Knowledge graph construction for research literatures

Alisson Oldoni

A research project report submitted for
the degree of
Master of Computing and Information Technology

School of Computer Science and Engineering
The University of New South Wales

1 November 2016

Abstract

This research provides a method for extracting information from academic text from the databases domain, using a verb as a query. The amount of latent information in documents in non-structured format or natural language texts is known to be very large, and this is motivation for the development of methods that are able to bring this information into a structured format that can be computationally useful. Most of the academic output is provided in different formats, mostly PDF (Portable Document Format), and contain a very large amount of information and comparison across methods and techniques. We chose to use language models to extract language information, such as part-of-speech tags or dependency trees, and use sets of rules to output a relation in the Relation(Arg_1, Arg_2, Arg_n) format. Our results, for the types of relation we propose to extract, are similar to other existing tools.

Contents

Chapter 1

Introduction

Information extraction (IE) is the process of obtaining in an automatic fashion facts and information from unstructured text that can be read by a machine [26].

Historically, it mostly started with exercises on template filling based on raw natural text [34] as part of the Message Understanding Conferences (MUC) from the late 1980s and 1990s. As part of the MUC, competitions would take place in which a corpus would be made available of a specific domain, and different teams with different programs would try to extract the information from the natural text as to fill in the intended templates.

Note the following text from a news report regarding the result of a soccer match:

> '*Though Brazilian star Diego Tardelli's equaliser denied the Sky Blues victory at Jinan Olympic Sports Centre Stadium on Wednesday night, David Carney banked a precious away goal that will bode well for Graham Arnold's side when they host Shandong in next week's second round-of-16 leg. Sydney FC have taken a sizeable step towards a maiden Asian Champions League quarter-final berth after securing a 1-1 draw with Shandong Luneng in China.*'

```
Team 1:       ____________________
Team 2:       ____________________
Winner:       ____________________
Location:     ____________________
Final Score:  ____________________
```

Figure 1.1: An example of template to be filled in the sports domain.

An example of a task would be, to solely based on the above raw text, to fill in the template shown in Figure 1.1. The MUC competition would be

based in various corpus and tasks based on varieties of news reports, such as satellite launches, plane crashes, joint ventures and other different data in these specific domains.

On the above example, one can observe that the *Team 1* is *Sydney FC*, *Team 2* is *Shandong Luneng*, there was no *Winner*, and consequently the *Location* and the *Final Score*. It gets more interesting as you observe the same type of information being delivered by a different reported:

> '*SYDNEY FC take the advantage of an away goal in China, leaving the second leg of their Asian Champions League Round of 16 tie with a 1-1 draw with Shandong Luneng.*'

Although roughly similar in this case, the approach to retrieve the data from Natural Language text needs to be able to generalise to the various ways a reporter might write such information. This effort becomes more complex as one moves through different domains and audiences of a text, such as: technical manuals, academic papers from different areas, legal text, contracts, financial news, biomedical, among others.

More recently, the output of such Information Extraction systems are used as to build other systems, more prominently Knowledge Graphs. A Knowledge Graph (KG), also known as the knowledge base, is a collection of the machine-readable database that contains entities, the attributes of entities and the relationships between entities [20]. Information Extraction tools would harvest data form unstructured or semi-structured text and provide such databases.

Popular search engines such as Google [20] and Bing [6] leverage Knowledge Graphs as to provide entity summary information and the related entities based on the query that the user is searching for. It is an essential foundation for many applications that requires machine understanding.

The use of Knowledge Graphs then allow users to be able to see extra information in a summarised table-like form, as to resolve their query without having to navigate to other sites. Note in the example in Figure 1.2 how the right column represents a sequence of facts of the *'Jimi Hendrix'* entity, in this case an entity of the class (or type) *PERSON*, such as: his official website; where and when he was born; where and when he died; and a list of movies where this person is the subject of.

Modern pipelines for building Knowledge Graphs from raw text would then encompass several Information Extraction techniques, such as the ones below:

1. Discover entities in the text;
2. Discover relationships between these entities;
3. Perform entity disambiguation

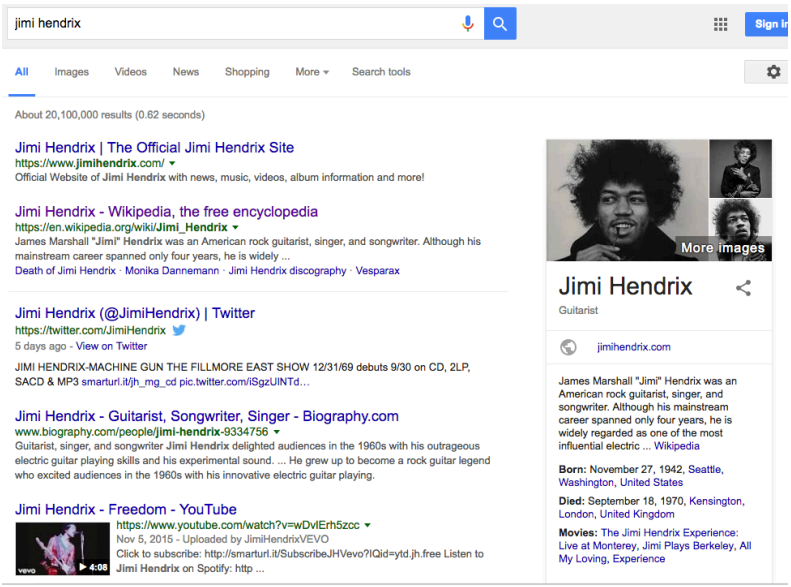


Figure 1.2: An example of knowledge graph application in the Google's result page.

4. Link entities to a reference Knowledge Graph (e.g., Yago2 [44] or DBPedia [27]).

5. Improve the quality of the output via input data cleaning, robust extraction, and learning-based post-processing methods;

6. Reason about how accurate these facts are;

7. Finally presenting the facts in a graph (the Knowledge Graph).

Some of these techniques will be explained further as part of this document.

In this project, we focus on studying and presenting some Information Extraction techniques as to build a domain-specific verb-centric information extraction tool that extracts relations from academic papers. More specifically, we focus on papers from the topic of databases and attempt to extract information from these papers for posterior usage by other systems with applications such as:

- Allow for structured and fast search of techniques in the papers and the possible relations between them;

- Possibly group papers by their used techniques;

- Discovery of techniques to improve performance on a certain problem;

- Generate a hierarchy of concepts, and their use;
- Among others.

