degree of truth of any of the sources. Formally, let t_1 and t_2 be the two tuples to be joined, and $W(\cdot)$ be the function that returns the weight (truth value) of a tuple. We should respect:

$$W\big(Join(t_1, t_2)\big) \leq \min\big(W(t_1), W(t_2)\big) = W(t_1) \otimes W(t_2)$$

Which suggest defining, for an $\alpha_J \in [0,1]$ that represent the truth value of the join:

$$W\big(Join(t_1, t_2)\big) = \alpha_J \otimes W(t_1) \otimes W(t_2)$$

Similarly, t-conorm generalizes the Boolean operation of disjunction. In SPARQL/T the UNION clause represents the disjunction of two sets of solutions, which intuitively suggests that if a tuple t appears in both, it should appear in the result with a weight that is not less than in any of the two source sets. Formally, let t_L and t_R be two tuples belonging to the result set of the left and the right part of the UNION clause respectively. When t_L and t_R are judged to refer to the same object, a single tuple t should replace both, with a weight respecting the following:

$$W(t) = W\big(Union(t_L, t_R)\big) \geq \max\big(W(t_L), W(t_R)\big) = W(t_L) \oplus W(t_R)$$

6.3.1 Some (open) issues due to similarity comparisons

In SPARQL for RDF, when tuples from two input relations need to be matched (by the various Relational Algebra operators), the couples are made if and only if the values corresponding to

the common variables of the two relations (the keys) are exactly the same. In SPARQL/T instead, because we are not expecting to always find exact matches, a more robust approach based on similarity has been adopted. However, no matter what measure of similarity we choose, this choice brings two issues:

- Which couples should we consider?
- In the output relation, what is the correct value to give to the key?

Let Ω_L and $\Omega_R$be the left and right relation sets involved in the operation, of cardinality M and N respectively. For the simplicity of the exposition, let's assume they only have one variable V in common, and let v_L^i be the value of V in the i-th tuple $\lambda_L^i$of Ω_L, let v_R^j be the value of V in the j-th tuple $\lambda_R^j$of Ω_R, and let $w_{i,j} = W_M(v_L^i, v_R^j)$ be the similarity measure between the two.
The first issue is due to the fact that, switching from equality to similarity, even relationships that were of the one-to-one kind unavoidably becomes of the many-to-many kind. With similarity comparison instead of equality, each value v_L^i will match, in general, many values v_R^j , with different degree of truth. What is, then, the best way to form the output tuples $\lambda_O^k = (\lambda_L^i, \lambda_R^j)$? One approach is to generate all the possible couples with similarity greater than zero, calculate their weight (that depends also on the weight of λ_L^i and λ_R^j) and then choose the best ones (to fulfill the Memory Constraints search requirement, see Chapter 6.4.6). A different approach is to form couples by choosing only one tuple per set, trying for example to maximize the sum of the similarities of the chosen couples. At the moment is not clear which approach is best.
The second issue can be re-stated in this way: if $\lambda_O^k = (\lambda_L^i, \lambda_R^j)$ is kth output tuple, but the similarity between λ_L^i and λ_R^j is strictly less than 1, i.e. $v_L^i \neq v_R^j$, which value should we use for v_O^k? Choosing one of the two appears quite arbitrary. Let's considering instead that such values represent snippets of text, i.e. intervals of the document, and the fact that they are similar already involves, in a way or another, a certain overlap between the two. Thus, only two other choices seem reasonable: their union and their intersection. In this case the right choice exists, at least in principle, depends on the operation considered, and is dictated by the need of maintaining the associative and commutative properties of the operators. For the Join operation, the key of the output relation should be the intersection between the keys of the input ones. This become apparent if we consider the Join of three sets Ω_1, Ω_2 and Ω_3, all of them sharing a single variable V. In principle, we would like:

$$(\Omega_1 \bowtie \Omega_2) \bowtie \Omega_3 = (\Omega_1 \bowtie \Omega_3) \bowtie \Omega_2$$

Equation 1

As an aside, this is not going to happen anyway in practice, because of the Beam Search employed during evaluation (see Chapter 6.4.6 about memory contraints). However, performing the Union of the keys in the Join operation can only make the things worse. Let's call v_1^i, v_2^j and v_3^k the snippets of text corresponding to the variable V in the sets Ω_1, Ω_2 and Ω_3 respectively. As already stated, two snippets of text v' and v'' are similar, to some extent, if and only if $v' \cap v'' \neq \emptyset$.

Now, let's say that the function that generates the new key of a join operation is the union of the snippets, i.e.:

$$Key^*_{Join}(v', v'') = v' \cup v''$$

Equation 2: WRONG keyword generator for the Join operation

To show that it is not a good choice, let's say that there is no couple of similar elements between Ω_1 and Ω_2 , but that exist v_3^k which is similar to both, i.e.:

$$\forall i, j \; v_1^i \cap v_2^j = \emptyset$$

Equation 3

$$\exists i, j, k \; v_1^i \cap v_3^k \neq \emptyset \wedge v_2^j \cap v_3^k \neq \emptyset$$

Equation 4

Equation 3 means that, no matter the choice of the Key function, the left term of Equation 1 is the empty set ∅, as $\Omega_1 \bowtie \Omega_2 = \emptyset$.
Equation 4 on the other hand means that $\Omega_1 \bowtie \Omega_3 \neq \emptyset$. Let $v^*_{1,3}$ be the key generated by the Key^*_{Join} functions with the elements *i* and *k* of Ω_1 and Ω_3, i.e.:

$$v^*_{1,3} = Key^*_{Join}(v_1^i, v_3^k) = v_1^i \cup v_3^k$$

Obviously $v^*_{1,3} \cap v_2^j \neq \emptyset$, so the right term of Equation 1 has a solution, and the equality does not hold. In simple words, with the Key^*_{Join} function a snippet of text in the Ω_3 result set may act as a "bridge" between two solutions in Ω_1 and Ω_2 , making the result dependent on the order of evaluation. This doesn't happen if we define:

$$Key_{Join}(v', v'') = v' \cap v''$$

Equation 5: Correct key generator for the Join operation

Similarly, we can show that for the Union operation, the key of the output relation should be the union of the keys of the input ones**.**

In summary, and perhaps not surprisingly, **for the Join operation the output keys should be the intersection of the input ones, whilst for the Union operation they should be the union of the input ones**. For the Minus and Optional operations there is no new tuple generation, only a variation of the weights, therefore the keys remain unchanged.
As a final observation however, it is likely that performing the Union of the keys add robustness to the process, partially correcting possible annotators imprecisions (in scope). Moreover, as already noticed, commutative and associative properties of the Join operation in SPARQL/T, from the practical point of view, have been already given up in favor of speed and reduced memory consumption. For these reasons, we prefer to leave as an open issue which operation is really more suitable for the keys during the Join evaluation.

6.4 Main Differences between the Text and the RDF Models

6.4.1 Results are Text References and not URIs

The major difference between the text and the RDF case, for the conceptual model point of view at least, is that the results of a SPARQL/T SELECT query are not URIs[22], but references to snippets of text inside the document. Each single SPARQLT/T triple pattern extract snippets of text from the document and produces a set of tuples, with one or two elements each (depending on the number of variables involved), binding the variables with a set of references to the document itself. Formally, adapting the W3C terminology[23], a binding is here a pair *(variable, TextReferenceObject)*, where *TextReferenceObject* is an instance of a class derived from the abstract class *TextReference,* which represents a snippet of text, and contains at least two things:

- The position[24] *(beg,end)* of the snippet inside the document
- A weight that represent the confidence in the extraction (or better, its truth value, see Chapter 6.3 about Ambiguity and Uncertainty)

TextReference has three possible implementations (see Figure 15 for the UML representation):

- TextWord: represent a single word (token)
- TextSnippet: represent a sequence of contiguous words in the text (n-grams)
- TextSnippetsSet is a set of possibly noncontiguous TextSnippets

Which implementation of *TextReference* is actually used depends on the NLP function employed. For example, the Named Entity Recognition (NER) task consists in finding entities like person and company names, which are often multi-words elements. This implies that NER triple pattern need to returns n-grams (TextSnippet objects). A Part Of Speech tag (POS) on the other hand, is a label applied to every single word in the text (by a POS tagger), which allows to employ TextWord objects, that simply refers to single words in the text. The TextSnippetsSet is included here for future uses, to represent noncontiguous snippets of text that may result from the co-reference resolution task or from considering subtrees of non-projective parsing trees.

[22] However, SPARQL/T CONSTRUCT queries generates URIs exactly like SPARQL for RDF ones
[23] https://www.w3.org/TR/rdf-sparql-query/, fetched May 3, 2019
[24] The position inside the text is specified in two ways: the indexes of the first and last characters in the plain text and index of the token in the tokenized text

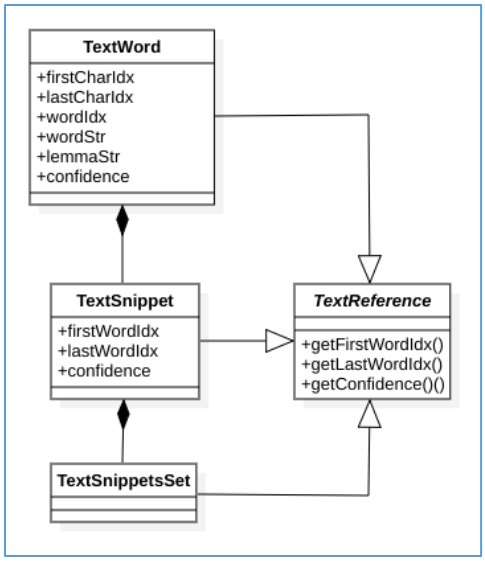


Figure 15: UML representation of the classes of the objects in the result set of a pure SPARQL/T query (URI required for hybrid Text/RDF queries not shown)

6.4.2 Input Objects

In SPARQL/T some triple patterns require an input, which means that, in some cases, some variable needs to be bound before the triple pattern is encountered in the query string. There are mainly two cases when this happens:

- When a search of some specific words has to be performed
- When fuzzy behaviors are not desirable (like in navigating a parse tree)

The concept of "input variable" is not present in SPARQL for RDF model, and in principle it could be avoided here as well. However, forcing the user to specify what to look for before the search can be performed (which, by the way, seems more than reasonable) allows for some simplification, like avoiding the need of query reordering. Moreover,expression employing the same variable twice would otherwise be allowed, like the following one, that can be read as "extract words that are most similar to each other, according to some Word Embedding":

```
?var EMB:ANY   ?var
```

Figure 16: example of forbidden "same variable" triple pattern.

Although probably interesting from some point of view, that cases would require a special treatment and, at the moment, we don't see any practical advantage in that.
Inputs always occupies its third (object) position of the triple patterns, and are called here Input Objects

Definition: **Input Object** An Input Object in a SPARQL/T query is either a literal or a variable that must be bound before the triple pattern is evaluated. It always occupies the third (object) position in the triple pattern

Being in the object position allows the input object to be either a variable or a literal. In the latter case, when allowed by the specific triple pattern, it comes in the form of a string containing a list of words or of n-grams.

6.4.3 OPTION and MINUS (positive and negative Re-ranking)

In SPARQL for RDF the MINUS clause evaluates both its arguments, then calculates solutions in the left-hand side that are not compatible with the solutions on the right-hand side and remove them. The same idea is applied by SPARQL/T to the text case, and is especially useful when similarity measures are employed. Sentence similarity in particular (like Universal Sentence Encoding) often lead to sets of surprisingly good results mixed with a bunch of easy identifiable wrong ones. In this case a MINUS clause can be used, with a pattern matching the wrong ones in its right-hand side. However, the sharp behavior of the RDF counterpart, i.e. removing from the left-hand side results those appearing in the right-hand one, should be carefully reconsidered. Two sources of errors are playing in this case. First of all, solution extracted from different NLP algorithms are hardly exactly the same, which implies that trying an exact match would be quite ineffective. Second, the definition of the wrong cases is likely to be imprecise itself, and a sort of "smoothing effect" may be appropriate. Therefore, in SPARQL/T the MINUS construct corresponds to a re-ranking of the solution of the left-hand side according to the solution of the right-hand one, decreasing their weights on the left by an amount that depends on the weights on the right and on the degree of compatibility of their match.

6.4.4 UNION vs Ranking Aggregation

In SPARQL for RDF the UNION keyword is used to allow two or more[25] alternative graph patterns. The result set of the UNION is, as expected, the union of the result sets of each of its graph pattern. **Errore. L'origine riferimento non è stata trovata.** shows an example of use of the UNION keyword taken from the W3C SPARQL 1.0 recommendations.

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { { ?book dc10:title ?title } UNION { ?book dc11:title ?title } }
```

Figure 17: example of use of the UNION keyword, taken from the SPARQL 1.0 W3C recommendation. The query finds titles of the books in the data, whether the title is recorded using Dublin Core properties from version 1.0 or version 1.1

In SPARQL/T a very similar notion is adopted: the UNION keyword is used to allow the extraction of a concept from the text when different syntactic alternatives are expected. During query evaluation, each graph pattern of the UNION is evaluated alone, extracting zero or more tuples from the text, that are merged together to form the solution of the UNION statement. A possible improvement, with respect to this classical view of the UNION clause, could be to merge similar solutions and increase the weights of those that occur multiple times. The idea is that solutions that match many user-specified patterns are more desirable than others, and should be ranked higher. This resembles the concept of Ranking Aggregation in Information Retrieval. According to Li [Li 2011], "*ranking aggregation is aimed at combining multiple rankings into a single ranking,*

[25] UNION clauses can be concatenated, like: `{} UNION {} UNION {}`

which is better than any of the original rankings in terms of an evaluation measure". However, it should be noticed that the task here is different, mainly for two reasons:

- Classical Ranking Aggregation algorithms, only consider as inputs ordered lists of results. In meta-searching for example, the results of the same query from different search engines are combined into a single, supposedly better, one. Search engines do not return the score of their result, just the ordered lists, and most Ranking Aggregation algorithms are based solely on that ordering. Here instead we still have that score, a precious value that we certainly do not want to disregard.
- We are still facing with the problem of identifying duplicate solutions that, as already noticed, in the NLP context are just expected to be similar, and not exactly the same.

As an aside, notice that the UNION clause may allow, in some cases, to implement a sort of conditional execution. Some triple patterns may return an empty relation, causing the depletion of the entire BGP's one. Se chapter 7.10.1 for an example involving a Constituency Parse tree, that allows different paths of execution on the bases of the kind of parent (Noun/Verbal Phrase) of a given snippet of text.

6.4.5 GROUP BY clause

Like in SQL and in SPARQL for RDF, the GROUP BY clause groups together records which have the same values in the columns indicated in the clause. However, as with any other SPARQL/T clauses, we cannot rely on the possibility that values extracted from text by different algorithms comes out exactly the same. We need to consider some measure of text similarity. Moreover, this similarity measure cannot be the same employed in the other functions, like for example in the join operation. The GROUP BY clause is intended to be used at corpus level, involving different documents, and therefore the idea of measuring similarity in terms of overlapping of snippets of text (of the same document) doesn't have sense anymore. We need to compare groups of different words, in a way tolerant to reordering, i.e. in the classical Bag Of Words fashion.
One possibility is to consider the number of words, or even better of lemmas, that the two BOWs share, possibly applying a weighting schema that consider stop words and the TF-IDF measure. Another, totally different approach, is to employ the already available Word Embeddings, for example by comparing the average of the vectors of the words of the to the two BOWs. In the first case the groups are crisply determined: we can say that two BOWs belong to the same set if they have some words in common. If we consider a graph G where the nodes are the BOWs and arches between nodes indicates the sharing of some words, finding the groups becomes the problem of finding all the connected subgraphs of G. (Some care must be taken, like stop word removal or TF-IDF thresholding, to avoid to obtain single, huge and quite useless groups). In the second case instead, any BOW is similar, to some extent, to any other, and the concept of grouping becomes the one of clustering.
The idea here is to provide the user with a choice between the two approaches, to be made with the use of the LIMIT keyword, which in SPARQL for RDF is used to limit the number of results of a query. In SPARQL/T the use of the LIMIT keyword in conjunction with a GROUP BY one selects the clustering approach, indicating also the number of clusters requested.

In a SELECT query, the GROUP BY clause specifies the list of variables used to form the groups, while for any other variable that we want to appear in the result set we must indicate how to aggregate the results. Figure 18 reports the seven Aggregate Functions of SPARQL. Because we want to strictly comply with SPARQL syntax, we cannot add any more.

COUNT	the number of elements
SAMPLE	any element
GROUP_CONCAT	concatenates all elements
MIN	the minimum value
MAX	the maximum value
SUM	the sum all elements
AVG	the average of all elements

Figure 18: SPARQL Aggregate Functions (aka Set Functions)

The COUNT, SAMPLE and GROUP_CONCAT easily find a counterpart in SPARQL/T. The MIN and MAX functions in SPARQL/T refer to the truth value of the tuple (not of the specific column), and return the value of the variable in correspondence respectively to the worse and the best one (If there are many tuples with the same min/max value, one is picked at random). The AVG function employs Word Embedding, averaging the vectors of the group and then selecting the result that is closer to the average. The SUM function has no counterpart in SPARQL/T.

6.4.6 Memory Constrained Search

The result of a search that employs Word Embeddings, or any other similarity measure, potentially includes all the words of the document, as they are probably all similar, albeit by a very little amount, to the search keyword (except for the few out of vocabulary ones). Moreover, when joining the results of two triple patterns that does not share variables, a cross join is performed, which produces a row in output for each possible couple of rows in input. In other words, queries involving similarity may easily become intractable. To avoid that, when appropriate, result sets are truncated (after being ranked) to the best N ones, transforming the query evaluation in a **beam search** of ray N.

To better illustrate the process, Figure 19 reports a three triple patterns BGP that should roughly extract, using solely Word Embedding, snippets of text talking about some "*payments*" that have been "*raised*" (without) the client "*knowing*" it. Figure 20 reports a possible sentence expressing this concept, followed by an unrelated one that uses words with similar meaning. The similarity measures between the words of the triple patterns and those of the sentence, returned by a hypothetical Word Embedding, are reported in the matrix below the sentence itself. For the purpose of illustration, let's assume that the similarity is high (≅ 0.8) for related words and neglectable (≅0.1) for any other couple of words.

```
?i    EMB:ANY    "rise"
?b    EMB:ANY    "payment"
?k    EMB:ANY    "knowing"
```

Figure 19: A query using only Word Embedding triple patterns. It is expected to extract snippets of text talking about some "payments" that have been "raised" (without) the client "knowing"

	Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
	Word	*They*	***increase***	*the*	***billing***	*without*	***telling***	*you*	*,*	*and*	*I*	***believe***	*they*	*are*	*going*	*to*	***loose***	*a*	*lot*	*of*	***clients***	*for*	*that*	*.*
Similarity	rise		0.8														0.7							
	payment				0.8																0.7			
	knowing						0.8					0.8												

Figure 20: Two sentences that may match the query in Figure 19. The first, with token in positions [0,6], conveys the meaning we are looking for. The second (tokens [9,21]) does not. The last three rows report, for each word of the query, the degree of similarity a hypothetical Word Embedding is likely to assign (low values omitted for readability).

At each cross-product join, the weight of each new row is calculated considering:

- The weights of the two input rows
- The mutual proximity between all the words in the resulting row

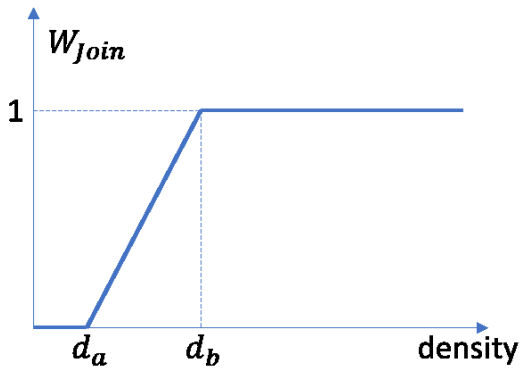
As a simple measure of mutual proximity between words we can use the "density of extracted words", roughly definable as the ratio between the number of extracted words and the number of words of the snippet[26]. Then, a fuzzy value W_{den} in the range [0,1] is obtained by applying to such density a fuzzy R-function with the two thresholds d_a and d_b as follow:

- Below d_a, W_{den} is 0
- Above d_b, W_{den} is 1
- In between, W_{den} varies linearly

Finally, the weight of the output tuple Wout is calculated as the t-norm of those of the two input tuples and W_{den} (here, adopting the Standard (Gödel) t-norm min())
To account for punctuations, a penalty is introduced by increasing the distance between words on the opposite sides of the punctuation mark (and thus the result of the density calculation). This makes sense, as in natural language punctuations are used to separate concepts. Also, when reading them, a pause of a certain length is normally introduced.
Figure 22 shows the beam search with d_a=1/20, d_b=1/2 and the punctuations penalty = 10 words. The number of rows of results kept at each step is in this case N=21. In this ideal situation, the best results appear on top at each step. The correct one is found on the very top of the output set, together with the "almost unrelated one" just below it but with a much lower score, and followed by all the irrelevant ones. Obviously, in reality things are not so clean, but the point here is that the employment of a heuristic memory constraint search seems bot an unavoidable and an acceptable choice make in order to deal with this sort of searches and joins. On the other hand, for patterns that only involve crispy NLP annotations, the number of results is normally much more manageable. Moreover, when joins involve variables the number of matches is also greatly reduced. Thus, in this last two cases, the truncation of the ranked result set should be avoided, or at least performed with a safer, much higher ray N.

[26] A better measure should weight the words according for example to their part of speech, or presence in a list od stop words.

As an aside, let's notice that the results of such Memory Constraint search depend in general on the order of evaluation. In other words, because of the Beam Search **in SPARQL/T the Join operation is neither commutative nor associative**.



$$density = \frac{N° \; of \; extracted \; words}{Snippet \; Length} = \frac{|E \cap S|}{|S|}$$

$$W_{den} = \begin{cases} 0 & if \; density \leq d_a \\ 1 & if \; density \geq d_b \\ \dfrac{density - d_a}{d_b - d_a} & otherwise \end{cases}$$

$$W_{OUT} = W_1 \otimes W_2 \otimes W_{den} = \min(W_1, W_2, W_{Join})$$

E = set of extracted words
S = set of snippet's words
W_1, W_2 = Weights of the two input tuples

Figure 21: Weight calculation of an output tuple of a Cross-Product Join. First, the density of the snippet of text is calculated as the ratio between the number of extracted words and the snippet's length. Then, a weight W_{Join} is calculated appling an R-function to the density. Finally, the weight of the output tuple is calculate as the t-norm of W_{Join} and the two input weights.

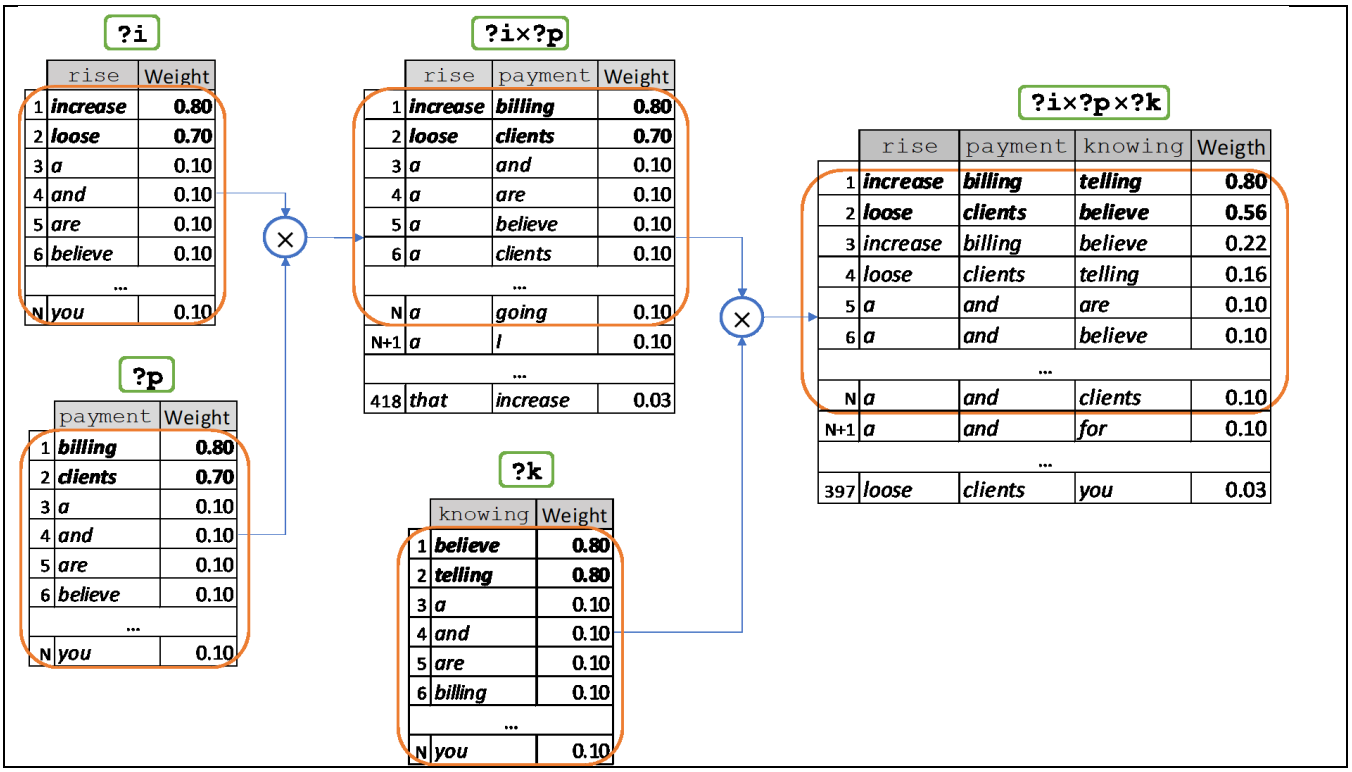


Figure 22: Example of memory constrained Cross-Product Join. At each point of join, the set of all couples is generated, ranked, and then truncated to the top N results, that are passed to the next stage. The weight of the output tuples depends on the weights of the input ones and on the density of the resulting snippet (N° Extracted Words / Snippet Length)

6.4.7 Machine Learning Classification of the results

When Word Embedding or any other similarity measure like is employed in a search, the query behaves in a way similar to an Information Retrieval system, returning a list of possible results, with the best ones on top, but that makes it difficult to make a crispy choice about which result is good and which is not. Even when the query contains only crispy requests (like searching for specific words and lemmas or POS tags) the result set may not allow a crispy classification, due to the uncertainty introduced by join operation. Normally, manual inspection allows to spot a threshold that allows an easy separation, with a reasonable margin of precision. Sometimes, it is also possible to recognize categories of wrong results and remove them with a subquery included in a MINUS clause. But what if such inspection is not feasible, because for example of a huge number of results? And what if the separation is not so crispy, with good and bad cases highly mixed, or if the exceptions are too many to be manually coded? We obviously need a classifier. Classification in SPARQL/T is supported by an external tool, and accessed from within the query with a triple pattern of the form:

```
parametersURL  ML:algorithmName
     ?requestedPrecision
```

where:

- `algorithmName` is the name of a Machine Learning algorithm, known to the system
- `parametersURL` is the address of a file, generated with the tool, and available through HTTP protocol. It contains the parameters of the algorithm, like for example the structure and thresholds of a decision tree, or the weights of a Deep Neural Network.
- `requestedPrecision` is an input numeric variable, in the range [0,1], indicating, when appropriate, the degree of precision requested. It may not make sense for a specific algorithm, which is always free to ignore it, but when appropriate, it allows a certain degree of flexibility during query execution.

Such triple acts as filters, removing from the result set all the tuples classified as negatives by the algorithm. It can be placed anywhere inside a graph path but, in a way similar to the SPARQL FILTER clause, it is always evaluated after all the other triple patterns of the same graph path, i.e. when the result set is available. The algorithm is supposed to work primarily on a result set itself, considering maybe a few words around each extracted snippet, but it has anyway access to the entire document and to all of its annotations. Loading the algorithm implementation, i.e. the executable code, at run time from the WEB through the HTTP protocol would obviously rise serious security issues. Therefore, it must be somehow manually loaded in the SPARQL/T server, identifiable by name. The algorithm's parameters instead, typically an XML or JSON file, with the proper precautions do not represent a threat and can be loaded from the web. This mechanism[27] allows the safe and easy sharing of queries containing Machine Learning algorithms. The generation of the training set and the training of the algorithm instead, is an implementation detail that should be taken care of with a proper Graphical User Interface, and is not discussed here. Suffice to say that its modus operandi, together with the choice of the algorithms, should be carefully studied in order to minimize the user effort, i.e. the number of elements necessary for the training set. Ideas related to the task of Adaptive Learning should seriously be considered (see for example [Settles 2012] for a review).

6.4.8 Indexing

There is no doubt that indexing should be part of SPARQL/T Conceptual Model. However, the use of Word Embedding introduces the problem of employing indexes in searches that involve similarities. In the words of Lashkaria et al [Lashkaria et al. 2019]: "*While the process for building and querying keyword-based indices is quite well understood, the incorporation of semantic information within search indices is still an open challenge*." Which means that, at the moment, there is no well-established way of indexing Word Embeddings. But fortunately, there are already good algorithms to explore. Briefly, the problem of efficiently finding a Word Embedding vector *v* closed to a given (query) one v_q is a nearest neighbor problem, stated by Indyk & Motwani [Indyk & Motwani 1998] as follows:

[27] Which admittedly need better formalization

Definition: **Nearest Neighbor Problem**
given a set of n point $P=\{v_1, \ldots, v_n\}$ in some metric space X, preprocess P so that to efficiently answer queries which require finding the point in P closest to the query point $v_q \in X$.

Recently, Sugawara et al. [Sugawara et al. 2016] compared several algorithms that address the problem, which can be divided into three categories:

- Hash-based: uses a hash function such that the probability of collision of two elements are higher if, in a certain metric space, these two objects are close to each other. Locality-Sensitive Hashing (LSH) [Gionis et al. 1999] is probably the most known of this class of algorithms.
- Tree-based: recursively divide the search space into non-necessarily disjoint subspaces. The search is performed hierarchically.
- Graph-based: a neighborhood graph is built, where each node is connected to its nearest neighbors. The search can be performed in a best first mode, starting at an arbitrary node.

Sugawara et al. found that graph-based algorithms (NGT in particular, [Iwasaki 2015]) perform better than others. Very recently, Lashkaria et al [Lashkaria et al. 2019] explored the possibility of building inverted indexes based on the similarity of the vector representation of terms instead of on term occurrence in documents.
Given the time constraints and the lack of well-established methods for Word Embedding indexing, at the moment no true indexing capability has been implemented in SPARQL/T. However, as a temporary measure, a form of document filtering is possible: a triple pattern referring to a Lucene query can be used to retrieve the subset of document to be further analyzed by the rest of the SPARQL/T query. (See Chapter 8 – Architecture)

6.5 Core Conceptual Model

The exposition that follows resemble the one given in chapter 12, “Definition of SPARQL”, of the W3C Recommendation for SPARQL 1.0[28]. Definition are given side-by-side, highlighting the differences from the RDF and the Text cases. Here, only things that has a counterpart in the RDF case are considered. Other things strictly related to SPARQL/T, like Hybrid Queries and the Reasoning Interface, are exposed in the next chapters.

6.5.1 Definitions

Let:

- V be the set of all the Variables
- F be the set of the NLP Functions
- L be the set of the literals

[28] SPARQL Query Language for RDF W3C Recommendation, 15 January 2008, https://www.w3.org/TR/rdf-sparql-query/, fetched 6 Jan 2019

Definition: **Triple Pattern (for Text)**
A triple pattern is member of the set:

$$(V \times F \times L) \cup (V \times F \times V)$$

This means that, syntactically, there are only two possible kind of SPQRQL/T triple pattern: one with two variables and one with a variable and a literal, both with the NLP function in the central (predicate) position. The NLP function is specified using a prefix (recognized by the system) that indicates the function group, followed by a subfunction code. Constant arguments of the NLP functions are placed in the literal (see Figure 23**Errore. L'origine riferimento non è stata trovata.**).