An existing service that organises data from academic papers is the Semantic Scholar [40] project, by Professor Oren Etzioni from Allen Institute for AI. However, Semantic Scholar only understands a limited number of relationships (such as 'cite', 'comment', 'use_data_set', and 'has_caption') which are also more closely related to the meta-data about the paper, but not from the knowledge that the paper itself presents. Other similar services are Microsoft Academic Graph [32], Google Scholar [21], and CiteSeerX [11].

In the next sections, this document will give some background information on the techniques needed to achieve the above (Chapter 2), and it will also define the problem more precisely (Chapter 3), building as to introduce the development of this research (Chapter 4). In Chapter 5 we will describe some of the results, followed by the some final remarks in Chapter 6.

Chapter 2

Information Extraction

Information Extraction, a term already defined in the introduction, is a hard problem which mostly relies in attempting to use a computer to understand information explicitly stated in the form of natural language.

It is interesting to observe that, before writing down thoughts in a paper, an academic form the ideas of what facts he/she wants to express in his/hers head, and then attempt a structure to most clearly state these in text form. These multiple facts, and the relations between them, are then stated in a sentence in what is assumed to be a somewhat logical format, following the semantics of the language, whether English or any other. Following this example, one must then also assume that the future reader of this paper will use the reverse process to decode this information into facts or ideas to be understood. In fact, this assumption is what justifies the attempt of Information Extraction.

Several initiatives in the Natural Language Processing area attempt to understand and map what these semantic rules are, and how one could use a computer for tackling natural language related tasks. These initiatives are fruitful and provide advanced tools and techniques in which some will be described in this chapter.

2.1 Natural Language Processing

Natural Language Processing (or *NLP*) can be a term used to discuss any kind of computer manipulation of natural language text, also called raw text. It can mean simple things such as counting words and obtain their frequency distribution to compare different writing styles, or in a more complex sense it would require the understanding of human writings, to the extent of being able to extract information and meaning from it, or also give useful responses to them [7]. On this more complex end of the spectrum, where one wants to understand raw text, language technology and existing tools rely on formal models, or representations, of knowledge of language at the levels of

morphology, syntax, semantics, among others linguistic concepts. A number of formal models including state machines, formal rule systems, logic, and probabilistic models are used to capture this knowledge from text and reason with it [26].

When information is laid out in natural language form, one start the analysis of the information presented by constructing a phrase or sentence based on smaller pieces of information such as verbs, nouns, and adjectives which are called the constituents. These then build up to form sequences of simple and complex sentences.

Observing more carefully a simple sentence, such as *'The police chased him.'*, it is possible to attempt to sample the different syntactic information presented in it As a first step it is possible to dissect its constituent parts as per Figure 2.1.

```
The/DT police/NN chased/VBD him/PRP ./.
```

Figure 2.1: An example of tagged sentence.

The first word in the sentence, *The*, is a *DT* or determiner. Other possible determiners include *'my'*, *'your'*, *'his'*, *'her'*. The second word is *'police'* is a *NN* which is the tag for a singular noun. With this information we can already tell that this sentence is speaking about something, and this something is the noun *'police'*. Subsequently the tag *VBD* is presented which specifically indicates the word *'chased'* is a verb (an action) in the past tense. At this point one can observe that something or someone (in this case the *'police'*) did something in the past.

This is already great information to have about the sentence. These tags that were added to the text in Figure 2.1 are called Part-Of-Speech tags, or POS tags [26]. The standardization of these tags and work to develop and tag existing text with them is done by the Penn Treebank project [30].

Note how specific the tags are, dictating the type of the word, a verb for an example, and its variation either in quantity or tense. Some other examples of these tags are shown in Table 2.1. Although most of them are simple to understand, note that *ADP* are the adpositions, which encompasses prepositions and postpositions. Some systems, such as the spaCy Natural Language Processing parser [24, 42] also maps these more specific tags into more general ones, for an example, while three different words in a sentence are tagged independently as *VBD*, *VBG* and *VBZ*, they are also tagged with a *VERB* tag. This is useful, if the user is not too interested in the detail of which verb variation was used.

A Part-Of-Speech Tagger is then a system that, given a raw text as input, assigns parts of speech to each word (or token) and is able to produce as output the tagged version of this text. The text in Figure 2.1 was tagged

POS tag	**Meaning**	**Sample**
ADP	Adpositions	*at*, or *in*
CONJ	Coordinating conjunction	*and*, or *or*
DT	Determiner	The
JJ	Adjective	She is *tired*
JJR	Adjective, comparative	That one is *larger*
JJS	Adjective, superlative	That is the *largest*
NN	Noun, singular or mass	Car
NNS	Noun, plural	Cars
NNP	Proper noun, singular	Microsoft
NNPS	Proper noun, plural	The *Kennedys*
RB	Adverb	She said *firmly*
VB	Verb, base form	Attack
VBD	Verb, past tense	Attacked
VBG	Verb, gerund or present participle	Attacking
VBN	Verb, past participle	Broken
VBP	Verb, non-3rd person singular present	I *attack*
VBZ	Verb, 3rd person singular present	He *attacks*

Table 2.1: List of some of the possible Part-Of-Speech (POS) tags.

using the Stanford Log-linear Part-Of-Speech Tagger [47].

POS Tag	**Word**	**Prev. Word**	**Prev. Tag**
DT	The	<START>	<START>
NN	police	The	DT
VBD	chased	police	NN
PRP	him	chased	VBD
.	.	him	PRP

Table 2.2: Features for sequential POS tagging.

The task of assigning these tags starts by deciding what are the tokens in a raw text sentence, and what are its sentences. As an example, the tokenizer needs to decide if a period symbol near a word represents an abbreviation (e.g. *Dr.*) or a sentence boundary - in case of an abbreviation, this period is then considered simply a token within the sentence. Another common problem in this step is deciding if a single quote is part of a word,(e.g. *'It's'*), or is delimiting a quoted part of the sentence, thus potentially hinting other semantic meanings. The Stanford POS Tagger used in this example also contain a tokenizer, which is part of the Stanford CoreNLP [29], a set of natural language analysis tools.

Modern POS Taggers tackle this task using a technique called Sequence Classification. A machine learning classifier model is then trained with a cor-

pus of manually tagged text and has as an input certain features that might indicate which tag a token being currently analysed should be assigned with. Observing again the example from Figure 2.1, but now presented in table format in 2.2, it is easier to see how this learning algorithm would be trained to predict tags on unseen test. Note the second word *'police'*. For this word we are providing 4 features for the learning algorithm: the manually labelled POS Tag, the word itself, the previous word and the previous POS Tag. Suppose now that this sentence in included in a bigger corpus, and this pattern is a common one and the learning algorithm is provided with a substantial amount of labelled data in which this situation repeats itself: a *NN* is the second word in a sentence, with *'The'* and *DT* being the previous word and tag respectively.

After the training, this model would behave in a similar fashion once presented with unseen data. Suppose now the first column on Table 2.2 is not presented. The model would pick the first word *'The'* and observe the features: $<START>$ and $<START>$ respectively and given that in our previously described corpus this is a common occurrence, it would then label this word with *DT*. Now, for the second word *'police'*, the features would be *'The'* for previous word, and *DT* for previous tag. Again, it is common for a noun to be placed after a determiner so the model assigns the label *NN* to the word *'police'*. The features used by the Sequence Classifies both while training and when using the model may vary and they impact the quality of the predictions it makes. The Stanford POS Tagger uses a broad use of lexical features, including jointly conditioning on multiple consecutive words [47].

A helpful concept at this stage is known as the lemma. A lemma is a canonical way of representing a word which strips out variations for quantity, tense, among others [26]. For an example, given the words *'running'* or *'runs'*, NLTK [7] outputs *run* as their lemma. The lemma can, for an example, be used together or instead of the existing features for training models.

```
Jetstar/NNP Airways/NNPS ,/PUNC a/DET unit/NN of/ADP Qantas/NNP
                    Airways/NNP Limited/NNP
```

Figure 2.2: An example of another tagged sentence.

When reading a text, it is also important to understand the relation between the words or sub-sentences that are contained in the phrase, and this is specially useful for information extraction. As an example, suppose the sentence *'Jetstar Airways, a unit of Qantas Airways Limited'*. There are different ways of breaking down the relationship of the words in this sentence.

Starting again with the POS tagging discussed earlier, one can see in

2.2 what are the types of the words that constitutes this sentence. As a further step, it's possible to see what are the sub-sentences or parts that form the phrase structure, also called constituency parsing. The sub-sentences are connected upwards to one head (also called, *parent*), and downwards to one or more governors (also called *dependants*, or *children*), in a recursive structure. In Figure 2.3, the phrase *'of Qantas Airways Limited'*, which is part of the bigger phrase we are using as an example, is a prepositional phrase. A prepositional phrase lacks either a verb or a subject, and serve to assert binary relations between their heads and the constituents to which they are attached, in this case *'a unit'* [26]. The sub-sentence *'a unit of Qantas Airways Limited'* is then a noun phrase, since it contains and talks about a noun *'unit'*.

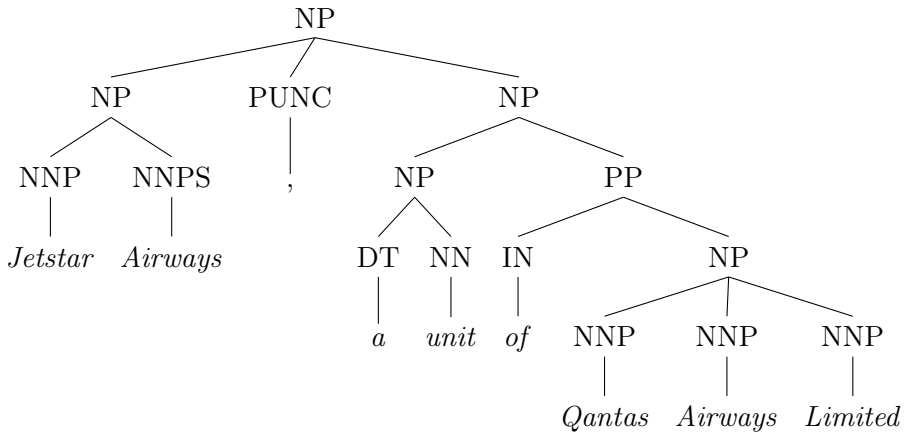


Figure 2.3: A sentence broken down to its Phrase Structure (sub-sentences), also known as constituency parsing.

After discovering the structure between the phrases and its sub-phrases, finding the syntactical dependency between words themselves is also a very interesting and useful task. Continuing on the same example, one can see how the sentence states the simple fact that one company (*Jetstar*) is a unit of another company (*Qantas*). The Dependency Tree in this case tells us that *Jetstar/NN* is the head of another noun *Airways* and that the relation between them is of the *compound* type. This relation is held between any noun that serves to modify the head noun and in this case indicate that this single entity is formed by two different words that are nouns. Note that *Jetstar* has no parents and thus is the root of the sentence. The dependencies can be fully viewed is Figure 2.4.

The next relation is the *appos* from *'Airways'* to *'unit'*. The appositional modifier relation indicates that the noun immediately to the right of the first noun that serves to define or modify the meaning of the former. One now already knows that *'Jetstar Airways'* is a *'unit'*, and, in the busi-

ness context, it probably means that it is a company that belongs to a bigger company.

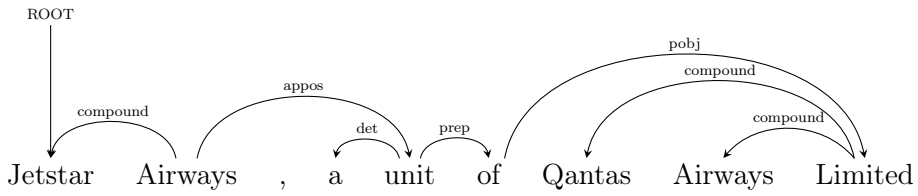


Figure 2.4: A sentence and the dependencies between the words.

Continuing the analysis, the next relation is of the *prep* type and it indicates a prepositional modifier of a verb, adjective, or noun (our case), and it serves to modify the meaning of the verb, adjective, noun, or even another preposition. Note that this relation simply indicates the word pointed to by the edge is the preposition (in this case *'of'*). The next relation, *pobj*, indicates the actual object of the preposition and the noun phrase following the preposition and what it related to. This tree was created by the spaCy dependency parser [24, 42]. This same tree can be visualised in a more traditional tree structure in Figure 2.5. Although not in this example, another very common relation is the relative clause modifier *recmod* (or *relcl* in some notations) relation. A relative clause modifier of an NP (*Noun Phrase*) is a relative clause modifying the NP. The relation then in the tree points from the head noun of the NP to the head of the relative clause, normally a verb. The explanations for the cited relations in this document were obtained from the Stanford typed dependencies manual [31], and the Universal Dependencies (UD) project [37], both which are reference for the possible relation and contain a full lists of their meanings.