```
?Variable  Function:SubFunction 'Literal'
?Variable1 Function:SubFunction ?Variable2
```

Figure 23: the two kinds of SPARQL/T triple patterns, with one and two variables, both with the NLP function in the central (predicate) position

The two-variable case must be further split into two different ones: patterns that extracts two variables and patterns that require an input (see Chapter 6.4.2 about Input Objects).
In SPARQL/T the Basic Graph Pattern BGP has the same definition it has in SPARQL for RDF, but a slightly different semantic:

Definition: **Basic Graph Pattern (both for RDF and Text)**
A Basic Graph Pattern is a set of Triple Patterns.

The evaluation of a BGP consists in the evaluation of each of its triple patterns followed by a proper join of the results. In SPARQL/T this join operation must take into account a set of measures and constraints, like the distance of the snippets in the document and the fact that some triple patterns may share some variables. Intuitively, from the evaluation of a BGP we expect a sort of "logical AND" of the results of each single triple pattern. In the RDF case, given a graph, this idea of an "AND" is achieved in this way: a tuple is added to the result set if and only if all the triple patterns of the BGP match some part of the graph. Similarly, in the text case, given a unit of text, we want a tuple to be generated from that unit if and only if all the triple patterns in the BGP find some match (and thus extract something) inside it, with a reasonable degree of confidence. In SPARQL/T model such unit of text has been defined to be the sentence (and not for example the entire document or a sliding window of fixed size), i.e. :

Rule: **BGP Search inside a document**
a tuple is generated from a BGP search if and only if all its triple patterns extract something **inside the same sentence**

The reason for this choice is apparent when we consider languages poor of linguistic resources, for which parsers may not be available. As a matter of fact, when concepts are expressed inside the same sentence it is normally a simple but quite good indicator of their relatedness (think for example the three concepts "*to buy*", "*expensive*" and "*car*" in the sentence "I'm going to *buy an*

expensive car"). Therefore, in absence of better options, the BGP search rule allows a crude but effective semantic analysis of the sentence. Obviously, for a more precise analysis, syntactic or semantic parsers become necessary. However, they also all normally work at sentence level, suggesting again the sentence as the best level of granularity for a BGP. (Of course, a method is also provided in SPARQL/T to deal with searches that involves multiple sentences. See later the section about Group Graph Patterns).

But what is the proper way to combine the results of each NLP triple patterns to form the result of the full BGP? Because we need to merge sets of solution, following also the SPARQL for RDF model, we appeal to a Relational Algebra.
A relational algebra is a language used in the field of relational databases that allows to formally specify the operations of a query language. ([Date 2003], [Elmasri 2010]) It is an intermediate language for the expression and analysis of queries, whose expressions can easily be translated into the code that will actually perform the task. Also, and perhaps most importantly, it is employed in the task of optimizations. Cyganiak [Cyganiak 2005] introduced the idea of employing a relational algebra for the SPARQL query language, defining a relational model over RDF terms. The W3C SPARQL Recommendation [29] however adopts symbols and definitions different from those in the Cyganiak's work. Here, in the attempt to define a relational model over NLP functions, to simplify comparisons with SPARQL, the W3C Recommendation is mirrored as much as possible, with the major difference being the need to consider uncertainty.
First of all, the W3C definition of Solution Mapping for the text case must be slightly changed. In SPARQL for RDF a Solution Mapping μ^{RDF} is a mapping from a set of variables to a set of RDF terms.

Definition: **Solution Mapping (for RDF graphs)**
A Solution Mapping μ^{RDF} is a partial function $\mu^{RDF}: V \rightarrow RDF\_T$
The domain of μ^{RDF}, $dom(\mu^{RDF})$, is the subset of V where μ^{RDF} is defined.

Where the RDF Terms RDF_T are defined as follows:

Definition: **RDF Term**
Let I be the set of all IRIs.
Let RDF_L be the set of all RDF Literals
Let RDF_B be the set of all blank nodes in RDF graphs
The set of RDF Terms, RDF_T, is I union RDF_L union RDF_B.

In SPARQL/T the mapping must be made from a set of variables V to a set of *TextReference* objects (defined above). Also, to deal with uncertainty, we introduce a real number *w* in the range [0,1] that indicates a degree of confidence in the mapping:

[29] https://www.w3.org/TR/rdf-sparql-query/

Definition: **Weighted Solution Mapping (for Text)**
A Weighted Solution Mapping λ is a couple (μ^{TXT}, w) where μ^{TXT} is a partial function $\mu^{TXT}: V \rightarrow TextReference$, and *w* is a real number in the range [0,1] indicating the degree of confidence in the mapping.
The domain of μ^{TXT}, $dom(\mu^{TXT})$, is the subset of V where μ^{TXT} is defined.

When a mapping contains just a single variable, its degree of confidence *w* is simply the degree of confidence in the extraction performed by the triple pattern, which is stored in the *TextReference* object. For mappings with multiple variables, the degree of confidence *w* is calculated when mappings are merged, during the Join or Union operations (see later), and possibly altered by some other operations (like Minus, see also later on).
We now need to adapt the notion of compatible mappings defined in the W3C recommendation, taking into consideration the uncertainty of NLP annotations. Basically, the idea of compatible mapping is that two mappings are compatible when every variable that belongs to both is bound to the same object. When dealing with RDF graphs, this simply means that the variable is bound to the same URI, and this is expressed from the following definition, adapted[30] from the W3C recommendation:

Definition: **Compatible Mappings (for RDF graphs)**
Two solution mappings μ_1^{RDF} and μ_2^{RDF} are compatible if, for every variable v in $dom(\mu_1^{RDF})$ and in $dom(\mu_2^{RDF})$, $\mu_1^{RDF} = \mu_2^{RDF}$. To indicate such compatibility, we write $compatib_{RDF}(\mu_1^{RDF}, \mu_2^{RDF})$

If μ_1^{RDF} and μ_2^{RDF} are compatible then their union is also a mapping, and we can define the following function:

$$merge_{RDF}(\mu_1^{RDF}, \mu_2^{RDF}) = \mu_1^{RDF} \cup \mu_2^{RDF}$$

It may seem natural now, translating this concept for the text case, to simply require that the variable is bound to the exact same piece of text. However, when joining elements extracted by different triple patterns, i.e. by different algorithms, it is quite naïve to expect exact matches. Moreover, what we want to be the same here is not the piece of text itself, but the concept it represents. A wiser and more robust approach is therefore to employ a measure of overlapping between snippets of text. Another thing to consider is that the same entity of the world may be represented in the text by different snippets (for example, first by its name and later by a pronoun). Actually, this linguistic phenomenon, called Anaphora or Co-reference[31], seems like a natural counterpart of the Join operation in Natural Language and deserves a special treatment inside SPARQL/T.
Let's then $overlap(t_1, t_2)$, with $t_1, t_2 \in TextReference$, be a number in the range [0,1] that measures the degree of overlap between the two snippets of text t_1 and t_2 . The details of this measure are left as implementation dependent. It can be related to chars or to tokens, and possibly weight words differently accordingly to their Part Of Speech tag and to a TF-IDF measure.

[30] μ has been changed into μ^{RDF} for clarity

[31] They are similar concepts, but not exactly the same. See for example [Poesio et al. 2016] for a discussion

But for the sake of clarity of the examples that follows, let's assume from now on that $overlap(t_1, t_2)$ is simply defined as the number of common tokens divided by the number of tokens in their union.
Let's also $coref(t_1, t_2)$, with $t_1, t_2 \in TextReference$, be a number in the range [0,1] that measures the degree of confidence that the snippets of text t_1 and t_2 refer to the same entity of the world, according to the annotations of some Anaphora Resolution tool.
Let's then combine them, to define the compatibility coefficient between two snippets of text:

$$compat_{TXT}(t_1, t_2) = overlap(t_1, t_2) \otimes coref(t_1, t_2)$$

We also need to change, with respect to the RDF counterpart, the definition of mapping, associating to each mapping μ^{TXT} a weight w

Definition: **Weighted Mapping**
A weighted mapping λ is a couple (μ^{TXT}, w), where μ^{TXT} is a mapping and w is a real number in the range [0,1] indicating its degree of truth
Let also define the following two functions, that extract the mapping and the weight from a weighted mapping $\lambda = (\mu^{TXT}, w)$
$\mathrm{M}(\lambda) = \mu^{TXT}$
$W(\lambda) = w$

We want now to state that two weighted mappings λ_1 and λ_2 are compatible if every common variable either overlap or refer to the same entity to some extent:

Definition: **Compatible Mappings (for Text)**
Two weighted solution mappings $\lambda_1 = (\mu_1^{TXT}, w_1)$ and $\lambda_2 = (\mu_2^{TXT}, w_2)$ are compatible if, for every variable v in $dom(\mu_1^{TXT})$ and in $dom(\mu_2^{TXT})$, $compat_{TXT}\left(\mu_1^{TXT}(v), \mu_2^{TXT}(v)\right) > 0$

Continuing along the W3C recommendation, if $\lambda_1 = (\mu_1^{TXT}, w_1)$ and $\lambda_2 = (\mu_2^{TXT}, w_2)$ are compatible then $\mu_1^{TXT} \cup \mu_2^{TXT}$ is also a mapping, and we can define the function *merge* as the union of two compatible mappings:

$$\lambda_m = merge_{TXT}(\lambda_1, \lambda_2) = (\mu_m^{TXT}, w_m) = (\mu_1^{TXT} \cup \mu_2^{TXT}, W_m(\lambda_1, \lambda_2))$$

Here W_m is a function that returns the degree of confidence in the resulting mapping. The details of W_m should also prudently be left as an implementation specific, although some ideas related to Fuzzy Logic are reported in Chapter 6.3. It is expected to take into consideration:

- The degree of confidence in the two source mappings w_1 and w_2
- The degree of compatibility (overlapping and coreference) between the two mappings. Notice that the function $compat_{TXT}(t_1,t_2)$ just defined express such degree for a couple of text snippet, and has been applied to (common) variables. A global degree of compatibility for the whole mapping needs then to be defined. A good candidate is again the t-norm function $\otimes$.

The following definitions is exactly the same given for SPARQL for RDF:

Definition: **Solution Sequence (for RDF and Text)**
A solution sequence is a list of solutions, possibly unordered.

Notice that, as also mentioned in the W3C recommendation, a solution sequence Ω is not a set: it is defined as a list, i.e. a multiset, an unordered collection of elements in which each element may appear more than once.

6.5.2 SPARQL/T Algebra

In this section the operators for the evaluation of a SPARQL/T query are defined. At the moment they are just a limited subset of the one defined for the SPARQL for RDF algebra: only the Join and the Union ones, plus the Minus, that is an addition of SPARQL 1.1.
The Join and the Union operations defined below are the ones that allow to merge solution sequences in the RDF case:

Definition: **Join (for RDF)**
Let Ω_1 and Ω_2 be two solution sequences. We define:

$$Join_{RDF}(\Omega_1, \Omega_2) = \{merge_{RDF}(\mu_1^{RDF}, \mu_2^{RDF}) | \mu_1^{RDF} \in \Omega_1 \wedge \mu_2^{RDF} \in \Omega_2 \wedge compat_{RDF}(\mu_1^{RDF}, \mu_2^{RDF})\}$$

Definition: **Union (for RDF)**
Let Ω_1 and Ω_2 be two solution sequences. We define:

$$Union_{RDF}(\Omega_1, \Omega_2) = \{\mu^{RDF} | \mu^{RDF} \in \Omega_1 \vee \mu^{RDF} \in \Omega_2\}$$

Definition: **Join (for Text)**
Let Ω_1 and Ω_2 be two weighted solution sequences. We define:

$$Join(\Omega_1, \Omega_2) = \{merge_{TXT}(\lambda_1, \lambda_2) | \mathrm{M}(\lambda_1) \in \Omega_1 \wedge \mathrm{M}(\lambda_2) \in \Omega_2 \wedge compat(\mathrm{M}(\lambda_1), \mathrm{M}(\lambda_2)) > 0\}$$

Definition: **Union (for Text)**
Let Ω_1 and Ω_2 be two weighted solution sequences. We define:

$$Union_{TXT}(\Omega_1, \Omega_2) = \{\mu^{TXT} | \mu^{TXT} \in \Omega_1 \vee \mu^{TXT} \in \Omega_2\}$$

Informally, the Join operation requires that, in a specific unit of text, both extraction patterns find some match (corresponding to a logical AND), whilst the Union operation requires just one of the two (corresponding to logical OR).
Notice that the definition of Compatible Mapping (for Text) includes the case where the two mappings have no variables in common. In this case the Join function simply becomes a Cross Join, i.e. the Cartesian Product of the two sets Ω_1 and Ω_2, defined as the set of couples $((\omega_1, \omega_2)|\omega_1 \in \Omega_1 \wedge \omega_2 \in \Omega_2)$.
For the Cross Join however, the function W_m that calculates the degree of confidence in the resulting mapping cannot rely anymore in measures of overlapping or coreference between objects mapped by a common variable (as there is no common variable). A reasonable replacement can be some measure of proximity between the snippets of text of the couples, again left as implementation dependent.
A third operation, somewhat related to Join and Union, is the Minus one, introduced in SPARQL 1.1 and that allows to remove from a set of solution mapping Ω_1 all the elements of a second set Ω_2

Definition: **Minus (for RDF)**
Let Ω_1 and Ω_2 be two solution sequences. We define:

$$Minus(\Omega_1, \Omega_2) = \{\mu_1^{RDF} | \mu_1^{RDF} \in \Omega_1 \wedge \forall \mu_2^{RDF} \in \Omega_2 \neg compat_{RDF}(\mu_1^{RDF}, \mu_2^{RDF})\}$$

In words, the result of the Minus operation, in the RDF case, is the set of all the mappings belonging to the first set that are not compatible with any of the mappings in the second.
In the case of text snippets, it is again desirable to avoid crispy tuple removal. A Fuzzy interpretation of the Minus operation can be a sort of negative reranking, that lowers the weights of the results in the first set that also belongs to the second. More specifically, for each weighted solution mapping $\lambda_1 = (\mu_1^{TXT}, w_1) \in \Omega_1$, let $\lambda_2' = (\mu_2^{TXT}, w_2) \in \Omega_2$ be the mapping of Ω_2 with the higher compatibility with λ_1, i.e.:

$$\lambda_2' = \underset{\lambda_2 \in \Omega_2}{\operatorname{argmax}} W_M(\lambda_1, \lambda_2)$$

Let $w_{1,2} = W_M(\lambda_1, \lambda_2')$. If $w_{1,2} > 0$ we decrease the weight w_1 of λ_1 by a quantity that depends on two things: the weight w_2 of λ_2' and the degree of compatibility $w_{1,2}$ between the two. Using the Fuzzy Logic negator operator $\ominus$:

$$w_1 = w_1 \otimes \ominus (w_2 \otimes w_{1,2})$$

The idea is that the higher is the weight of the best match λ_2', and the lower becomes the weight of λ_1, but if λ_1 and λ_2' are not so similar, the negative effect of λ_2' is decreased.

Definition: **Minus (for Text)**
Let Ω_1 and Ω_2 be two solution sequences. We define:

$$Minus(\Omega_1, \Omega_2)$$
$$= \left\{ (\mathrm{M}(\lambda_1), W(\lambda_1) \otimes \ominus (W(\lambda_2) \otimes W_M(\lambda_1, \lambda_2))) \middle| \lambda_1 \in \Omega_1, \lambda_2 = \underset{\lambda_2 \in \Omega_2}{\operatorname{argmax}} W_M(\lambda_1, \lambda_2), W_M(\lambda_1, \lambda \right.$$

To deal with concepts possibly expressed in multiple sentences, it then become natural to adapt the SPARQL for RDF notion of Group Graph Patterns, which is simply a BGP delimited with braces: {}.
Group Graph Pattern can be used to fragment the query pattern (the outer-most graph pattern) into a set of BGP pieces, each of which can then be evaluated against each sentence of the text. And then again, the result of the Group Graph Pattern should be the "logical AND" of its component. Of course, this principle can be applied recursively, allowing to query concepts of any complexity.

6.5.3 Extraction Scopes

Intuitively, BGP are intended to represent basic concepts, things that can typically be expressed within a sentence and that, in Frame Semantic terms, roughly resembles the idea of a frame.
During query evaluation, each BGP is therefore evaluated one sentence at the time, with the effect that all the triple patterns in the BGP must be satisfied inside a single sentence. This approach has been taken with languages poor of resources in mind, where we may be forced to work with, say, just a Word Embedding and a POS tagger. In that case, a BGP that asks, for example, for a verb like "to buy" and a word like "car", is expected to find, with a reasonable level of precision, sentences where a car (or similar) is the object of some purchase. Of course, this is more likely to be true when the two words are part of the same sentence, instead of when they are far apart in the document. For sure, good results requires more sophisticated approaches, like the employment of a dependency or constituency parser (provided they are available for the specific language). However, also that tools work at sentence level, suggesting again the sentence as the most appropriate level of granularity for a BGP search.
For clarity, let's call this idea of granularity of search the "extraction scope" of the pattern, defined as follows:

Definition: **Extraction Scope** The Extraction Scope of a SPARQL graph pattern is the unit of text inside which all the constraints of the graph must be satisfied for a tuple of the result to be produced.

We can then say that **the Extraction Scope of a BGP is the sentence**, meaning that each tuple of a solution of a BGP will pertain to a single sentence.
Of course, there are situation when we need to consider different scopes. A discourse describes a complex concept and normally spans several sentences. Sometimes instead, we may need to drill down inside a big chunk of already extracted text to find its components (for example, to extract the Named Entity from a chunk of text expressing a certain sentiment). The ability of looking for a concept that may span different sentences comes with no effort: for what has just been said, everything that is extracted from different BGPs are allowed to pertain to different sentences. Syntactically, BGP can be created by enclosing them in curly braces '{' and '}'. A portion of a SPARQL query string delimited with braces {} that contains zero or more BGPs is called, in the W3C SPARQL 1.0 recommendation, a Group Graph Pattern. Therefore, we can say that **the Extraction Scope of a Group Graph Pattern is the document**. Figure 24 shows an example of Group Graph Pattern, taken from that recommendation.

```
{    ?x foaf:name ?name .
     {}
     ?x foaf:mbox ?mbox .
}
```

Figure 24: Example of Group Graph Pattern reported in the W3C SPARQL 1.0 recommendation. It is a group of three elements: a basic graph pattern of one triple pattern, an empty group, and another basic graph pattern of one triple pattern.

On the other hand, to limit the extraction scope to a portion of already extracted text we need to employ a special keyword. In SPARQL for RDF, the GRAPH keyword directs the search inside a named graph. Figure 25 shows an example taken from the W3C recommendation. The idea there is that a variable `?g` contains an URI that specifies the graph on which the subquery that follows (enclosed in braces) has to be performed.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?who ?g ?mbox
FROM <http://example.org/dft.ttl>
FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/bob>
WHERE
{    ?g dc:publisher ?who .
     GRAPH ?g { ?x foaf:mbox ?mbox }
}
```

Figure 25: example of use of GRAPH keyword, taken from thw W3C SPARQL 1.0 reccomendation. The first triple pattern binds the variable ?g with a set of URI taken from the default graph. The second one performs a search inside the named graph specified by ?g.

The idea in SPARQL/T is not so different: a variable `?g` contains a reference to a piece of text inside which the extraction has to be performed. Following our metaphor of Virtual Graph, such piece of text can be seen as a "subgraph" of the document. We then can say that **the Extraction Scope of a subquery contained inside a `GRAPH ?g {}` statement is the snippet of text to which `?g` refers**. Table 9 summarizes the Extraction Scope possibilities.

Unit of the Query	Extraction Scope
BGP	Each sentence separately
Group Graph Pattern	The document
`GRAPH ?g {}`	The snippet of text `g`

Table 9: Extraction Scope of different units of a query

6.5.4 Query Evaluation

During query evaluation each triple pattern is applied to the document, one sentence at a time, and extracts from it a relation containing one or two columns (depending on the number of variables) of objects of type *TextReference*, plus an extra column containing the degree of truth of the extraction.

Then, the relations pertaining to each BGP are formed by joining those of the triple patterns that it contains, taking care in this particular BGP-internal join operation of not creating tuples pertaining to different sentences. A BGP's relation is therefore a table with K_i+1 columns of type *TextReference*, where K_i is the number of distinct variables of the BGP i, plus an extra column contains the fuzzy weight of the entire tuple, calculated by the join operation.
Different BGPs are then joined together in a similar fashion, following the structure of the algebra tree, but this time without the constraint on sentence belonging.
Because of memory constraints and of the potentially huge number of tuples that certain queries may generate, at each step of the process the relations are ranked, and only the best N results are passed to the next operation. Finally, by default, the relation of the root of the tree is also ranked and only the best solution is returned for the document.

6.6 SPARQL/T peculiar concepts

6.6.1 Hybrid Queries

A SPARQL/T query that involves access to a triple store is called here a Hybrid Query, as part of it involves the text and part an RDF graph. SPARQL syntax provides a method to direct a portion of the query to an external server (SERVICE …), and that syntax can in principle be used also for Hybrid Queries construction. However, such syntactic overhead is not necessary, since the compiler can easily distinguish an NLP triple pattern (which we will call internal) from one intended for the triple store (external) by simply looking at the prefixes of the predicate part of the triple. By doing so, a BGP can be divided into homogeneous chunks called BGP (homogeneous) fragments, that are separately evaluated and joined. More formally:

Definition: **Internal Triple Pattern**
An Internal Triple Pattern of a SPARQL/T query is a triple pattern that evaluates against a document

Definition: **External Triple Pattern**
An External Triple Pattern of a SPARQL/T query is a triple pattern that evaluates against an RDF graph

Definition: **Hybrid Query**
A SPARQL/T Hybrid Query is a query that contains both internal and external triple patterns

Definition: **BGP Homogeneous Fragment**
A BGP Homogeneous Fragment (or BGP Fragment for simplicity) is a group of contiguous triple patterns of a BGP that are homogeneous, i.e. that are either all internal or all external

Because External Triple Patterns returns traditional RDF objects, like URI and literals, and Internal ones return TextReference objects, we need a way to compare these two heterogeneous results

in order to make the requested joins. To this purpose, a new class of object is introduced, that contains three fields: an URI, a TextReference and a string, some of which may be null. In brief, the idea is that a comparison between two objects of that class is performed by first comparing the three fields separately (when present), and then merging the results. Ideally:

- URIs should be compared either crispy (exact match) or by using a reasoner that deals with entailments (a fuzzy reasoner may also consider degrees of truth)
- TextReferences continue to be compared based on they overlapping on the document
- String similarity can be calculated using Word Embeddings, Edit Distance, or again with a measure of overlapping

Provided that the non-trivial task of rendering these three measures comparable has been reasonably tackled, a t-conorm (for example, the highest of the three values) can be taken to represent the global similarity of the two objects.

6.6.2 Reasoning Interface (RI)

SPARQL/T aim to give to external OWL reasoners the ability to reason directly on the text, using a document in place of an RDF graph. Ideally, and similarly to the RDF case, the reasoner should be able to extract from the document facts that are not explicitly stated, but that can be entailed from it using the axioms of a specified ontology. To illustrate the idea, let's first see a simple example of traditional OWL reasoning, adapted from [DuCharme 2013]. Let's say an RDF graph contains the OWL triples in Figure 26:

```
1 p:tony           p:playInstrument    p:guitar.
2 p:playInstrument rdf:type            rdf:property;
3                  rdfs:domain         p:Musician;
4 p:mario          rdf:type            p:Musician.
```

Figure 26: an OWL graph. Triple 3 can be read as "anyone who plays something is a musician". Prefixes declarations have been omitted for brevity.

Let q be a query containing the following triple pattern in its WHERE clause:

```
?x   rdf:type              p:Musician.
```

A plain query engine, without a reasoner, will only match the triple 4, binding `x` with `p:mario`. An OWL reasoner on the other hand, from triple 3 knows that the domain of the property `p:playInstrument` are objects of the class `p:Musician`, i.e. that any triple in the form:

```
s    p:playInstrument        o
```

means, among other things, that `s` belongs to the class `p:Musician` ("anyone who plays an instrument is a musician"). Therefore, because of triple 1, the reasoner can infer from the RDF graph that also `p:tony` is a valid bind for `x`. Basically, we want a reasoner to be able to do a similar inference with text documents, employing the exact same ontology and extracting facts at run time directly from the text. An example of document expressing the same facts of the graph in Figure 26 is reported in Figure 27.