The software that is able to output a syntactical dependency tree, given a sentence, is called a dependency parser. Several different methods can be used to achieve this. One way is by defining dependency grammars, and then parsing text using these grammar. The grammar would contain words and its possible heads, and it would be applied repeatedly into the text in a process called cascaded chunking [7]. More recent methods use a process called Shift-reduce, in which the sentence is kept in a queue with the left-most token in front of the queue. The model could then decide between applying 3 operations:

1. Shift: move one token from the queue to stack.

2. Reduce left: top word on stack is head of second word.

3. Reduce right: second word on stack is head of top word.

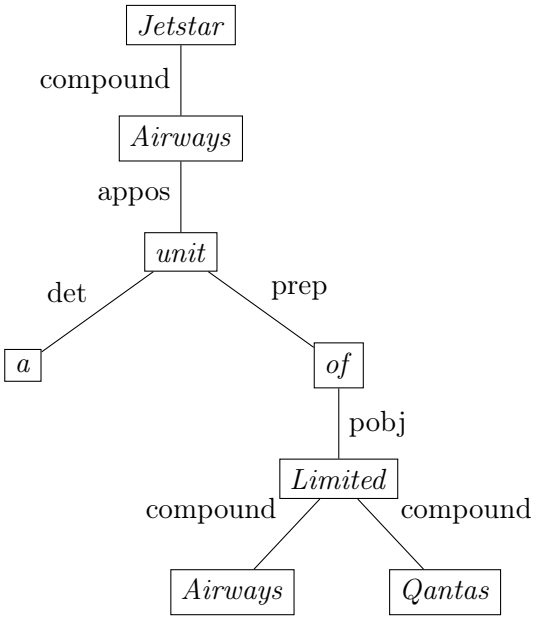


Figure 2.5: A sentence and the dependency tree, showing the syntactical relation between these words.

A model is then trained to predict, given a text that is added to the queue, what is the next move that it should take, and what is the sequence of moves that will result in the best possible final dependency tree. This is done in a monotonic manner, in the sense that once a decision is made by the model, it cannot change it. Full working examples of this method are described in [10]. Other methods also use these 3 possible decisions, but allow the parser to be non-monotonic and go back in the tree and change previous decisions given new evidence form the features, such as spaCy and its dependency parser [24, 42]. Another method also uses the same set of decisions, but does a beam-search observing multiple partial hypotheses and keeping them at each step, with hypotheses only being discarded when there are several other higher-ranked hypotheses under consideration, such as the Syntaxnet parser [46, 3]. All these cited models use neural networks as the method to form the model.

As a further example, besides providing dependency parsing tree, spaCy also provides an iterator so you can obtain what is called the noun chunks of the document. The noun chunks are smaller pieces of the sentences within the document that are base noun phrases, or a 'NP chunk' (as per Figure 2.3). They are noun phrases that do not permit other NPs to be nested within it so no NP-level coordination (e.g.: *'cat/NN and/CONJ dog/NN'*), no prepositional phrases, and no relative clauses [42].

Other problems tackled by the Natural Language Processing discipline

and that are relevant for Information Extraction are *Coreference resolution* and *pronominal anaphora resolution*. Coreference resolution intends to define all possible entities that a text can reference to in some sort of definitive list, or more precisely a *discourse model*, find in the text all the chained references to these entities, and link them to the specific entities. While very similar in nature, pronominal anaphora resolution is more simple as it is the problem of resolving in a given sentence to which previous NN (noun) or NNP (proper noun) a single PRP (pronoun) refers to [26].

Take for an example the sentence *'John is a quiet guy, but today he is furious.'*, the initial mention of the entity *John* appears in the first token. Token five talks about a *guy*, which although not a pronoun, is still a reference to the previous entity in the first word. The ninth token is a pronoun and again refers to the same *John*, so is part of the chain of mentions. The full resolution chain would be denoted in Figure 2.6.

Normally these systems work towards analysing pairs of tokens using a probabilistic model, and then decide how likely they are references for the same entity. More recent approaches also group possible tokens in a cluster and use cluster-level features to determine the chains of coreferences [12]. The tool used to extract Figure 2.6 is the Stanford Coreference Resolution annotator [12], which is part of the latter group of tools that use cluster level features. This annotator is also part of the CoreNLP toolset [29].

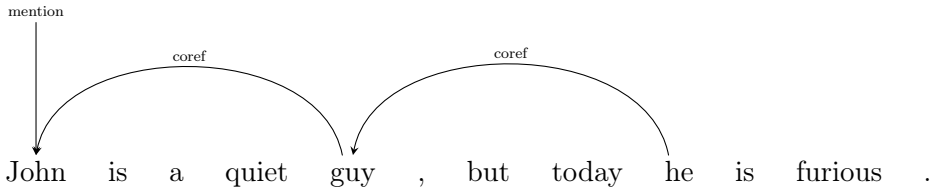


Figure 2.6: A sentence, the mentions of an entity, and the proposed coreference resolutions.

All linguistic data mentioned in this section is data that can be obtained from the raw text itself, and is then the base for several features in which methods for Information Extraction act on.

2.2 Information Extraction

The IE (Information Extraction) process is described by the following subtasks: Named Entity Recognition (NER), Coreference Resolution, Entity Disambiguation, Relation Extraction (RE), Event Detection, and Temporal Analysis [26]. The main subtasks relevant to this report will be described further in this section.

Once the information is extracted it is then used for tasks such as Template Filling [26], Question and Answering systems [34], or stored as a Knowledge Graph for downstream logical reasoning or for further queries.

[PER James Cook] was born on 27 October 1728 in the village of [LOC Marton] in [COUNTY Yorkshire].

Table 2.3: An example of Named Entity Recognition (NER).

Named Entity Recognition (NER) is the process of, given a sentence, identify and extract what are the entities that are part of it. Once the entity is detected, it needs to be classified within the classes of the given domain - in the spirit of the previous examples this would be e.g.: *CITY* or *PERSON*. Different types of entities are relevant to context of the data being worked with. A few approaches exist for the problem of NER, mostly related to Pattern Matching or Sequence Classification.