Mario	*just*	*plays*	*guitar*	*from*	*time*	*to*	*time*	*.*	*Tony*	*instead*	*is*	*a*	*great*	*musician*
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Figure 27: an example of (tokenized) document containing, expressed in natural language, some concepts of the graph in Figure 26.

The UML Sequence Diagram in Figure 28 gives the basic idea of how the things should work. In this example the sequence of action is started by a SPARQT/T query, that asks the reasoner how to find instances of the class `Musician` (Step 1: Reasoner Request 1). However, a similar sequence can be started by any other actor or by the Reasoner itself. The reasoner then queries his own RDF/OWL graph, here represented by the "Triple Store" lifeline, to get information about the class `Musician` (Step 2: OWL Request 1). From the Triple Store come the information that `Musician` is the domain of a predicate `play`, that has the class `Instrument` as range. Now somehow (see later on), the reasoner knows that the SPARQL/T endpoint can be the source of triples involving the predicate `play`, and therefore sends it a query about that (Step 4: Text Query 1). There are many options here, but the simplest one is to just send the name of a stored query to run. The query succeeds, finding inside the document the sentence "*Tony is playing the guitar*", and returns such information in the form of an RDF graph, whose schema should be well defined and known to both (see also later about this). To play safe, in step 6 the reasoner further checks the SPARQL/T results against its own Triple Store, verifying that the object of `play` actually belongs to the class `Instrument`, and finally closes the request with a positive answer. In principle, by implementing a reasoner inside SPARQL/T, or by adapting an existing one, the entire process just described could be run inside a single SPARQL/T query. However, implementing an interface decouples the extraction task from the reasoning one, with at least the following advantages:

- Clarity: queries are smaller and simpler, and thus also easier to maintain
- Reusability: a simple query with limited purpose can be reemployed in different contexts
- Speed: the reasoner may be able to selectively call only the queries strictly needed to reach the conclusion
- Abstraction: once libraries of extraction queries and related ontologies will be available, the user will be able to work at higher conceptual level, dealing with ontologies instead of with cumbersome NLP details.

Moreover, there is the hope that a simple and well-designed reasoning interface may stimulate reasoner's authors to try their product in the task of reasoning with text.

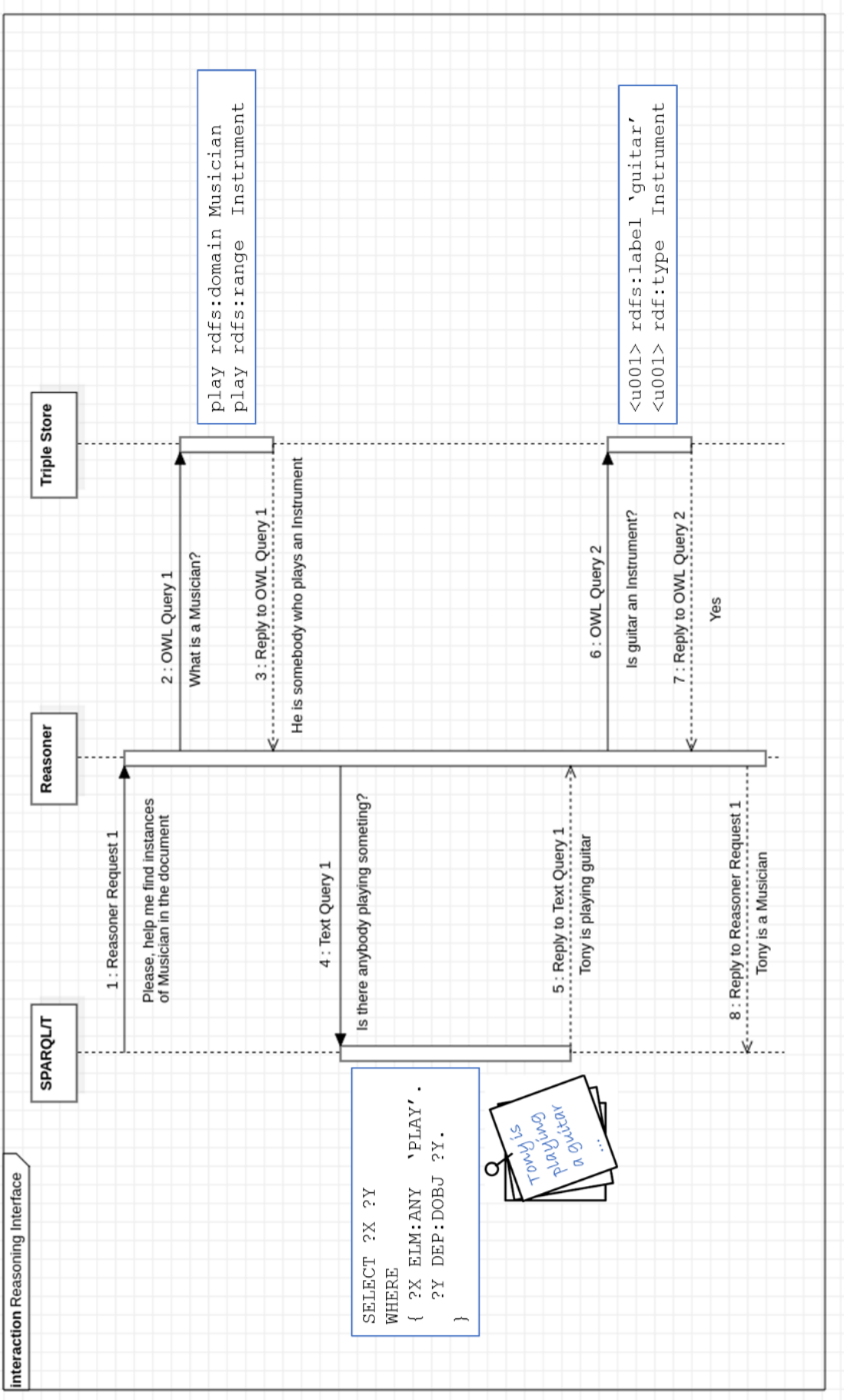


Figure 28: UML diagram describing the interaction between a SPARQL/T engine and an external reasoner. Ideally, the reasoner should be able to work on natural language documents as they were RDF graphs stored in a triple store, with SPARQL/T acting as an interface between the two.

In defining that interface, the primary goal must be to make it simple to adopt by the majority of existing reasoners. However, it should be noticed that probably some categories of reasoners are more suited than others for the task. The process just described resembles more a Backward Chaining reasoning one (see for example [Russel & Norvig 2010] for the general case definition, and [Curé & Blin 2014] for the RDF case). Backward Chaining works backward from the goal (in this case, of finding instances of `Musicians` in the document), using rules to find facts that support the proof. In this case, when appropriate, those facts are asked back to the SPARQL/T endpoint. On the other hand, another big family of reasoners, the Forward Chaining one, start instead from the facts in the knowledge base and apply rules to generate new facts until no further inference can be made. This approach, in our case, requires to run all possible useful queries before the reasoning process can start, which can lead to a huge waste of time when the size of such set of queries is large (unless of course, a good machine learning algorithm could be employed to select just the useful ones). In other words, SPARQL/T query interface seems more suitable for Backward Chaining reasoning algorithms (or hybrid ones, like implemented for example in Apache Jena[32])

With the primary goal of minimizing the effort required to adapt existing reasoners, a major issue to consider is to define the most suitable protocol for data exchange[33]. Specifically, from the reasoner point of view:

1. What is the easiest way to formulate a request to the SPARQL/T endpoint?
2. What is the best format for the result?

Moreover, what is the best way to let the reasoner know that a certain triple pattern should be directed to the SPARQL/T endpoint instead of matched inside its own RDF Triple Store?

SPARQL/T implements a standard SPARQL endpoint interface, allowing complete queries that may return either CSV tables (for SELECT queries) or RDF graphs (for the CONSTRUCT ones).

However, forcing the reasoner to create or store complex queries on its side does not help integration, and also violates the principle of decoupling the extraction and reasoning tasks. A better, simplified approach is probably to allow the reasoner to formulate queries of just one triple pattern at a time, and have in return the same kind of RDF triples that it could expect from a Triple Store. If appropriate, upon receiving the results the reasoner can in this way store them in the triple store itself, and proceed with the reasoning activity just as if they have always been there. However, from the SPARQL/T point of view, this does not need to be just a trivial single pattern query, but it is instead an evocation schema for a full fledge CONSTRUCT query locally stored in the SPARQL/T endpoint itself. So, to continue with our example, two possible queries can be:

Q1	`?x`	`rdf:type`	`p:Musician`
Q2	`?x`	`p:playInstrument`	`?y`

Figure 29: Two possible single-triple pattern queries

and the results may look something like:

[32] https://jena.apache.org/documentation/inference/ , fetched 18 February 2019.

[33] Assuming HTTP as the low level protocol

R1	<http://.../001>	rdf:type	p:Musician
	<http://.../001>	rdfs:label	"Mario"
R2	<http://.../002>	p:playInstrument	
	<http://.../003>		
	<http://.../002>	rdfs:label	"Tony"
	<http://.../003>	rdfs:label	"Guitar"

Figure 30: Two possible results of the queries in Figure 29

The problem with the results in this format is that we are unable to assign a degree of truth to the triples, losing in this way possibly important information. Moreover, with the purposes of efficiency and reusability, a SPARQL/T query may want to extract an entire frame from the document, complete of all its possible Frame Elements, inside a single call. Therefore, we need to return a graph in a format suitable to represent both frames and truth values, but that still contains the triples in the form expected from a Triple Store (let's call the latter the "main triples"). A similar problem is faced in the Linked Open Data community, where n-ary relations are represented with RDF triples in different ways inside different knowledge resources, complicating their integration and ultimately reducing the queries recall. Rouces et al. [Rouces et al. 2017] expose the problem, making a survey of the most relevant approaches, and suggest their own solution. Figure 31 illustrates the approach suggested here, with an example of how the result of query Q1 of Figure 29 may appear. Like the one suggested by Rouces et al., it is a two-layered structure, with a neo-Davidsonian part representing the frame and its element, and the main triple represented with a direct binary predicate. As suggested by Bobillo & Straccia [Bobillo & Straccia 2011], the truth values are reported in form of annotations, which are meant to assign human readable labels and text definitions to classes and properties, and thus to be ignored by inference engines. Moreover, while a fuzzy reasoner can take advantage of the fuzzy information, a crispy one can simply proceed as they do not exist. Notice that `p:fuzzyLabel` is not a predefined OWL annotation[34], and must be declared as one with a triple like:

```
p:fuzzyLabel   rdf:type  owl:AnnotationProperty.
```

[34] OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation 11 December 2012, https://www.w3.org/TR/owl2-syntax/ , fetched 27 January 2019

Stored Query	?X playInstrument ?Y

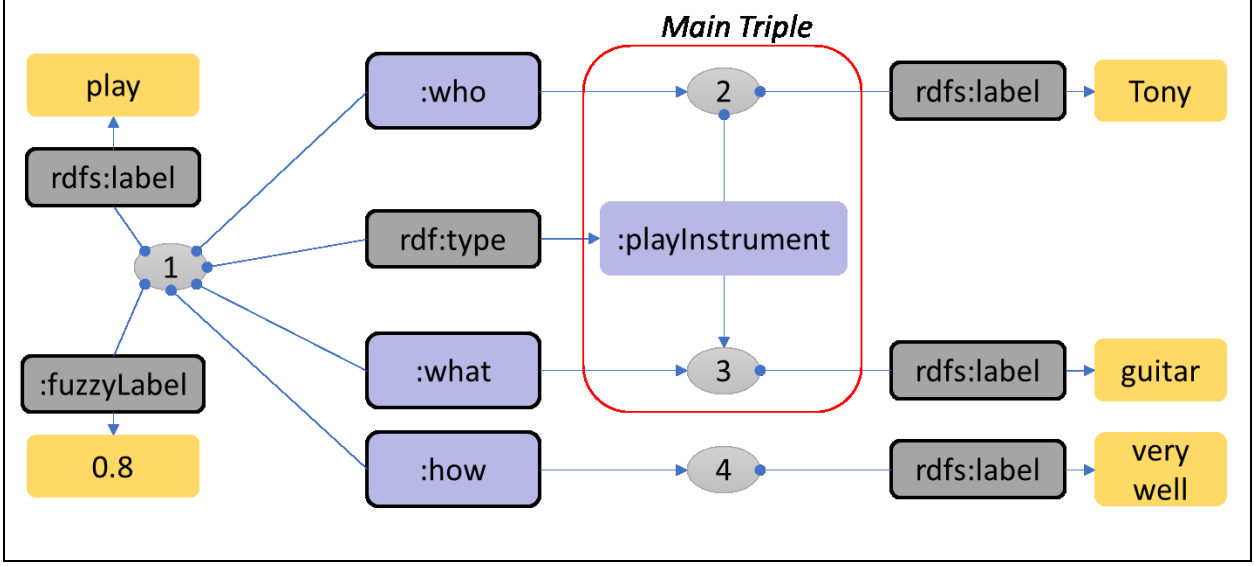


Figure 31: an example of RDF graph returned by a call to a SPARQL/T stored query. The call is made passing to the SPARQL/T endpoint a single triple pattern, in this case containing the predicate playInstrument and the two variables X and Y. The Main Triple of the result (surrounded by the red rectangle) is in the format that should be expected from a Triple Store, when queried with the same triple pattern. The rest of the graph contains other information extracted by the query (in this case the entire frame) and a fuzzyLabel indicating the degree of truth of the extraction. The snippets of text extracted from the document are reported as labels.

Clearly, this approach limits the expressivity of the communication to a single triple, and how much this may be a problem is at the moment unclear.
Finally, a safe approach to allow the reasoner to distinguish the NLP triple patterns to send to SPARQL/T endpoint must be decided. Probably, like in the case of the fuzzyLabels suggested by Bobillo & Straccia, the best way is again to use RDF annotations, as they should not interfere with any reasoning process. However, from the computational point of view, there may be better methods to explore.

```
p:Musician                p:nlpFunction          "SPARQLT-
endpoint1" .
p:playInstrument     p:nlpFunction        "SPARQLT-endpoint2" .
```

```
p:nlpFunction        rdf:type
owl:AnnotationProperty.
```

Chapter 7 – Functions List

7.1 Search for Words, Lemmas and Embeddings

This set of functions considers words, n-grams and lemmas of the document, allowing to search inside it those listed in the triple pattern. The comparison can be either crispy (an exact match), or employ a similarity measure based on a Word Embedding. The distinction between these two cases is made by the prefix of the predicate function (second position in the triple), whilst what follows that prefix (the subfunction code) specify how to use with the list of strings to search. This search list is given in the third (object) position of the triple pattern, and can be specified in two ways: either with a string literal (a space separated list of tokens) or with an input variable. As described in section 6.4.2, the concept of input variable is specific of SPARQL/T semantic, it does not exist in SPARQL for RDF. It simply means that the variable must already be bounded to a result before this triple pattern is encountered in the query string. Otherwise, quite obviously, the engine doesn't know what to search. The evaluation of the triple pattern, i.e. the search inside the document, binds the subject variable (specified in the first position in the triple) with a set of results. In case of Word Embedding search, to each record of the result is assigned a score according to the similarity value. In case of crispy tests, the score is equal to 1.

Given the list of words (or lemmas) to search, the subfunction code specify how to interpret such list. In case of crispy matches, the **ANY** subfunction compares the words of the document with every word of the search list, and returns any possible exact match. The **SEQ** subfunctions considers the search list as a sentence and searches inside the document all of its words in the exact same order. In this case wildcards are allowed: starting the search list with a set of *n* asterisks allows the engine to skip up to *n* position in the document in order to achieve the match (words can be skipped anywhere in between the first and the last token of the match). The **PER** subfunction is similar to the SEQ one, but allows permutations. It is obviously computationally more expensive, but may add flexibility when necessary. In case of Word Embeddings, we have a very similar set of possible subfunctions. The Word Embedding version of the ANY subfunction still compares the words of the document with every word of the search list, keeping the matches with the highest score. The **AVG** subfunction calculates an average vector of the Word Embedding of the search list, and compare it against the vectors of the document. The idea here (quite experimental) is to allow to search for a sort of average-concept, the "center" of the concepts listed in the search list. The **SUM** subfunction on the other hand, is the analogous of the crispy SEQ, interpreting the search list as a sentence. It considers a sliding window on the document of the same size of the search list and, at each step, compares the sum of the Word Embedding vectors of the words in the window with the sum of those in the search list. The result is a set of snippets of text ranked accordingly to the similarity to the searched sentence (search list). This is a basic but quite effective way to achieve sentence similarity employing just Word Embeddings. It is outperformed by recent methods like Universal Sentence Encoder, but unfortunately these are still not available for most languages.

All the above-mentioned function can work with words and with **lemmas**. A lemma is a root form of a word, like the infinite of a verb or the singular of a noun. It should not be confused with the word's stem, which is a string obtained by stripping from the words its morphological suffixes and prefixes. The stemming process is much easier and faster than the lemmatization one, often employed in Information Retrieval for these reasons. However, just stripping the suffix from a string may lead to represent with the same stem words with very different meaning. For example, all the words in the set {"*univers*al",

"*univers*ity", "*univers*e"} are likely to be stemmed into the string "*univers*"[35]. This problem is known as overstemming, and has a counterpart called understemming, that happens where different word inflection of the same lemma result in different stems. For this reason, SPARQL/T only consider lemmas and not stems. Another point to consider about lemmas is whether the kind of lemmatization process employed only considers the inflectional morphology of the word (tense of a verb, number of a noun, …) or if it also takes into account derivational morphology, i.e. transformations that usually involve a change in grammatical category (for example, creating the noun *worker* from the verb *work*, or the noun *loudness* from the adjective *loud*).
The advantage of employing lemmas (or stems) in a crispy match is obvious: we can represent with a single string the full set of possible morphological variations of a word, involving tense, number and so on. This is in practice more a necessity that a pure advantage, especially when dealing with verbs. With Word Embedding instead, at least in principle, employing lemmas should not give any advantage, as we expect that morphological variations should not substantially change the word's meaning (and thus its vector representation). Unfortunately, this is often not the case, and using the lemma instead of the word in Word Embedding comparisons appears to give much better results.

`?var`	`FUNCTION:subfunction`	`'searc-list'`
`?var`	`FUNCTION:subfunction`	`?inputVar`

Figure 32:the two variation of the word search triple pattern: with a constant search list and with an input variable.

FUNCTION PREFIX	**The subfunction applies to:**
`WRD`	The words in the text
`LEM`	The lemmas of the words in the text
`EMB`	The Word Embedding vectors of the words in the text
`EML`	The Word Embedding vectors of the lemmas of the words in the text

Figure 33: the possible prefixes of the word search function, indicating if the comparison is made between words or between lemmas, and if it is crispy (they are the same or not) or fuzzy, involving a Word Embedding.

Crispy Matches (WRD & LEM)	
`subfunction`	**Description**
ANY	Returns any word or lemma in the text that matches at least one of those specified in `search-list`.
SEQ (sequence)	Considers the `search-list` as a sentence, and returns matching snippets of text, i.e. sets of contiguous words of the text that matches all the words in `search-list`, exactly in the same order. A number n of wildcards (*) in the beginning of the list instruct the engine to skipping, if necessary, of a maximum of n words, in any position of the sequence.
PER (permutation)	Like SEQ, but does not impose the order of the words in the document to be the same of those in the `search-list`

Figure 34:Possible subfunctions of the crispy (exact match) functions

[35] Example reported by Wikipedia, attributed to the Porter stemmer [Porter 1980], https://en.wikipedia.org/wiki/Stemming , fetched 28 January 2019.

Word Embedding Matches (EMB & EML)	
subfunction	**Description**
ANY	Each word or lemma of the document is compared, using cosine similarity, with each words or lemmas in `search-list`, and the best match is taken.
SUM	The sum of the vectors of the words or lemmas in `search-list` is compared with the sum of the words (prefix `EMB`) or lemmas (prefix `ELM`) of a sliding window (of the same size) in the document.
AVG	The average of the vectors of the words (or lemmas) of `search-list` is compared with the vector of each word (or lemma) of the document

Figure 35:Possible subfunctions of the fuzzy (Word Embedding) functions

7.2 Regular Expressions

Regular expressions in SPARQL/T can be applied to words, lemmas and sentences and have the same syntax of the Java's ones.

`WRD:REX`	the regular expression is applied one token at a time
`LEM:REX`	the regular expression is applied one lemma at a time
`SEN:REX`	the regular expression is applied to the entire sentence

For example, the following triple pattern searches, sentence by sentence, a few possible names of telephone offers:

`?Offer`	`SEN:REX`	`"(senza ?limiti|(iper|turbo) ?giga| (ten|top) go|(tim) ?(young|special|base))|(entra in tim)" .`

7.3 Named Entities

A named entity is a real-world object that can be denoted with a proper name (or proper noun). According to Wikipedia: "*A proper noun is a noun directly associated with an entity and primarily used to refer to that entity, such as London, Jupiter, Sharon, or Microsoft, as distinguished from a common noun, which is a noun directly associated with a class of entities (city, planet, person, corporation)*"
As another example, "Theresa May" is a named entity whilst "Prime Minister" is not, as it does not refer to a specific real life one.
The task of identifying and classifying the named entities in a document is called Named Entity Recognition (NER). (Although sometimes in the literature NER indicates just the subprocess of finding the boundaries of the named entity, whilst the process of classifying it is called NEC and collectively they are called NERC).
The number of classes taken into consideration by the NER task may vary from the very few of what is often called the core set (Person, Organization, Location, and Date and Time)[36] to the more recent fine-grained ones, that may include classes like biologist, composer, or athlete [Ekbal et al 2010].
It should be mentioned that the NER task does not reduce to a simple string look-up in a table (technically, a gazetteer), for a number of issues. Among others (see also [Maynard et al. 2016]):

- names are ambiguous: "May" can be the name or surname of a person, a month of the year or a common noun ("you may go")
- many companies, diseases and laws are named after people

[36] developed for the shared NERC task at MUC-6, see [Grishman & Sundheim 1995]

- the same entity can be mentioned in different ways (John Smith, Mr. Smith, John, J. S. Smith, Smith)
- we need to consider acronyms (U.K. / United Kingdom) and aliases (IBM / Big Blue).

The syntax of the NER triple pattern is given in Figure 36. The result is a relation, binded to the variable `?var`, containing all the snippets of text of the sentence that had been annotated with the class `NE-class` (specified in the object of the triple).

`?var`	`NLP:NER`	`'NE-class'`

Figure 36:syntax of the NER triple pattern

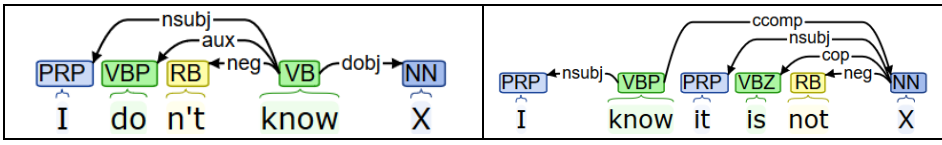
The present implementation of SPARQL/T employs the NER provided by the Stanford CoreNLP library [2], with the set of NE classes listed in Figure 37

PERSON	MONEY
LOCATION	NUMBER
ORGANIZATION	ORDINAL
MISC	PERCENT
SET	DATE
	TIME
	DURATION

Figure 37: List of Named Entity classes provided by the Stanford CoreNLP NER

7.4 Dependency Parsing

Proximity is a good indicator of relatedness, but it obviously often fails. For example, looking for a word that expresses a negation in proximity of the verb "to know" is a naïve (albeit sometimes effective) way of finding expression of the concept of <not knowing something>. But it is not enough for example to sharply distinguish between "*not knowing X*" (our target) and "*knowing that something is not X*". A Dependency Parser [Kubler 2009] makes this kind of distinction easier to make, as in the first case the head of the relation "*neg*" is the verb "to *know*", whilst in the second is the noun "*X*" (see Figure 38)




```
?k emb:any
'know'.
?k DEP:NEG   ?n
.
```

Figure 38: The Dependency Trees of the two sentences "I dont't know X" and "I know it is not X" [37]. Both sentences contain a negation in proximity of the verb "to know". The Dependency Trees allows to easily distinguish the two cases by looking at thehead of the relation "neg". On the right: the SPARQL/T triple patterns that searches in the sentence instances of the concept "not knowing", by using the Dependency Parsing together with the Word Embedding.

The syntax for the triple pattern is:

`?head`	`DEP:RELATION`	`?dependent`

[37] http://corenlp.run/

Watching only at the annotations, and not at the document, it extracts a relation of two columns, with all the possible couples (head, dependent) found in the sentence linked by the specified relation. The list of allowed relations depends on the parser employed. For English the Stanford CoreNLP[38] library has been adopted, which output grammatical relations in the Universal Dependencies v1 [Nivre 2016] [Schuster 2016]. The philosophy of Universal Dependency[39] (UD) is to provide a universal inventory of categories consistent across languages. At the time of writing UD tree banks were available for over 60 languages. For Italian the TINT library[40] [Aprosio 2016] has been used.

Dependency	Description	Dependency	Description
acl	adjectival clause	fixed	fixed multiword expression
advcl	adverbial clause modifier	flat	flat multiword expression
advmod	adverbial modifier	goeswith	goes with
amod	adjectival modifier	iobj	indirect object
appos	appositional modifier	list	list
aux	auxiliary	mark	marker
case	case marking	nmod	nominal modifier
cc	coordinating conjunction	nsubj	nominal subject
ccomp	clausal complement	nummod	numeric modifier
clf	classifier	obj	object
compound	compound	obl	oblique nominal
conj	conjunct	orphan	orphan
cop	copula	parataxis	parataxis
csubj	clausal subject	punct	punctuation
dep	unspecified dependency	reparandum	overridden disfluency
det	determiner	root	root
discourse	discourse element	vocative	vocative
dislocated	dislocated elements	xcomp	open clausal complement
expl	expletive		

Table 10: Universal Dependencies (http://universaldependencies.org/u/dep/index.html)

7.5 Semantic Role Labeling

According to Palmer et al., the syntactic parser "… are *long way from representing the full meaning of the sentences […] they do not specify 'Who did What to Whom and How, Where and When?*'" [Palmer et al. 2010]. A purchase event for example can be described using different verbs (buy, sell, acquire, …) and nouns (purchase, order, …), and the purchased thing can be the

[38] https://nlp.stanford.edu/software/stanford-dependencies.html
[39] http://universaldependencies.org/
[40] http://tint.fbk.eu/

subject or the (indirect) object of the verb, or be in a noun compound relation [Jurafsky et al. 2008]. The purchased thing and the purchaser are called Semantic Roles of the action purchase, and the Semantic Role Labeling (SRL) is the task of identifying them inside the text.
Compared with syntactic structures, SRL gives a much coarser representation of the sentence, generally easier to deal with. The roles can often be quite long descriptions of places, person, manners of behavior and events, and purely syntactic distinctions, like active vs. passive form of the verbs, are removed. [Dagan et al. 2013]. Moreover, at least in principle Semantic Roles should be highly independent from the language, simplifying the development of multi-language queries. The set of roles of an SRL tool depends on the linguistic resource against which it is trained. At the moment, there are three major resources, that differ primarily in the granularity of the role labels:

- FrameNet[41], based on Fillmore's Frame Semantics, is the most fine-grained. Different roles are defined for each of the 1224 (at the time of writing) different frames. For example, the Apply-Heat frame include Cook, Food and Heating Instrument roles (Frame Elements)
- VerbNet[42] consists of hierarchically arranged verb classes, with 24 semantic roles (Agent, Patient, Theme, Experiencer, …)
- PropBank[43] [Kingsbury & Palmer 2002] , explicitly developed for the purpose of becoming a training set for machine learning algorithms, defines semantic roles numerical (Arg0, Arg1, …), with Arg0 normally corresponding to the Agent, Arg1 to the Patient or Theme, and no consistent generalization across verbs for the higher numbered classes. It also defines Argument Modifier roles (ArgM) like location, temporal and manner.

SPARQL/T adopts (for the moment) only the PropBank set of roles. The general triple pattern is written using the SRL prefix followed by the PropBank role. For example, assuming V contains the verbs of the sentences in the document, the following triple pattern fills R with their Arg0 roles.

```
?R      SRL:ARG0      ?V
```

7.6 Word Net

The Word Net functions allows simple query expansion: the actual list of words to search inside the document is formed by taking each word or synset[44] in the list specified in the triple pattern and augmented it with every related word, according to Word Net and depending on the relation specified in the triple pattern itself. The possible relations are reported in Table 11.
For example, the first of the two following triple patterns will search for seven words: "*telephone*", "*phone*" and "*telephone set*" (the noun.artifact synset), but also for "*telephony*" (noun.communication) and "*call*", "*call up*" and "*ring*" (verb.communication) . The second one instead, having specified the synset code, will only consider the first three.

[41] https://framenet.icsi.berkeley.edu/fndrupal/
[42] https://verbs.colorado.edu/verbnet/
[43] https://propbank.github.io/
[44] Synsets are specified with their eight digit numbers, than can be queried online at the web site of the Princeton University: http://wordnetweb.princeton.edu/perl/webwn

?tel1	wn:synonym	"telephone"
?tel2	wn:synonym	"04408223"

synonym
hyponym
hypernym
similar
member_meronym
member_holonym
part_meronym
part_holonym
instance_hyponym
instance_hypernym
domain_category
domain_member_category
also
verb_group

Table 11: relations that can be used in th Word Net triple pattern

7.7 Sentence Embeddings

The sentences specified in the triple pattern are encoded employing the current encoder, at the moment either Google Universal Sentence Encoder or Facebook LASER. The nearest neighbors of the resulting vectors are then searched[45] and the corresponding snippets of text returned.
Like in the Word Embedding case, we can combine more than one sentence in the same triple pattern, in this case by separating them with any character in the set {'.', '!', '?' }. The sub-function code, as indicated in , specify what to do with the different sentences.

SEN:AVG	the query string is split into sentences and the average vector is employed in the search
SEN:ANY	the query string is split into sentences, the search is performed with each vector, and the best match is taken
SEN:EQU	the query string is taken as a single sentence

For example, the following triple pattern search for sentences similar to either "*Change the appointment*" or "*No show*".

?x	SEN:ANY	"Change the appointment. No show ".

7.8 Snippet concatenation and score threshold

This function performs two operations. Its main purpose is to allow the concatenation of the snippets of text retrieved inside the same BGP, to form a single snippet that extends from the beginning of the first to the end of the last (with respect to their position inside the document). Optionally, a threshold t>0 can be specified, that causes the removal of the results with score below t.
The example below searches for (opposite) sentiment words near an entity representing a telephone company, concatenates the results (variable S and O into variable C), and filters the records with lower

[45] At the moment, with a sequential search inside a binary file.

score (which may be due to words not so similar to “love” and “hate”, or too far apart from the company name). The threshold is arbitrary and depend on the context, which means that it has to be found manually, and that a good one may not even exist.

```
SELECT *
WHERE
{  { ?S   ELM:ANY    'love' . }
   UNION
   { ?S   ELM:ANY    'hate' . }
   ?O NLP:NER 'TEL_COMPANY' .
   ?C NLP:CAT "0.5" .
}
```

7.9 Lucene Queries

?var	IR:QRY	“Query String”

The query string is passed to Lucene and the returned set of documents is used in place of the entire corpus for the rest of the query. The purpose of this triple pattern is to speed up the query execution, and it must be the first of the query.

7.10 Unimplemented Functions

This section lists some functions that are either not yet implemented or still experimental. They are report here nevertheless because we believe they are essential for the proper use of SPARQL/T.

7.10.1 Constituency Parsing

Constituency parser create trees that represent the sentence in term of their constituents, i.e. Noun Phrases (NP), Verbal Phrases (VP) and so on (see for example [Jurafsky & Martin 2008]). At the moment this kind of parsers are not as widely available as the Dependency ones, but when they are, they allow a more practical approach to Information Extraction, especially when dealing with multi-word concepts. A single triple pattern may be used for example to extract the full n-gram object of a certain action (starting from the verb), or to retrieve the verb that refer to a certain entity (starting for example from a Named Entity). Of course, we need to deal with imprecisions, errors, and misalignments of various kind between different annotation tools. So, a first group of triple patterns must be created for alignment purposes: we need a way to reach the first valid node of a parse tree starting from any snippet of text. Let X be such snippet of text. The following triple patterns allows such searches:

?Y	CP_UPR:ANY	?X	Y = Any Upper node	The smaller node containing X entirely, of any kind
?Y	CP_UPR:NP	?X	Y = Upper NP node	The smaller Noun Phrase containing X
?Y	CP_LWR:ANY	?X	Y = Any Lower node	The largest node fully contained in X, of any kind
?Y	CP_LWR:NP	?X	Y = Lower NP node	The largest Noun Phrase fully contained in X

Here, for simplicity of exposition, only the Noun Phrase (NP) cases have been shown, but any other kind of node can be used in place of NP. Also, here "largest" and "smaller" obviously refer to the extension of the snippet of text of the node.
The remaining triple patterns assume instead that X is already aligned with a node of the tree, and allow to navigate it up and down:

?Y	CP_PAR:ANY	?X	Y = Any Parent of X	the parent of X (can be of any kind)
?Y	CP_PAR:NP	?X	Y = Parent NP of X	the parent of X, if it is a Noun Phrase
?Y	CP_ANC:ANY	?X	Y = Any Ancestor of X	Al the nodes on the path from X to the root (excluding X)
?Y	CP_ANC:NP	?X	Y = NP Ancestors of X	All the NP nodes on the path from X to the root (excluding X)
?Y	CP_CHL:ANY	?X	Y = children of X	All the children of X
?Y	CP_CHL:NP	?X	Y = children of X	All the Noun Phrases children of X
?Y	CP_DES:ANY	?X	Y = Any Descendant of X	Al the X subtree (excluding X)
?Y	CP_DES:NP	?X	Y = NP Descendant of X	All the NP nodes in the X subtree (excluding X)
?Y	CP_SIB:ANY	?X	Y = Any Sibling of X	Al the nodes descending from the parent of X (excluding X)
?Y	CP_SIB:NP	?X	Y = NP Siblings of X	Al the NP nodes descending from the parent of X (excluding X)

Notice that most expressions can sometimes return an empty relation. For example, the triple pattern

```
?Y    CP_PAR:NP   ?X
```

does not return any triple if the parent of X is not a Noun Phrase. Because the join of an empty relation with any other relation is again an empty relation, together with the UNION clause this fact can be used to implement a sort of conditional execution:

```
{   # if the parent of Y is a noun phrase
    ?Y      CP_PAR:NP   ?X
    …
}
UNION
{   # if the parent of Y is a verbal phrase
    ?Y      CP_PAR:VP   ?X
    …
}
```

7.10.2 Negations

Information Retrieval systems often disregards negation clues, treating them as stop words and thus becoming unable to discriminate between cases in which things that are stated to be present and others where the same things are stated to be absent. Although in many cases this may not be a problem, there are situations where being able to do this kind of distinction becomes critical. Clinical reports for example, according to Chapman et al [Chapman et al. 2001b], often contain lots of observations that denies the presence of a particular clinical condition. In their own words: "*many of the most frequently described findings and diseases in discharge summaries, radiology reports, history and physical exams, and other transcribed reports are denied in the patient. […] Differentiating pertinent negatives*[46] *from positive conditions in a clinical report is crucial to accurate indexing of the report*". More specifically, they found ([Chapman et al. 2001a]) that the number of pertinent negatives were between 39% and 83%, depending on the type of report considered.

[46] Here the term "*pertinent negatives*" indicates findings and diseases explicitly or implicitly described as absent in a patient.

A similar situation, probably with similar figures, arises with complaints, as they are often statements about things that the user expected to receive or achieve (and didn't), and with technical reviews, where similar products with different set of features are compared.
[Díaz & López 2019] provide a very recent survey on negation and speculation detection (speculations, or "edging", are phrases used to mark an assertion as not sure).
Correctly identifying a negation is not a simple task. With a certain approximation, it can already be done inside SPARQL/T, by looking for the Negation Clues (words like "*no*", "*not*", "*never*"), or by using a Dependency Tree (following the "`neg`" relation). However, there are many possible negation clues to consider, like pronouns ("*nobody*"), determiners ("*any*") and prepositions ("*without*"). Negation can also be affixal (negative prefixes like "*un-related*"), or can be expressed by verbs ("I *refuse* to talk") or adjectives ("an *imperceptible* smell invaded the room"). Moreover, identifying the scope, i.e. the exact snippet of text the negation refers to, is a highly difficult problem itself. See for example [Morante et al. 2011] (for English) and [Altuna et al. 2017] (for Italian). At the moment (according also to [Díaz & López 2019]), the only publicly available library for negation annotations (in English) seems to be the negtool[47] [Enget et al. 2017]. We believe that would be very important to have in the future a function in SPARQL/T that allows the user to identify a negation and its scope with a single triple pattern:

?X	NLP:NEG	"negation-form"

Here, X will bind to the scope of the negation, whilst the suitable set of negation-forms need t be studied according to usefulness and tools capabilities.