Pattern	**Would yield ENTITY of type**
[PERSON] was born	PERSON
in the village of [LOC]	LOCATION
in [LOC]	LOCATION

Table 2.4: Examples of Named Entity Recognition (NER) patterns, based on the sentence from Table 2.3.

Observe, for an example, the sentence in Table 2.3. Several articles regarding prominent figures, either historical or of our current society, can be of the format *'Jimi Hendrix was born'*. One approach might be Pattern Matching, which is to mine the input natural language text while looking for the pattern *'[ENTITY] was born'*, using Regular Expressions (Finite-State Automata) [26]. The entities found by this pattern would then also receive the *PERSON* class. This pattern would miss the sentence *'Jimi Hendrix, born in Seattle'* since it does not fit the pattern and, because of this, one generally needs to build a list or database of patterns to work with in a corpus. An example of such database was generated by the PATTY system [36]. Table 2.4 depicts other possible similar patterns.

Another way to extract entities from text is to frame the NER problem as a Sequence Classification problem, similar to the POS tagging problem described earlier. It requires the training of a classifier in which, given the class of the previous word, and other surrounding features of the current word, will attempt to guess if the current word is an entity, and if it is, also guesses its class.

To achieve this, previously annotated data with existing sentences and its entities is needed. This can be obtained by manually labelling data, or by semi-automated methods, like the one proposed later by this document.

The format in which this annotated data is provided varies, however the IOB format (Table 2.5) is more commonly used in several of the NER tools, including NLTK [7] and the popular Stanford Named Entity Recognizer (NER) [19], part of the Stanford CoreNLP [29]. Stanford CoreNLP provides a set of natural language analysis and information extraction tools.

Word	**Tag**
James	B-PERSON
Cook	I-PERSON
was	O
born	O
on	O
27	B-DATE
October	I-DATE
1728	I-DATE
in	O
the	O
village	O
of	O
Marton	B-LOC
in	O
Yorkshire	B-LOC
.	O

Table 2.5: Example of IOB-formatted sentence used to train classifiers for the Named Entity Recognition (NER) task, based on the sentence from Table 2.3.

The IOB format also helps remove ambiguity in case there are two contiguous entities of same class without any word tagged as *O* in between. In practice these cases are somewhat rare in several domains, and even when trained with such tags classifiers struggle to accurately determine the boundaries of an entities, and thus a simplified version of this annotation without the *B-* and *I-* prefixes is more commonly used [45].

The Stanford Named Entity Recognizer (NER), also known as CRFClassifier [19], provides a general implementation of (arbitrary order) linear chain Conditional Random Field (CRF) sequence models. A CRF is a conditional sequence model which represents the probability of a hidden state sequence given some observations.

Several relevant features can be used as an input during the training of a NER CRF classifier model. In Table 2.6, examples are presented. The Word Shape feature is an interesting addition from recent research, as it captures the notion that most entities are written in capital letters, or starting with capital letter, or containing numbers in the middle of the word, and

other specific shapes.

Feature	Description
Word	The current word being classified.
N-grams	A feature from n-grams, i.e., sub-strings of the word.
Previous Class	The class of the immediate previous word.
Previous Word	The previous word.
Disjunctive	Disjunctions of words anywhere in the left or right.
Word Shape	The shape of the word being processed captured using. In general replaces numbers with d, x to lowercase letters, and X to upper-case letters.

Table 2.6: Examples of features used to train the CRFClassifier [19].

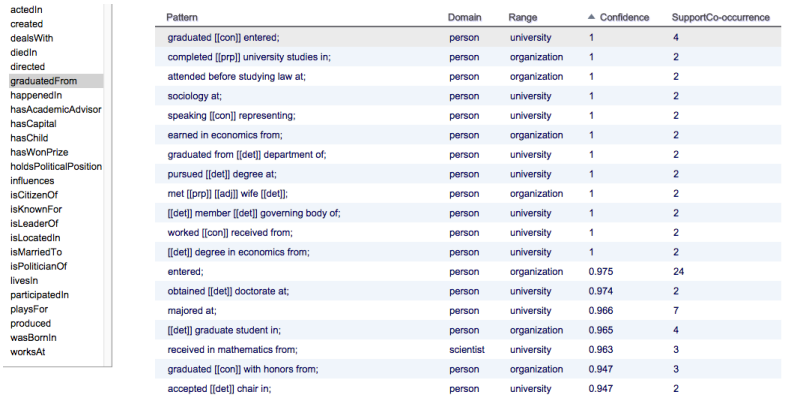
In addition to the above methods another useful technique is the use of gazetteers. Gazetteers are common for geographical data, where government provided lists of names can contain millions of entries names for all manner of locations along with detailed geographical, geologic and political information [26].

Relation Extraction (RE) is the ability to discern the relationships that exist among the entities detected in a text [26], and is naturally the next challenge after being able to detect entities. It is generally denoted as a triplet: two entities, and the one relation between them (Table 2.7). It can be done using Pattern Matching, Classifiers, or purely by exploiting linguistic data available form a sentence. The previously described Pattern Matching technique from NER can be improved upon in the Relation Extraction step, and involve more than one entity, yielding binary relations. This approach is used in tools such as PROSPERA [35] or those mined by PATTY [36]. More specifically, examples of patterns mined by PATTY for the *graduatedFrom* relation are seen in Figure 2.7.

```
Located_In(Kiel, Germany)
```

Table 2.7: An example of a triplet that represents a relation.

PROSPERA's main technique is that not only it obtain facts based on a small set of initial seed patterns, but also obtain new candidate patterns that can be extrapolated from the corpus based on the mined known facts. Once the process of obtaining new candidate patterns finishes, these are evaluated and then added to the the existing pattern repository for re-use. The whole process then iterates again finding even more facts from these new patterns, and new candidate patterns [35]. Moreover, another interesting characteristic of the PROSPERA's approach is the care in Entity Disambiguation. For an example, given that in a text it finds a name, such as *'Captain James Cook'*. It then uses a knowledge base such as YAGO [44]



actedIn
created
dealsWith
diedIn
directed
graduatedFrom
happenedIn
hasAcademicAdvisor
hasCapital
hasChild
hasWonPrize
holdsPoliticalPosition
influences
isCitizenOf
isKnownFor
isLeaderOf
isLocatedIn
isMarriedTo
isPoliticianOf
livesIn
participatedIn
playsFor
produced
wasBornIn
worksAt