7.10.3 Textual Entailment

Being able to retrieve sentences with meaning similar to a given one, almost independently from the way the concepts are linguistically expressed, is already a wonderful and quite surprising result, achievable in SPARQL/T by using Sentence Embedding triple patterns, which in turn rely on tools like Google Universal Sentence Encoder of Facebook LASER. However, similarity is quite a vague concept, that may lead to many false positives (according to the user's information need). In some cases, what the user may want is to retrieve things that are equivalent from the logical point of view, i.e. things that entail one another. A definition of Textual Entailment, also known as Natural Language Inference (NLI), can be found for example in [Dagan et al. 2013]:

> **Textual Entailment** is a directional relationship between pair of text expressions T and H (Text and Hypothesis). We say that T entails H if humans reading T would typically infer that H is most likely true

In other words, Textual Entailment consider plausibility of the hypothesis, not just certainty. Similarly, they also define contradiction:

> The hypothesis H of an entailment pair **contradicts** the text T if a human reader would say that H is highly unlikely to be true given the information described in T

[47] https://github.com/marenger/negtool

As an example, the text "*The drugs that slows down or halt Alzheimer's disease work best the earlier you administer them*" entail the hypothesis H "*Alzheimer's disease is treated using drugs*". From the procedural point of view, three approaches to Textual Entailment are generally defined:

- Recognition Mode: given T and H, classify whether entailment holds or not
- Search Mode: given H and a corpus, find all text fragments that entails H
- Generation Mode: given a text T, generate sentences that entails H

From the SPARQL/T point of view, Search Mode is clearly what is needed. Ideally, a triple pattern like the following one should fill the relation X extracting from the documents all the sentences that entail the possibility of treating Alzheimer with drugs:

```
?X    NLP:ENTAIL "Alzheimer's disease is treated using drugs"
```

At the moment, results are still experimental, but promising. The Stanford Natural Language Inference (SNLI) corpus [Bowman et al.2015] is a collection of 570152 sentence pairs labeled for entailment, contradiction, and semantic independence. It has been manually created by about 2500 workers using Amazon Mechanical Turk, starting from the captions of the Flickr30k corpus. According to the authors, the dataset is SNLI is large and diverse enough to be used to train Deep Neural Networks models. In their test with a LSTM they obtained accuracy above 77%. Moreover, having employed Word and Sentence Embeddings, there is the hope that multi-language versions of such embeddings will allow a zero-shot transfer learning for languages where a corpus as big as SNLI does not exist.

7.10.4 Sentiment Analysis

Although the primary aim of this work is to deal with the details of the problems exposed in complaints and reviews, finding the snippets of text that clearly express a negative sentiment will certainly help to focus the search. For this purpose, a simple syntax like the following should allow the user to extract such snippets:

```
?X      NLP:SEN      "sentiment-level"
```

Here, sentiment-level should be a label in a restricted range to be defined, like { "Very Positive", "Positive", "Neutral", "Negative" and "Very Negative" }

7.10.5 OpenIE

OpenIE extracts from documents triples of strings in the form *(subject, predicate, object),* without spending effort in trying to relate them to some ontology, nor to put them into a canonical form (see section 3.1). However, for the SPARQL/T point of view, this will probably be a very interesting resource. Because of their structures, we need to reserve to OpenIE triples a specific triple pattern: a triple pattern with three variables extracts from the sentence all its OpenIE triples:

```
?Subject      ?Predicate      ?Object
```

(which resembles the SPARQL case, where a triple pattern with three variables retrieves the entire graph)

Chapter 8 - Architecture

The actual implementation of SPARQL/T engine has three main components (see also Figure 40):

- The "core", NLP one, that deals with natural language documents
- The RDF/OWL one, that uses Apache Jena[48]
- The traditional Search Engine one, implemented with Apache Lucene[49]

The NLP part is almost entirely written in Java, with very small Python parts that interfaces with the Sentence Encoders. For the parsing of the queries the Antlr 4 library[50] [Parr 2013] has been used, together with an initial SPARQ 1.0 grammar available from the author repository[51]. The corpus of natural language documents is stored on groups (segments) of three files each:

- The actual documents with their annotations, serialized into a JSON file
- The Sentence Embeddings, saved as a raw binary file
- A small index files that provides the links between the two

The segments are generated by the user, normally according to source, topic and dates. To focus the search, or simply to speed up query execution, the user is allowed to restrict the set of segments to work with[52]. A special segment (the Working Segment) is intended to provide a meaningful set of documents of interest, where the user can test the queries and train the Machine Learning filters. The Working Segment is kept in RAM, possibly with the Embedding part in the GPU RAM. Its optimal size obviously depends on the hardware capabilities. Ideally, it should be big enough to provide a suitable number of examples to work with, but also small enough to allow an almost real time execution of the test queries. It can be loaded from disk or generated on the fly with a Lucene query, collecting from the Corpus segments the document whose id is returned by Lucene. Although Lucene queries are primarily intended for the purpose of building the Working Segment, they can also be part of the SPARQL/T query itself (see section 7.9), allowing to quickly select at run time a limited number of documents to submit to the heavier NLP analysis[53].

An important feature of SPARQL/T is that, thanks to its SPARQL syntax, NLP triple patterns that refer to the documents and their annotations can be seamlessly intermixed with traditional SPARQL ones that refer to an RDF/OWL graph. This kind of hybrid queries is achieved using Apache Jena[54] RDF API and ARQ. Figure 41 shows an example of how a hybrid query is executed. The hybrid query on top left of the figure has two BGP (Basic Graph Patterns). The first one in an NLP one, that looks inside the documents' annotations for all the Entities of the kind 'TEL_OFFER'. The function NLP:EL (where 'EL' stays for 'Entity Linking') is the only one in SPARQ/T that return an URI together with the snippet of text to which it refer.

[48] https://jena.apache.org/
[49] https://lucene.apache.org/
[50] https://www.antlr.org/
[51] https://github.com/antlr/grammars-v4
[52] It cannot be considered a true faceted search, as it follows the structure of different forums, that unfortunately in general have different subdivision into topics.
[53] For this reason, Lucene statements must appear in the SPARQL/T query before any NLP triple pattern. Albeit inelegant, this choice seems acceptable for most practical purposes.
[54] https://jena.apache.org/

The second BGP is an RDF/OWL one, and is passed to Apache Jena, decorated with some prefixes and a "SELECT {} WHERE {}" clause[55]. However, the two BGPs share a common variable X, which means that its values must be taken from the result set of BGP1 and passed to Jena (Variable Bindings). Jena executes the query and returns its result set. In this case, it checks that X is the kind of offer that gives unlimited access to social networks, and returns its monthly cost. Finally, control returns to SPARQL/T, that performs the JOIN operation (on X) between the results of the two BGP. Notice that in this simple case we only have variables with associated URIs. In general, this may not be the case: we may have for example a variable Y that represent a snippet of text in BGP1 and an RDF literal in BGP2. Jena, as any other SPARQL engine, cannot deal with similarity measures. Therefore, text only variables must not be passed to Jena, but treated in the JOIN operation employing string similarity measures. Figure 39 gives the pseudo-code for the Hybrid Query execution

```
A=set of variables belonging solely to the input relation
B=set of variables belonging to both the query and the input relation
C=set of variables belonging solely to the query

query = QueryFactory.create(QueryString) ;
model = RDFDataMgr.loadModel(RdfFile);
foreach(row in InputRelation)
{   QuerySolutionMap qsm=new QuerySolutionMap();
    foreach(var in B)
    {   if(var contains URI)
        {   Resource res =model.createResource(sURI);
            qsm.add(var.Name,res);
        }
    }
    QueryExecution qx=QueryExecutionFactory.create(query,model,qsm))
    ResultSet rs=qexec.execSelect();
    while(rs.hasNext())
    {   QuerySolution qs=rs.nextSolution();
        float score=row.score;
        foreach(var in B)
        {   RDFNode nd=qs.get(var.Name);
            if(nd.isLiteral())
            {    sim=similarity(nd.asLiteral(),var.Value);
                 score=adjustScore(score,sim);
            }
        }
        if(score>0)
            addSolution(A∪B∪C,score);
    }
}
```

Figure 39: Pseudo-code of Hybrid Query execution

[55] Notice that, for the sake of clarity but also for simplicity of implementation, triple patterns of two different kinds (NLP or RDF) must form homogeneous groups, syntactically enclosed in curly braces.

In general, the hybrid query execution proceeds along the NLP triple patterns, producing the usual relation (result set RS^{TXT}), until an RDF group is encountered. Then, the result set produced up to that point is passed to the Jena engine, together with a SPARQL SELECT query containing the RDF triple patterns in the BGP. The result set RS^{RDF} returned by Jena is then joined with the previous RS^{TXT} to produce the output one RS^{OUT}, and the execution continues.

The values of the variables of RS^{TXT} that are also used inside the RDF query are passed to Jena through a **`QuerySolutionMap`** object[56]. Unfortunately, it seems that this has to be done one record at a time, i.e. it is not possible to pass the entire relation to Jena, but we need to run the RDF query for each of the RS^{TXT} records. However, query compilation can be placed outside the loop (**`QueryFactory.create`** function). Moreover, being Jena an open source project, higher performances can probably be achieved in the future by tackling inside its code. As already mentioned, because SPARQL engines cannot deal with similarity measures, we only pass them the variables containing URIs. More specifically, let T be the set of variables that appears in RS^{TXT} and R those of RS^{RDF}.

We split T and R into three sets:

- Let $A = T \setminus R$ be the set of variables of RS^{TXT} that are not present in RS^{RDF}.
- Let $B = T \cap R$ be the set of variables present in both RS^{TXT} and RS^{RDF}.
- Let $C = R \setminus T$ be the set of variables of RS^{RDF} that are not present in RS^{TXT}.

B is then the set of variables involved in the JOIN operation, and is further divided into:

- B^{URI} : the subset of variables in B that contains URIs
- B^{STR} : the subset of variables in B that contains strings (snippets of text in RS^{TXT}, RDF literals in RS^{RDF})

The values of the B^{URI} variables are passed to Jena inside a `QuerySolutionMap` object, whilst the B^{STR} ones are used to calculate the score of the output record. The output record is formed by the concatenation of the values of the input record for the variables in the sets A and B, and the values of the Jena query for the variables in the set C[57].

[56] An obvious optimization, omitted in Figure 39 for simplicity, is to group records with the same values

[57] Which means that when literals variables from the input records differs from those of the query, we (arbitrarily) keep the former.

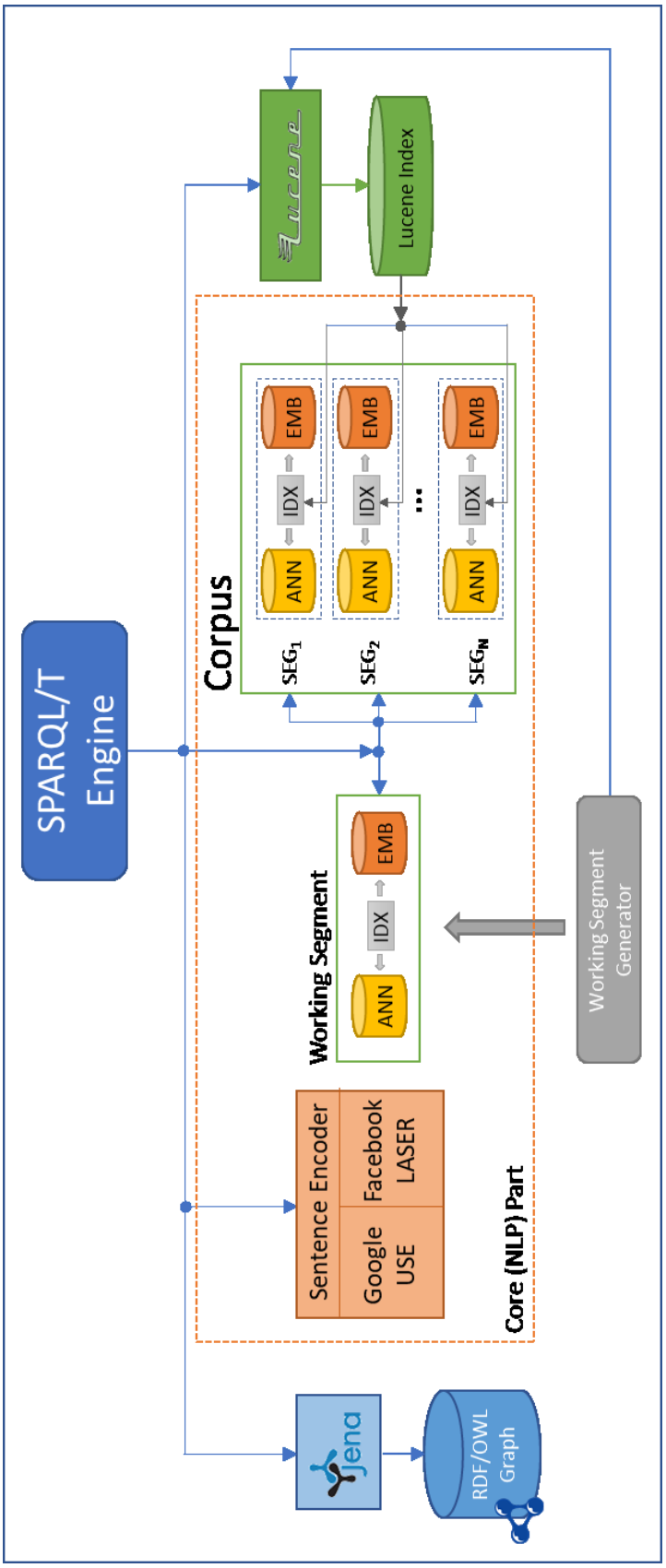


Figure 40: SPARQL/T Engine Architecture

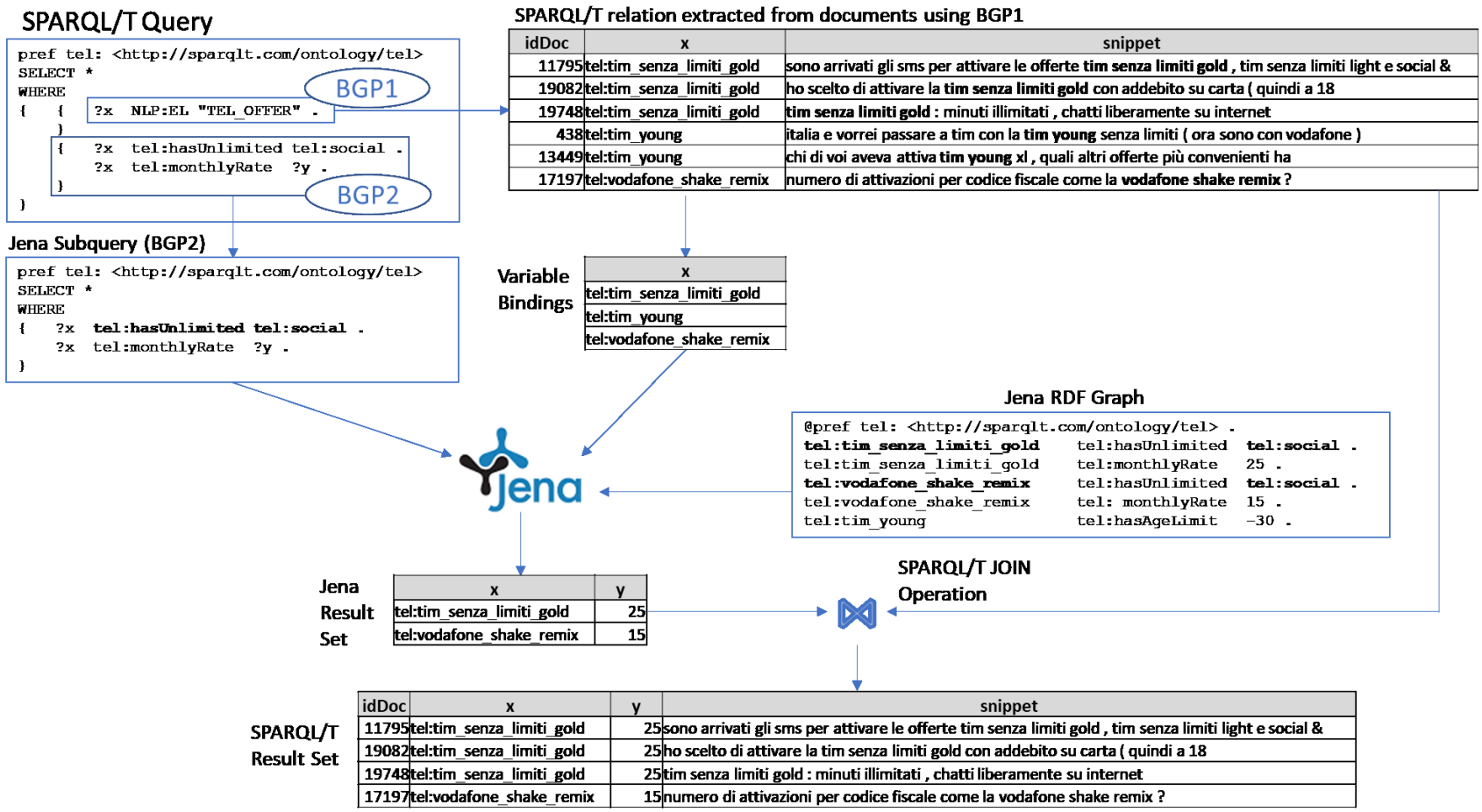


Figure 41: Example of Hybrid Query execution

Chapter 9 - Evaluation

This chapter evaluates the present implementation of SPARQL/T. Section 9.1 shows an example of its simplicity of use; section 9.2 measures its performances for the most common operations.

9.1 - Example – Low Effort Queries

This example shows the ease of use of SPARQL/T, which does not require the users to be aware of many of the available triple patterns to do something useful. The expressivity of the language is indeed illustrated by two very simple option allowed by SPARQL/T:

- Enter an entire sentence in a single triple pattern and rely on sentence similarity
- Pick only a few words for each concept of interest and compose a simple query, with a triple pattern for each concept, relying then on the internal join operation to properly combine the results

Suppose that we need to identify mentions of hard disk failures within a group of sentences/documents. By googling "*hard disk failure*" it is easy to spot a few web sites that make their own list of such possible symptoms. We have picked one of those, and quite mindlessly collected the major points, reported in Table 12. We can think at Table 12 as the limited initial knowledge that the user may have on a subject before starting an exploration.

SYSTEM	
S1	system fails to boot
S2	system freezes
S3	black screen appears
S4	sudden shutdown
S5	force restart
FILES & DIRECTORIES	
F1	hard disk is not recognized
F2	files and folders become invisible and corrupt
F3	bad sectors and block appear in the hard disk
F4	system files are altered
ELECTRONICAL	
E1	power source is unreliable
E2	power supply too high or too low
MECHANICAL	
M1	hard drive does not spin
M2	hardware makes noise
HEAT	
H1	fans do not work
H2	computer get heated

Table 12: Possible symptoms of Hard Disk Failure

We have then collected, from the same Google query, a set of 60 HTML documents (excluding the one used to build Table 12), kept the initial 5000 character of the text of each one, and manually labeled them with the labels of Table 12 (It is obviously a multi-label classification problem, as the same document may report more than a symptom). A random sample of 20 of those documents has been used for exploration and to trim the query parameters. Let's call it, quite improperly[58], the training set. All the remaining documents constitute the test set.
Three kinds of queries have been made:

1. Trivial queries of just one triple pattern: the `EMB:USE` triple pattern accepts in input a sentence and, employing Google Universal Sentence Encoder (USE) [Cer at al. 2018], fills a relation with the sentences of the document similar to the given one. Here, the sentences of Table 12 have been used exactly as they are.
2. To improve precision, to the same queries a simple filter has been added (`NLP:CUT`), which applies a threshold to the score of each result. Results below the threshold are discarded. Instead of manually choosing the best threshold for each case, for the sake of comparison, the thresholds have been arbitrarily chosen as the ones that maximize the F1 score in the training set.
3. We finally manually tried to improve over that precision with simple queries of two or three triple patterns. The `LEM:ANY` triple pattern accepts a list of lemmas of words and fills a relation with any word in the document whose lemma is contained in the list. (As a reminder, a result is returned from a sequence of triple patterns[59] when the elements extracted by each pattern co-occur inside the same sentence. Results are then scored according to their distance). Again, for the sake of comparison, only triple patterns of this kind have been used in these queries, whilst the lists of lemmas have been manually collected from the documents of the training set.