Pattern	Domain	Range	▲ Confidence	Support	Co-occurrence
graduated [[con]] entered;	person	university	1	4	
completed [[prp]] university studies in;	person	organization	1	2	
attended before studying law at;	person	organization	1	2	
sociology at;	person	university	1	2	
speaking [[con]] representing;	person	university	1	2	
earned in economics from;	person	organization	1	2	
graduated from [[det]] department of;	person	university	1	2	
pursued [[det]] degree at;	person	university	1	2	
met [[prp]] [[adj]] wife [[det]];	person	organization	1	2	
[[det]] member [[det]] governing body of;	person	university	1	2	
worked [[con]] received from;	person	university	1	2	
[[det]] degree in economics from;	person	university	1	2	
entered;	person	organization	0.975	24	
obtained [[det]] doctorate at;	person	university	0.974	2	
majored at;	person	university	0.966	7	
[[det]] graduate student in;	person	organization	0.965	4	
received in mathematics from;	scientist	university	0.963	3	
graduated [[con]] with honors from;	person	organization	0.947	3	
accepted [[det]] chair in;	person	university	0.947	2	

Figure 2.7: An example of patterns extracted from PATTY for the *graduatedFrom* relation.

to compare this with existing known entities, using techniques such as N-gram comparison [35]. With such effort, PROSPERA is able to know that *Captain James Cook* and *James Cook* are actually the same entity with a certain confidence, and thus don't differentiate these and assigns *'Captain James Cook'* as a representation of the canonical unambiguous entity *'James Cook'*. This helps in several ways, such as: it can then know other information about this entity, such as the fact that it is of the class *PERSON*; and it can also facilitate future queries in this knowledge base, centralizing the new information found about this existing entity.

Another tool in the Stanford CoreNLP package, the Relation Extractor [45] is a classifier to predict relations in sentences. This program has a model that extracts binary relations between entity mentions in the same sentence. The output is normally in the XML [18] format and denotes the tokens of each sentence, the possible relations, and the confidence level of these relations. The XML demonstrated in Figure 2.8 depicts a guess that 2 words in the sentence *'..., including approaches that use parallel computation [1, 2, 6, 13, 24].'* have the *Uses* relation with a confidence above 70%. The classifier in this case also indicates that one of the entities is of the class *CONCEPT*.

As part of its training, annotated relation mentions are used as an input together with the text it belongs to, and the annotation becomes a positive example for the corresponding label, while all the other possible combinations between entity mentions in the same sentence become negative examples. The feature set models the relation between the arguments by using the distance between the relation arguments, the syntactic path between the two arguments, using both constituency and dependency representations.

Note how at this point it's important to no note the notion of a *pipeline* of natural language processing tasks. Stanford's CoreNLP is able to, by itself, perform all the steps needed to produce such relation extraction output

```
<relation id="RelationMention-6728">Uses
  <arguments>
    <entity id="EntityMention-1324">O
      <span start="46" end="47"/>
      <probabilities/>
    </entity>
    <entity id="EntityMention-1319">CONCEPT
      <span start="36" end="37"/>
      <probabilities/>
    </entity>
  </arguments>
  <probabilities>
    <probability>
      <label>Uses</label>
      <value>0.7379587661527921</value>
    </probability>
    <probability>
      <label>Improves</label>
      <value>0.06475441029357998</value>
    </probability>
  </probabilities>
</relation>
```

Figure 2.8: An example of relation extracted with the Stanford Relation Extractor that demonstrated the *'Uses'* relation.

from the raw text input. The steps of this pipeline are executed in a certain order, as they depend on the previous step (e.g.: POS tagging is needed for Dependency Parsing). In this case, the process was:

1. Tokenize;
2. Sentence Splitter;
3. Part-of-Speech tagging;
4. Lemmatization;
5. Constituency parsing;
6. Dependency Parsing;
7. Named Entity Recognition;
8. and finally, Relation Extraction.

The Stanford Relation Extractor comes with a model that was trained to extract the following relations: *Live_In*, *Located_In*, *OrgBased_In*, *Work_For* - and the following classes: *PERSON*, *ORGANIZATION*, *LOCATION*. There are big challenges if one attempts to train the model for any relation outside

of these, mainly in obtaining or generating annotated data to train the classifier as to generate a useful model. There are attempts in which relation extraction is not based on annotated data, but on linguistic characteristics of the text itself, such as its semantics. These tools are normally called Open Relation Extractors and will be further described in Section 3.2.

2.3 Knowledge Graphs

Knowledge Graphs contain a valuable of information in a structured format, traditionally originally mined from table-like structures form places like Wikipedia [50] tables [27], or from processes like Information Extraction as described in the previous section. It can be used for a diverse range of applications, such as helping other systems reason about quality of harvested facts [44], provide table-like facts about an entity [20], and question-answering systems [22]. Moreover, recent years have witnessed a surge in large scale knowledge graphs, such as DBpedia [27], Freebase [8], Googles Knowledge Graph [20], and YAGO [44].

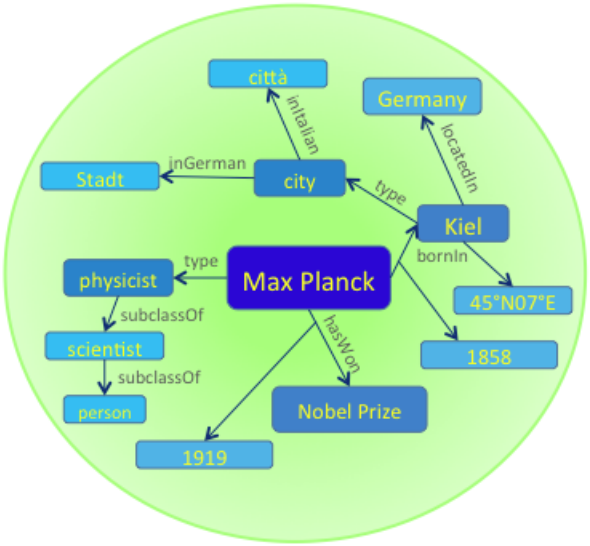


Figure 2.9: An example of knowledge graph from [44] plotted with vertices and edges.

The Knowledge Graph name follows from the data structure that is created from the facts in its final form, a graph with nodes representing entities and edges representing various relations between entities. In Figure 2.9, it is possible to observe an example plotted in this form. The list of possible entities classes, and allowable relations between entities is known as a schema. The schema represented in Figure 2.9 is detailed in Table 2.9; one can observe that, as an example, *'Max Planck'* is an entity of the type *physicist*.

```
type(A, D) :- type(A, B), subclassOf(B, C), subclassOf(C, D)
```

Table 2.8: This entailment example allows one to assert that `type(Max Planck, person)` is also true, based on the fact tuples presented in Table 2.9.