The following figures report examples of queries of type 2 and 3, for the label E1 (power source unreliable) together with their respective results on the test set.

```
SELECT *
WHERE
{     ?E1   EMB:USE    'power source is unreliable' .
      ?TH   NLP:CUT    '0.750' .
}
```

Figure 42: query of type 2 for case E1 (power source is unreliable). It consists of a triple pattern using Universal Sentence Encoder to measure sentence similarity, followed by one that specifies a threshold. The threshold is the value that maximizes, in the training set, the F1 score.

[58] There is no Machine Learning involved here, so the terminology Test/Training Set is quite abused. However, in a sense, in selecting the words to use and in trimming the threshold of the score we are still "teaching" the query what to return. In a more realistic set, many trial and error test would probably need to be performed in a limited number of examples, also involving different graph patterns, before running the lengthy job on the full dataset.
[59] In the case that, like here, they belong to the same BGP

Score	Extracted snippet
0.797	*while an electronic failure is often due to external issues such as a power spike, electrical surge or a major fluctuation in voltage.*
0.793	*lightning or power surges.*
0.757	*in reality, this is not something that computer fixers do, and the cost could be in the 1000 's if you can find the appropriate experts.*
0.757	*sudden power surges due to power outages will cause hard drive failure*

Figure 43: Results of query of Figure 42, applied to the Test Set

```
SELECT *
WHERE
{     ?PW   LEM:ANY    'power voltage electricity electrical'.
      ?FA   LEM:ANY    'surge outages lighting strikes failure'.
}
```

Figure 44: Query of type 3 for case E1 (power source is unreliable). It consists of few (in this case 2) triple patterns that search for the lemmas of some specific words. Proper words are most likely found by skimming the training set.

Score	Extracted snippet
1.000	*electrical failure.*
1.000	*strikes, can cause electrical failure.*
1.000	*functions, then an electrical failure may be the source*
0.889	*power surges due to power outages will cause hard drive failure.*
0.741	*while an electronic failure is often due to external issues such as a power spike, electrical surge*

Figure 45: Results of query of Figure 44, applied to the Test Set

The limited number of documents employed in this test does not allow to say a lot on the results. However, as shown in Table 13 and quite as expected, when applied to the Test Set, queries of type 2 give better recall, whilst queries of type 3 gives in general better precision. Overall, even this trivial approach, that totally disregards the structure of the sentence, often gives acceptable results, at least in an exploratory context.

		Single triple 'USE' query with threshold		Simple Query with 2 or 3 triple patterns	
		Pre	Rec	Pre	Rec
E1	power source is unreliable	16.7	100	50	50
F1	hard disk is not recognized	33.3	80	50	80
F2	files and folders become invisible and corrupt	50	66.7	80	44.4
H2	computer get heated	100	75	75	75
M2	hardware makes noise	100	38.5	100	46.2
S1	system fails to boot	62.5	71.4	20	14.3
S2	system freezes	62.5	62.5	100	62.5

Table 13:Precision and Recall of some queries of type 2 and 3 applied to the TEST set

9.2 Performance Measures

The following tests have been performed on an "average" PC, mounting an AMD Athlon X4 880k at 4GHz, with 16 GB of RAM, SSD disk and no GPU[60]. The dataset consists of 20293 messages downloaded from a single Italian forum on Telephony, mostly regarding the TIM telephone company, and related to the year 2018. The Word Embedding employed is a 300 vector one trained with fastText[61] and downloaded from the fastText web site[62]. Time is measured in millisecond, using java System.currentTimeMillis(), averaged over 10 trials. In many circumstances execution time depends on the memory constraint imposed, i.e. on the size of the window used to extract the relation (see chapter 6.4.6). Therefore, two kinds of measurement are here reported:

- With a small window size of 10 elements (indicated by S)
- With a large window size of 1000 elements (indicated by L)

The L window is large enough to perform unrestricted crispy searches (like words or lemma), but is still a limiting factor for Embedding searches: almost any word is similar to almost any other, albeit by a very small amount, which imply that the relations are fully filled at each step of execution.

The first test is a simple search of a word in the entire corpus (function `WRD:ANY`). Four words have been chosen, with an (almost) exponential decreasing frequency (measured on the corpus itself). As expected, with the memory constrained search S, which is supposed to be the normal case, we have a time that is almost independent from the size of the result[63]. Obviously, for the unconstrained search L the dependency is evident.

word	freq	Avg Time S	Avg Time L
ho	1.0567%	35	169
attivata	0.1010%	32	46
richieste	0.0100%	31	34
usate	0.0010%	31	32

For the Word Embedding case instead (`EMB:ANY`), for the reason just explained, time does not depend on the word frequency.

word	freq	Avg Time S	Avg Time L
ho	1.057%	780	1563
attivata	0.101%	777	1540
richieste	0.010%	779	1543
usate	0.001%	781	1556

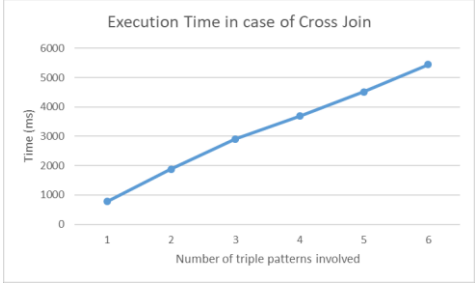
A basic form of Sentence Embedding can be achieved by calculating the sum of the vectors over a sliding window on the text (function `EMB:SUM`). As the following table shows, this can be achieved without performance concerns.

[60] With the present implementation, the only timing that could benefit from GPU are those involving Sentence Encodings, but only during the encoding of the encoding of the query sentences.
[61] https://fasttext.cc/
[62] fasttext-italian-cc.it.300.vec, from https://fasttext.cc/docs/en/crawl-vectors.html, fetched May 30, 2019
[63] Which also means, as discussed in section 6.4.6, that some potentially useful results can be missed.

words	Avg Time S	Avg Time L
ho	780	1563
ho attivato	1003	1780
ho attivato una	1073	1831
ho attivato una sim	1140	1899
ho attivato una sim vodafone	1191	1969

The following table and graph show the execution time when a CROSS JOIN operation is involved between the results of two consecutive triple patterns, i.e. in the case that they have no variable in common. As expected for the memory constrained case, time grows linearly with the number of triple patterns involved.

Triple Pattern	Avg Time S
`?X1 EMB:ANY "attivare".`	777
`?X2 EMB:ANY "richiesta" .`	1885
`?X3 EMB:ANY "sim" .`	2903
`?X4 EMB:ANY "vodafone"`	3694
`?X5 EMB:ANY "ieri"`	4516
`?X6 EMB:ANY "pomeriggio"`	5444



The last following table show the performances when an INNER JOIN is involved, i.e. when triple patterns shares some variables. The time becomes almost constant, as the increasing number of patterns to extract is compensated by the decreasing number of matches found at each step.

Triple Pattern	Avg Time
`?x LEM:ANY "attivare" .`	109
`?x NLP:POS "verb" .`	129
`?x DEP:NSUBJ ?y .`	117
`?y NLP:POS "noun" .`	118
`?y DEP:DET ?z .`	138

In conclusion, albeit in absolute terms there is still a lot of space for improvement, the actual SPARQL/T implementation does not present more than linear behavior on the most common and useful operations.

10 – Future Work

For time constraints, many things described in the Conceptual Model, like the GRAPH and the OPTIONAL clauses, are not yet properly implemented. Many useful NLP functions, detailed in section 7.10, are also missing, and the Reasoning Interface described in section 6.6.2 must be focused more, and more precisely defined. However, an even greater and more exciting development goes into the direction of Computer Vision (CV). Compared with the analysis of Natural Language texts, the analysis of images presents similar aims, employs similar techniques and ultimately reveals very similar problems. This suggest that SPARQL/T can have with CV a role very similar to the one it has in Natural Language Processing (NLP), allowing the user to employ the same query language to retrieve, classify and extract information from both text and images (or videos). It should also allow to refer to both modalities together, for example by working with a video and its commentary audio at the same time. Moreover, it should hopefully avoid the user to make explicit distinctions between the two modalities, allowing (some) triple patterns to seamlessly match either images regions or text snippets, whichever is more similar to the query pattern. The rest of the chapter is organized as follows: section 10.1 analyzes various CV tasks, comparing them with their NLP counterparts; section 10.2 consider related works; section 10.3 briefly exposes a possible practical approach.

10.1 CV-NLP Task Comparison

Object Detection and Recognition is the CV task of detecting objects inside images, finding their spatial location (for example, their bounding boxes) and classifying them into a certain number of predefined categories. A very recent survey of the field is given in [Liu et al. 2019]. Actually, there are two types of object detection: detection of categories (human, dog, bicycles …), and detection of specific instances, like a specific actor, building and so on. The former resembles the NLP task of **Named Entity Recognition** (NER), whilst the latter is the CV equivalent of the Entity Linking / **Named Entity Disambiguation** one (see for example [Nouvel et al. 2016]). Although Object Detection is still an open problem, nowadays, according again to [Liu et al. 2019], precision above 70% can be reached with detectors trained for almost one hundred classes, which is already clearly a good result.

Krizhevsky et al. observed that the vectors produced by the last hidden layer of a Convolutional Neural Network trained for image classification are similar (in terms of Euclidean distance) for similar images (as judged by humans) [Krizhevsky et al 2012]. Babenko et al. successfully used that vectors (named **Neural Codes**) in the context of Image Retrieval [Babenko et al. 2014]. In other words, Neural Codes forms the CV counterpart of the NLP **Word Embeddings**.

The Facial Expression Recognition (FER) task[64] aim at recognizing in images and video the human face expression. Most of the times it refers to the seven expressions defined by Ekman & Friesen (Happiness, Sadness, Surprise, Fear, Anger, Disgust, Contempt) [Ekman & Friesen 1978] [Ekman & Friesen 2002]. In a recent survey Ko et al. compared (on a same dataset) different conventional

[64] Sometimes referred as "Facial Emotion Recognition" task [Ko et al. 2018]

and Deep Learning approaches to the FER task, finding an average accuracy of 63.2% on the former and 72.65% on the latter [Ko et al. 2018]. Besides facial expressions, emotions can also be detected observing (in video and audio) nonverbal cues like the tone of voice (para-language), body movement, and physiological changes (skin color) [Avots et al. 2018]. Moreover, the task can be extended to identify the emotion of an entire group of people (Group-level Emotion Recognition, GReco [Dhall et al. 2017]). These are clearly new and more difficult tasks, for which a series of challenges exists from 2013 (Emotion Recognition in the Wild, EmotiW, [Dhall et al. 2016]). From the SPARQL/T point of view, **Emotion Recognition** of any kind can be considered the CV counterparts of **Sentiment Analysis**.

Images normally contains more than an object, and there are actually many different tasks in CV that consider interactions between objects of the scene. **Visual Relationship Detection** is the task of detecting relationships between objects in an image, generating triples of the form (*object*$_1$*, relationship, object*$_2$) [Lu et al 2016]. The task can be restricted for example to spatial relationships (above, below, inside, around), or to human-object interaction (HOI). As pointed out by Lu et al.: "*… it is the relationship between the objects that determine the holistic interpretation of the image […] an image with a person and a bicycle might involve the man riding, pushing, or even falling off of the bicycle*". The NLP counterpart of the CV Relationship Detection is probably the **OpenIE** philosophy of Information Extraction. HOI task can be limited to classify the coarse activity of the person ("*playing baseball*", "*cooking*"), or can go into the details, considering the possible semantic roles of the specific action ("*hitting the ball with a bat*", "*chopping onions with a knife*"). In the latter case it is called **Visual Semantic Role Labeling** [Gupta & Malik 2015]. Finally, objects, attributes and relationships of a scene can be combined in a **Scene Graph** [Johnson et al. 2015], which is the CV form of the NLP **Knowledge Extraction** task.

Table 14 reports the CV task listed here together with their NLP counterparts.

Computer Vision task	Natural Language Processing task
Object Detection and Recognition	Named Entity Recognition / Disambiguation
Neural Codes	Word Embeddings
Emotion Recognition	Sentiment Analysis
Visual Relationship Detection	Open Information Extraction
Visual Semantic Role Labeling	Semantic Role Labeling
Scene Graph	Knowledge Extraction

Table 14: CV - NLP task analogy

10.2 Related Work

Image and Video captioning is the task of generating a natural language description of an image or portion of video. In fact, if the captioning contains a complete description of the image there would be no need to adapt SPARQL/T to CV. However, this is often not the case. Captioning is expected to capture the salient features of a scene, leaving out most details. But what is relevant is a matter of what we are looking for, and cannot in general be predicted before-hand.

Scene Graphs can also be an approach alternative to SPARQL/T. They are meant to formally represent the scene, typically in RDF form. Although in this case there is no limit to the level of detail represented, the problem of extracting the semantic graph from a scene looks not easier

than the problem of extracting it from a document, i.e. to the task of Knowledge Extraction described in Chapter 2. Therefore, it is likely to suffer from the same issues we have described: semantically similar scenes are likely to generate graphs with very different structures, making the query construction very cumbersome from the user point of view.

10.3 Practical Approach

As observed for example by Barnard in [Barnard 2016], the relation between visual and linguistic information can be from largely intersecting to largely disjoint. Informally, he defines two extremes:

- Redundant: the two modalities are informative about each other
- Orthogonal: they completely independent from each other

In between these two extremes lies a continuum of complementary possibilities, where both modalities provide information about the same things, but are not entirely redundant.

As an example, a video intended for didactic purposes and its auditory comments will probably be highly redundant, whilst annotations intended for colleagues tends to be highly orthogonal, like some short notes about what to do.

To allow the user to easily deal with different degrees of orthogonality, we would like a query to seamlessly adapt to both modalities. He should not be forced to write different triple patterns to express the same concept in visual and textual terms. Let's say for example that we want to retrieve all the scenes from a set of movies where *someone is cooking broccoli*. The following query, albeit with some uncertainty, should do the job:

1	?X	NLP:POS	"verb"
2	?X	LEM:ANY	"cook"
3	?Y	EMB:ANY	"broccoli"

Notice that, for the sake of explanation, both crispy (using lemma) and fuzzy (using Word Embedding) Triple Patterns have been employed.

We expect three cases:

A. The information is fully contained in the text (subtitles), for example in a sentence like "*Mom is cooking broccoli*". Here the classical SPARQL/T rules apply:
    1. All the words x_i that are tagged as verbs and whose lemma is "*cook*" are inserted into X (Triple Patterns 1 and 2)
    2. All the words y_j similar to "*broccoli*", according to a certain Word Embedding, are inserted into Y (Triple Pattern 3)
    3. All the possible couples (x_i,y_j) are generated, provided x_i and y_j belong to the same sentence, and then ranked according (among other things) to the distance between the words x_i and y_j (number of tokens in between the two)
B. The information is fully contained in the visual part, i.e. the picture clearly shows somebody cooking broccoli. Then, in a way similar as before:
    1. The actions x_i of the scene are extracted, and the ones whose label is "*cook*" are inserted into X (Triple Patterns 1 and 2)

2. The Neural Codes of the objects y_j resulting from Object Detection are compared against a prototype Neural Code of some broccoli. The most similar objects are inserted into Y (Triple Pattern 3)
3. All the possible couples (x_i,y_j) are generated, this time ranked according to the geometric distance between x_i and y_j in the scene

C. The information is distributed among modalities. For example, a video may clearly show somebody cooking, but ingredients may not be easily distinguishable. However, a someone is saying "… *we first cut the broccoli …*". In this case:
    1. The first two Triple Patterns find a match in the image (x_i extracted like in B1)
    2. The last one finds a match in the text (y_j extracted like in A2)
    3. The couples (x_i,y_j) are ranked according to overlap in time between image and voice

From the query engine point of view thus, it is just a matter of trying both modalities and keep the best results. What really changes are the three different way of performing the join: based on word distance in the text, on geometrical distance in the video, and on time distance when considering both together.

10 – References

[Aggarwal 2018] Aggarwal C.C. (2018), Machine Learning for Text. Springer

[Altuna et al. 2017] B. Altuna, A.L. Minard, M. Speranza. 2017. The Scope and Focus of Negation:A Complete Annotation Framework forItalian. InProceedings of the WorkshopComputational Semantics Beyond Eventsand Roles, pages 34–42.

[Andrian 2009] 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).

[Angeli 2014][OIE?006] G. Angeli, C. D. Manning, "NaturalLI: Natural logic inference for common sense reasoning," in Proceedings of the 2014 Conference on Emprical Methods in Natural LanguageProcessing, Doha, Qatar, 25 to 29 October 2014 (Association for Computational Linguistics, Stroudsburg, PA, 2014), pp. 534–545

[Angeli 2015] Gabor Angeli, Melvin Johnson Premkumar, and Christopher D. Manning. 2015. Leveraging linguistic structure for open domain information extraction. In ACL.

[Aprosio 2016] Italy goes to Stanford: a collection of CoreNLP modules for Italian By Alessio Palmero Aprosio and Giovanni Moretti. eprint arXiv:1609.06204.

[Avots et al. 2018] Avots, E.; Sapinski, T.; Bachmann, M.; Kaminska, D. Audiovisual emotion recognition in wild. Mach. Vis. Appl. 2018, 1–11.

[Corcoglioniti et al 2016] 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)

[Babenko et al. 2014] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. InProc. ECCV, 2014

[Baroni et al 2014] Baroni M., Dinu G., Kruszewski G., 2014, Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In Proceedings of ACL, 238–247.

[Banko 2007] 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.

[Barnard 2016] Barnard, K. (2016). Computational Methods for Integrating Vision and Language. Morgan and Claypool Publishers.

[Bast 2012] H. Bast, F. B¨aurle, B. Buchhold, E. Haussmann, A case for semantic full-text search, in: Proceedings of the 1st Joint International Workshop on Entity-Oriented and Semantic Search, JIWES '12, ACM, 2012, pp. 4:1–4:3.

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

[Bast 2015] H. Bast, F. Baurle, B. Buchhold, E. Haussmann, Broccoli: Semantic full-text search at your fingertips, CoRR abs/1207.2615.

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

[Beekhuizen 2018] Beekhuizen, Barend, Sasa Milic, Blair Armstrong & Suzanne Stevenson (2018). What Company Do Semantically Ambiguous Words Keep? Insights from Distributional Word Vectors. Proceedings of the 40th Annual Conference of the Cognitive Science Society

[Beg & Ashraf 2009] I. Beg and S. Ashraf, Similarity measures for fuzzy sets, Appl. and Comput. Math., 8(2)(2009), 192-202.

[Bezdek & Pal 1996] J.C. Bezdek and S. Pal, Fuzzy Models for Pattern Recognition (IEEE Press, New York, 1996) chapter 1.

[Bhagdev 2008] R. Bhagdev, S. Chapman, F. Ciravegna,V. Lanfranchi, and D. Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In ESWC, pages 554-568, 2008.

[Bobillo & Straccia 2011] Fernando Bobillo and Umberto Straccia. Fuzzy ontology representation using OWL 2. International Journal of Approximate Reasoning 52, 7 (2011), 1073–1094.

[Bollegala et al. 2017] Danushka Bollegala, Kohei Hayashi, and Ken-ichi Kawarabayashi. 2017. Think globally, embed locally—locally linear meta-embedding of words. arXiv preprint arXiv:1709.06671.

[Bowman et al.2015] S.R Bowman, G. Angeli,C. Potts, C.D. Manning. 2015.A large annotated corpus for learning natural languageinference. InProceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.

[Brambilla et al 2017] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. Synthesis Lectures on Software Engineering, 3(1):1–207, 2017.

[Buitelaar 2005] P. Buitelaar, P. Cimiano, and B. Magnini 2005. Ontology learning from text: An overview. In Ontology Learning from Text: Methods, Evaluation and Applications, P. Buitelaar, P. Cimiano, and B. Magnini, Eds. IOS Press, Amsterdam.

[Cafarella 2006] 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.

[Cer at al. 2018] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzweil. Universal Sentence Encoder. arXiv:1803.11175, 2018.

[Chambers 2007][OIE?007] Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine de Marneffe, Daniel Ramage, Eric Yeh, and Christopher D. Manning. 2007. Learning Alignments and Leveraging Natural Logic. In ACL-07 Workshop on Textual Entailment and Paraphrasing.

[Chapman et al. 2001a] Chapman WW, Bridewell W, Hanbury P, Cooper GF, Buchanan BG. Evaluation of negation phrases in narrative clinical reports. Proc AMIA Symp 2001; 105–9.

[Chapman et al. 2001b] Chapman W, Bridewell W, Hanbury P et al. A simple algorithm for identifying negated findings and diseases in discharge summaries. J Biomed Inform 2001;34:301–10.

[Chen et al 2018] Chen Z, He Z, Liu X, Bian J. Evaluating semantic relations in neural word embeddings with biomedical and general domain knowledge bases. BMC Med Inform Decis Mak. 2018;18(Suppl 2):65. https://doi.org/10.1186/s12911-018-0630-x.

[Chiarcos 2015] Christian Chiarcos and Maria Sukhareva. 2015. Olia – ontologies of linguistic annotation. Web Semantics: Science, Services and Agents on the World Wide Web, 6(4):379–386.

[Cleverdon 1966] Cleverdon, C. W.,Keen, E. M. Factors Determining the Performance of Indezing Systems, Vol. 1,2. Cranfield,England. Aslib Cranfield Research Project, 1966.

[Coates & Bollegala 2018] Joshua Coates and Danushka Bollegala. 2018. Frustratingly easy meta-embedding – computing meta-embeddings by averaging source word embeddings. In Proc. of NAACL-HLT.

[Corcoglioniti et al. 2016] Frame-based Ontology Population with PIKES. By Francesco Corcoglioniti, Marco Rospocher, and Alessio Palmero Aprosio. In IEEE Transactions on Knowledge and Data Engineering, 2016, vol. 28, no. 12, pp. 3261-3275.

[Corcoglioniti et al. 2016A] F. Corcoglioniti, M. Rospocher, and A. Palmero Aprosio, “A 2-phase frame-based knowledge extraction framework,” ACM SAC, 2016

[Corcoglioniti et al. 2016B] F. Corcoglioniti, M. Dragoni, M. Rospocher, A.P. Aprosio, Knowledge extraction for information retrieval, in: The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29, - June 2, 2016, Proceedings, 2016, pp. 317–333, doi: 10.1007/ 978- 3- 319- 34129- 3 _ 20.

[Croft 1991] Croft, W.B., Turtle, H.R. and Lewis,D.D. The use of phrases and structured queries in information retrieval.In Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval, (1991), pp. 32-45.

[Curé & Blin 2014] O. Curé and G. Blin. RDF Database Systems: Triples Storage andSPARQL Query Processing, 1st Edition. Morgan Kaufmann, Nov. 2014

[Cyganiak 2005] 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

[Date 2003] C. J. Date. An Introduction to Database Systems. Addison Wesley, eighth edition, 2003.

[Dagan et al. 2013] I. Dagan, D. Roth, M. Sammons, F.M. Zanzotto. 2013.Recognizing Textual Entailment: Models and Applications. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

[Dhall et al. 2016] Dhall, Abhinav & Goecke, Roland & Gedeon, Tom & Sebe, Nicu. (2016). Emotion recognition in the wild. Journal on Multimodal User Interfaces. 10. 10.1007/s12193-016-0213-z.

[Dhall et al. 2017] Abhinav Dhall, Roland Goecke, Shreya Ghosh, Jyoti Joshi, Jesse Hoey, Tom Gedeon. 2017. From Individual to Group-level Emotion Recognition: EmotiW 5.0. In Proceedings of the 19th ACM International Conference on Multimodal Interaction.

[Díaz & López 2019] Noa P. Cruz Díaz, Manuel J. Maña López, Negation and Speculation Detection, John Benjamins Publishing Company 2019

[DuCharme 2013] B. DuCharme. Learning SPARQL, second edition. O'Reilly, 2013.

[Dubois & Prade 2001] D. Dubois, H. Prade, Possibility theory, probability theory and multiple-valued logics: a clarification Ann. Math. Artif. Intell. 32 (1–4) (2001) 35–66.

[Ekbal et al 2010] Asif Ekbal, Eva Sourjikova, Anette Frank, and Simone Paolo Ponzetto. Assessing the challenge of fine-grained named entity recognition and classification. In A. Kumaran and Haizhou Li, Eds., Proc. of the Named Entities Workshop, pages 93–101, Uppsala, Sweden,2010. Association for Computational Linguistics. 25

[Ekman & Friesen 1978] Paul Ekman, Wallace V. Friesen, Facial Action Coding System: Investigator's Guide, 1st ed.; Consulting Psychologists Press: Palo Alto, CA, USA, 1978; pp. 1–15, ISBN 9993626619.

[Ekman & Friesen 2002] Paul Ekman, Wallace V. Friesen, and Joseph C. Hager. Facial Action Coding System: The Manual on CD ROM. A Human Face, Salt Lake City, 2002.

[Elmasri 2010] Elmasri, R., Navathe, S.: Fundamentals of Database Systems. Addison-Wesley, 6th edn. (2010)

[Enger et al. 2017] M. Enger, E. Velldal, L. Øvrelid, An open-source tool for negation detection: a maximum-margin approach, Proceedings of the Workshop Computational Semantics Beyond Events and Roles, SemBEaR 2017, Valencia, Spain

[Etzioni2006] O. Etzioni, M. Banko, and M. J. Cafarella. Machine reading. In AAAI, 2006.

[Etzioni 2011] Etzioni, Oren, e. a. 2011. Open information extraction: The second generation. In *Proc. of IJCAI*.

[Fares et al 2017] Fares, M., Kutuzov, A., Oepen, S., & Velldal, E. (2017). Word vectors, reuse, and replicability: Towards a community repository of large text resources. In Proceedings of the 21st Nordic Conference on Computational Linguistics.

[Gangemi 2017] 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.

[Gionis et al. 1999] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB 2009), pages 518–529. Morgan Kaufmann Publishers Inc.

[Giunchiglia 2009] F. Giunchiglia, U. Kharkevich, and I. Zaihrayeu. Concept search. In ESWC, pages 429-444, 2009.

[Grave et al 2017] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL).

[Griethuysen 1982] Griethuysen JJ van (ed) (1982) Concepts and terminology for the conceptual schema and the information base. ISO TC97/SC5/WG3.

[Grishman & Sundheim 1995] R. Grishman and B. Sundheim. Message understanding conference-6: A brief history. In Proc. of COLING. Association for Computational Linguistics, 1995. DOI: 10.3115/992628.992709. 25, 26, 38

[Groth 2013] A, Groth P, Biemann C, Parreira JX, Aroyo L, Noy N, Welty C, Janowicz K (eds) Proc. 12th International Semantic Web Conference (ISWC 2013), Springer, Sydney, Australia, Lecture Notes in Computer Science, vol 8219, pp 98–113

[Gupta & Malik 2015] S. Gupta and J. Malik, "Visual semantic role labeling" CoRR, vol. abs/1505.04474, 2015.

[Harris 1954] Harris, Z. (1954). Distributional structure. Word, 10(23): 146-162.

[Hellmann 2013] Hellmann S, Lehmann J, Auer S, Brümmer M (2013) Integrating NLP using Linked Data. In: Alani H, Kagal L, Fokoue

[Hill et al 2015] Hill, F., Reichart, R., & Korhonen, A. (2015). SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. Computational Linguistics, 41(4), 665–695.

[Hinton et al,1986] G.E. Hinton, J.L. McClelland, D.E. Rumelhart. Distributed representations. In: Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations, MIT Press, 1986.

[Indyk & Motwani 1998] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In 30th Ann. ACM Symp. on Theory of Computing , 1998.

[Iwasaki 2015] Masajiro Iwasaki. 2015. NGT : Neighborhood Graph and Tree for Indexing. http://research-lab.yahoo.co.jp/software/ngt/.

[Johnson et al. 2015] J. Johnson, R. Krishna, M. Stark, L.J. Li, D.A. Shamma, M. Bernstein, L. Fei-Fei, (2015). Image retrieval using scene graphs. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[Jurafsky & Martin 2008] D. Jurafsky, J. Martin, Speech and Language Processing. 2nd edition. Prentice Hall 2008

[Kamp 1981] 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.

[Kingsbury & Palmer 2002] P. Kingsbury, M. Palmer. From Treebank to PropBank. 2002. In Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002), Las Palmas, Spain.

[Ko et al. 2018] Ko, B. A Brief Review of Facial Emotional expression recognition Based on Visual Information. Sensors 2018, 18, 401.

[Krizhevsky 2012] Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classificationwith deep convolutional neural networks. InNIPS, 2012.

[Kubler 2009] Sandra Kubler, Ryan McDonald, and Joakim Nivre. 2009. Dependency Parsing. Synthesis Lectures on Human Language Technologies. Morgan & Claypool.

[Lashkaria et al. 2019] Fatemeh Lashkaria, Ebrahim Bagherib, Ali A. Ghorbania. Neural Embedding-based Indices for Semantic Search. Elsevier. Information Processing & Management, Volume 56, Issue 3, May 2019, Pages 733-755

[Li 2011] Learning to rank for information retrieval and natural language processing, Synthesis Lectures on Human Language Technologies, Morgan & Claypool, 2011.

[Li 2017] Bofang Li, Tao Liu, Zhe Zhao, Puwei Wang, and Xiaoyong Du. 2017. Neural bag-of-ngrams. Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence

[Liu et al 2017] Liu Jialu, Jingbo Shang and Jiawei Han. “Phrase Mining from Massive Text and Its Applications.” Synthesis Lectures on Data Mining and Knowledge Discovery (2017).

[Liu et al. 2019] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., et al. (2019). Deep learning for generic object detection: A survey. arXiv:1809.02165v2

[Lu et al 2016] C. Lu, R. Krishna, M. Bernstein, and L. Fei-Fei, “Visual relationship detection with language priors,” in European Conference on Computer Vision. Springer, 2016, pp. 852–869.

[MacCartney 2007] Bill MacCartney, Christopher D. Manning. 2007. Natural logic for textual inference. In ACL-07 Workshop on Textual Entailment and Paraphrasing.

[Major et al. 2017] Major, Vincent & Surkis, Alisa & Aphinyanaphongs, Yin. (2017). Utility of general and specific word embeddings for classifying translational stages of research. arXiv:1705.06262

[Manning 2014] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60

[Manning and Schütze 1999] Manning, C. D., & Schütze, H. (1999). Foundations of statistical natural language processing. Cambridge Massachusetts: MIT Press.

[Marcus 1994] Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics, 19:313–330.

[Maynard et al. 2016] Diana Maynard, Kalina Bontcheva, and Isabelle Augenstein. Natural Language Processing for the Semantic Web. Morgan & Claypool, 2016.

[Miller 1995] George A. Miller (1995). WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41.

[Mikolov et al. 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. Distributed representations of words and phrases and their compositionality. In Proc. Advances in Neural Information Processing Systems 26 3111–3119 (2013).

[Mikolov, Chen et al 2013] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.

[Morante et al. 2011] R. Morante, S. Schrauwen, W. Daelemans. 2011. Annotation of negation cues and their scope: Guidelines v1. Computational linguistics and psycholinguistics technical report series, CTRS-003.

[Niklaus 2018] Niklaus C, Cetto M, Freitas A, Handschuh S, A Survey on Open Information Extraction, arXiv:1806.05599

[Niles & Pease 2001] Niles, I., & Pease, A., (2001), Toward a Standard Upper Ontology, in Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Chris Welty and Barry Smith, eds, pp2-9.

[Nivre 2016] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A Multilingual Treebank Collection. In LREC 2016.

[Nouvel et al. 2016] Nouvel, D., Ehrmann, M. & Rosset, S. (2016), Named Entities for Computational Linguistics, John Wiley & Sons

[Olivé 2007] A. Olivé, Conceptual Modeling of Information Systems, Springer, 2007.

[Palmer et al. 2010] M. Palmer, D. Gildea, N. Xue. 2010. Semantic role labeling. Synthesis Lectures on Human Language Technologies.

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

[Pease 2011] Pease, A., (2011). Ontology: A Practical Guide. Articulate Software Press, Angwin, CA. ISBN 978-1-889455-10-5.

[Petasis 2011] Georgios Petasis et al., Ontology Population and Enrichment: State of the Art, Knowledge-Driven Multimedia Information Extraction and Ontology Evolution, LCNS, vol. 6050, Springer, 2011, pp. 134-166.

[Poesio et al. 2016] Poesio, M.; Stuckardt, R.; and Versley, Y. (Eds.). 2016. Anaphora Resolution: Algorithms, Resources, and Evaluation. Springer Verlag.

[Poliak et al 2017] Adam Poliak, Pushpendre Rastogia, M. Patrick Martin, Benjamin Van Durme. 2017. Efficient, compositional, order-sensitive n-gram embeddings. In Proc. EACL

[Porter 1980] M. F. Porter. An algorithm for suffix stripping. Program, 14(3),pages 130–137, 1980. DOI: 10.1108/eb046814. 19

[Reiss 2008] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, S. Vaithyanathan. An algebraic approach to rule-based information extraction.ICDE, 2008.

[Resnik 1997] Resnik, P. 1997. Selectional preference and sense disambiguation. In Proceedings of the ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How? (Washington, D.C.). 52–57.

[Rouces et al. 2017] Rouces, J.; de Melo, G.; Katja, H. FrameBase: Enabling integration of heterogeneous knowledge. Semant. Web 2017, 8, 817–850.

[Russell & Norvig 2010] S. Russell and P. Norvig. Artificial Intelligence: a Modern Approach. Third edition. Prentice Hall, 2010.

[Sahlgren 2008] Sahlgren, M. (2008). The distributional hypothesis. Italian Journal of Linguistics, 20(1):33–54.

[Salton 1968] Salton, G. Automatic Information Organization and Retrieval. McGraw-Hill, New York; 1968.

[Schmitz 2012] M. Schmitz, R. Bart, S. Soderland, O. Etzioni, and others, "Open language learning for information extraction," in Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics, 2012, pp. 523–534.

[Schnabel et al.2015] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In Proc. of EMNLP.

[Schuster 2016] Sebastian Schuster and Christopher D. Manning. 2016. Enhanced English Universal Dependencies: An Improved Representation for Natural Language Understanding Tasks. In LREC 2016.

[Settles 2012] B. Settles. Active learning.Synthesis Lectures onArtificial Intelligence and Machine Learning,6(1):1–114, June 2012

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

[Sugawara et al. 2016] Sugawara, K., Kobayashi, H., Iwasaki, M., 2016. On approximately searching for similar word embeddings. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers.

[Tablan et al 2015] 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.

[Wang et al. 2018] Yanshan Wang, Sijia Liu, Naveed Afzal, Majid Rastegar Mojarad, Liwei Wang, Feichen Shen, Paul Kingsbury, Hongfang Liu. A Comparison of Word Embeddings for the Biomedical Natural Language Processing. arXiv preprint arXiv:1802.00400 [cs.IR], 2018

[Wang et al. 2013] Wang, M., & Manning, C. D. (2013). Effect of Non-linear Deep Architecture in Sequence Labeling. In IJCNLP, pp. 1285 1291.

[White 2018] Lyndon White, Roberto Togneri, Wei Liu, Mohammed Bennamoun, Neural Representations of Natural Language, Springer 2018

[Wimalasuriya 2010] D. Wimalasuriya and D. Dou, "Ontology-Based Information Extraction: An Introduction and a Survey of Current Approaches," J. Information Science, vol. 36, no. 3, 2010, pp. 306–323.

[Wu et al.2015] Yonghui Wu, Jun Xu, Min Jiang, Yaoyun Zhang, Hua Xu. 2015. A study of neural word embeddings for named entity recognition in clinical text. In AMIA Annual Symposium Proceedings, volume 2015.

[Xu et al., 2018A] Hu Xu, Bing Liu, Lei Shu, Philip S. Yu. Lifelong domain word embedding via meta-learning. In Proceedings of the 27th International Joint Conference on Artificial Intelligence. AAAI Press

[Xu et al. 2018B] Hu Xu, Sihong Xie, Lei Shu, and Philip S.Yu. Dual attention network for product compatibility and function satisfiability analysis. In AAAI, 2018.

[Yin and Schutze 2016] Yin, W., and Schutze, H. 2016. Learning meta-embeddings by using ensembles of embedding sets. In Proc. of ACL, 1351–1360.

[Zhang et al 2018] Yaoyun Zhang, Ph.D., Hee-Jin Li, Ph.D., Jingqi Wang, M.S., Trevor Cohen, MBChB, Ph.D., Kirk Roberts, Ph.D., and Hua Xu, Ph.D., Adapting Word Embeddings from Multiple Domains to Symptom Recognition from Psychiatric Notes, AMIA Jt Summits Transl Sci Proc. 2018; 2017: 281–289. Published online 2018 May 18. PMCID: PMC5961810 PMID: 29888086

[Zhu et al 2017] Zhu Y, Yan E, Wang F. Semantic relatedness and similarity of biomedical terms: examining the effects of recency, size, and section of biomedical publications on the performance of word2vec. BMC Med Inform Decis Mak. 2017 Jul 3;17(1):95. doi: 10.1186/s12911-017-0498-1.