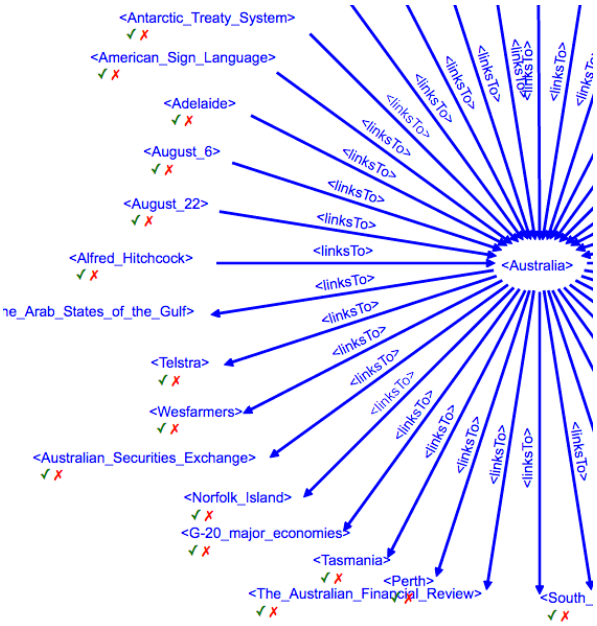


Figure 2.10: An example of patterns existants in YAGO.

Moreover, based on the facts presented, entailments can be made and one trivial example is denoted in Table 2.8. More complex examples of possible reasoning can be seen in [45]. This is equivalent to traversing the graph from a node that represents a more specific information, to a node that represents a more general information - e.g.: another possible child node of *'scientist'* could be the type *'biologist'*.

```
type(Max Planck, physicist)
subclassOf(physicist, scientist)
subclassOf(scientist, person)
bornIn(Max Planck, Kiel, 1858)
type(Kiel, city)
locatedIn(Kiel, Germany)
hasWon(Max Planck, Nobel Prize, 1919)
```

Table 2.9: Some facts regarding Max Planck, also depicted in Figure 2.9.

This example denotes a classical domain, more precisely important per-

sons, companies, locations, and the relations between them, in which Information Extraction (IE) tools have been very successful on.

As mentioned previously, YAGO [44] is a prominent Knowledge Graph database, and possesses several advanced characteristics. Every relation in its database is annotated with its confidence value. See the example of the resulting graph in Figure 2.10. Moreover, YAGO combines the provided taxonomy with WordNet [33] and with the Wikipedia category system [50], assigning the entities to more than 350,000 classes. This allow for very powerful querying. Finally, it attaches a temporal and a spacial dimension to many of its facts and entities, being then capable to answer questions such as *when* and *where* such event took place.

WordNet is a semantically-oriented dictionary of English, similar to a traditional thesaurus but with a richer structure [7]. More specifically, it provides relations to synonyms, hypernyms and hyponyms, among others.

Chapter 3

Analysis and Related Work

This work intends to deliver a tool or a process in which one can extract information from academic text, more specifically Computer Science papers form the Database and Data Mining topics. The intention is to obtain entities, and relations between these entities. The motivation is that, with such tool, one could for an example:

- Historically research algorithms that were mostly used during a certain time period;
- Find which algorithms are used to resolve, or related to, a certain problem;
- Find techniques that improve a certain algorithm problem, among others.

3.1 Analysis of Academic Text

The corpus of text used was generated utilising papers published from the following conferences during various years: ACL [49], EMNLP [16], ICDE [14], SIGMOD [1], VLDB [48]. More specifically, the section of *Related Work of* these papers were the ones used to build the corpus. This was done due to the characteristics and patterns of this section compared to the rest of the paper. After careful reading, we observed that the *Related Work* section generally contains objective comparisons between other algorithms or softwares in contrast with more opaque or abstract explanations form other parts of the paper. This would mean that this section was a good candidate to start the analysis from. Note the following examples of sentences from the *Related Work* section of papers from the corpus:

1. *'Bergsma et al (2013) show that large-scale clustering of user names improves gender, ethnicity and location classification on Twitter.'*

2. *'N-Best ROVER (Stolcke et al, 2000) improves the original method by combining multiple alternatives from each combined system.'*

3. *'By partitioning the velocity space, the Bdual -tree improves the query performance of the B x -tree.'*

Entities from academic text in this setting are not as straightforward to define as in, for an example, business news, or criminal news. Observe the following sentence:

- *'Japan's Toshiba Corp said it had nominated Satoshi Tsunakawa, a former head of its medical equipment division, to be its next chief executive officer.'*

Text	**Entity Type**
Japan	LOCATION
Toshiba Corp	ORGANIZATION
Satoshi Tsunakawa	PERSON

Table 3.1: Examples of Named Entity Recognition (NER) from the business news text example.

From the news text example above, Table 3.1 lists the entities that are clearly notes in the text. One can observe a very strong feature which is the common capitalization of the first letter of each of these entities. Another characteristic is how entities from this news text example are *global* or *unconditional*: *'Japan'* is a location regardless of any condition or any context in this document. Another observation is that, referring to the Stanford's Relation Extraction default relations, *'Toshiba Corp'* is an organisation *Located_In 'Japan'* regardless of other context in this document. This contrasts with concepts and their relations observed in academic papers, thus that while *'large-scale clustering'* has the *Improves* with *'gender classification'* in the context of the paper where this data is presented, it might not be true in all cases.

Text	**Entity Type**
Bergsma et al (2013)	AUTHOR
large-scale clustering	CONCEPT
gender classification	CONCEPT
ethnicity classification	CONCEPT
location classification	CONCEPT

Table 3.2: Examples of Named Entity Recognition (NER) from the academic text example.

Moreover, the entities in Table 3.2 are harder to classify in universally agreed classes. For an example, *'gender classification'* can be considered an action, or a task, or an algorithm. More generally, one can simply classify these as concepts.

```
IsA(Concept,Concept)
SimilarTo(Concept,Concept)
Improves(Concept,Concept)
Employs(Concept,Concept)
Uses(Concept,Concept)
Supports(Concept,Concept)
Proposes(Author,ComplexConcept)
Introduces(Author,ComplexConcept)
```

Table 3.3: Some observed and possible relations between concepts.

Other possible relations from the Stanford Relation Extractor standard relations that are applicable to the above news text example are: *OrgBased_In* (again for *'Toshiba Corp'* and *'Japan'*) and *Work_For* regarding its newly placed chief executive officer. Again, contrasting with the academic text, one might consider relations such as the one possibles between concepts as denoted in Table 3.3. In fact, by analysing the corpus for the top 50 words in the singular third-person form, such as *'improves'* or *'employs'*, one can have an idea of the possible relations that can be extracted. This process is illustrated in Figure 3.1: note that the top two words were removed from the graph (*'is'* with a count of 41694, and *'has'* with a count of 8157) as their usage counts are too high compared to the other words.

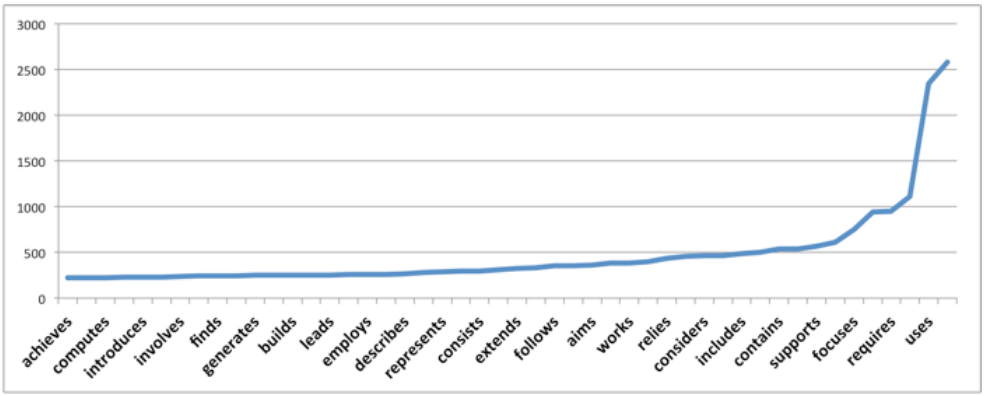


Figure 3.1: Samples of the most common words in the singular third-person form, after removing the top 2 words (*'is'* and *'has'*). The y axis represents the number of times the word in the x axis appeared in the text.

One of the initial attempts to explore how to extract information from the generated corpus was to use the Stanford Named Entity Recognizer (NER) to recognize the concepts discussed so far in the academic text. To do so, a small set of around 20 papers' *Related Work* section was annotated for the concepts contained in them using Brat [43]. An example of this annotated data can be seen in Figure 4.1.

The annotated data is then transformed form Brat's standoff format [43] into a Table Separated Value (TSV) format, using a custom script, based on customised version of from standoff2conll[1], renamed standoff2others. The output is similar to the one showed in Table 2.5, but its simplified version without the *B-* and *I-* prefixes.

The model was trained mostly with the recommended settings and features, such as the word itself, its class, surrounding words and word shapes. When applying this trained NER model (Figure 3.2), we observed that the success was moderated, as it was, at times, able to detect clearly delineated concepts by its shape (.e.g: capital words), but for non-capitalized words it appeared as it would only recognize the concepts if its words were present in the training set.

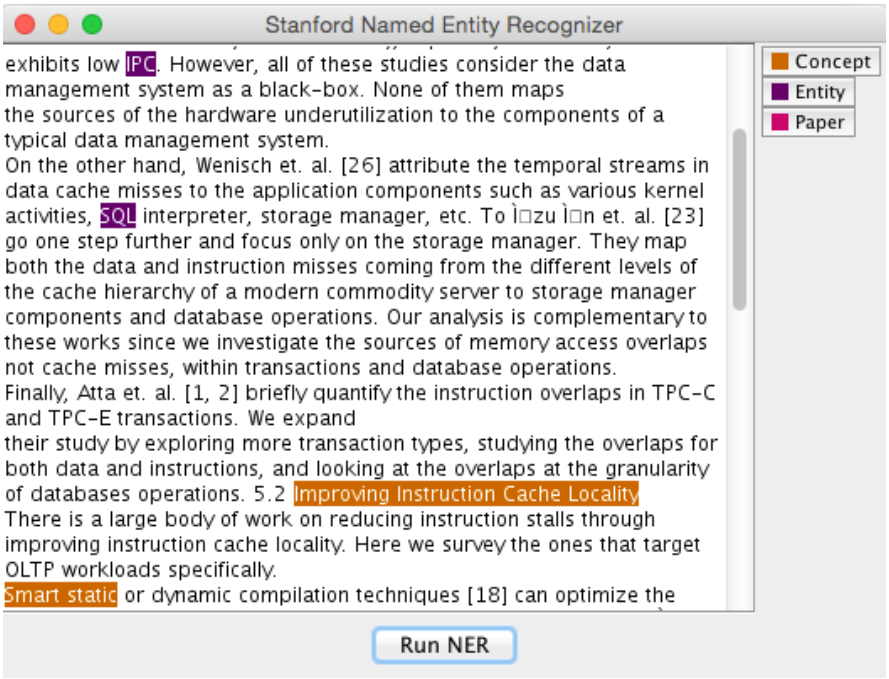


Figure 3.2: The Stanford NER GUI (Graphic User Interface) using our trained model.

[1] https://github.com/spyysalo/standoff2conll

In this image, please observer the attempt of differentiate entities such as *CONCEPT* and *ENTITY*. We also annotated references to other papers using the *PAPER* entity, in general they appears as numbers between square brackets. Initially, we attempted to annotated using a hierarchy where entities were very specific proper nouns, while concepts had a more loose definition, and would likely be more general concepts. During the process, however, this type of annotation also proved to be difficult as it would require domain-specific knowledge of very deep database discussions in order to differentiate concepts by these two classes, and could still sometimes generate debates.

As an attempt of further improve the quality of the NER model, we made use of a gazetteer. As part of this research, the Microsoft Academic Graph [32] was found to contain a very relevant list of *keywords* and *fields of study* available for download and academic use. Another custom script was developed to transform the data from the format provided by Microsoft into the input format accepted by Stanford's NER shown in Table 3.4. The Stanford NER utilises the gazetteer input in both ways: matching the concepts token by token in their entirety, or in a 'sloppy' manner accepting a positive match even if only one of the tokens in the gazetteer entry had a match [19]. In both cases, however, the gazetteer is treated simply as another feature and does not guarantee that if the entries are found in the text they would be marked as an entity [19]. The gazetteer format has its first token denoting the type of the entry, all of the type *CONCEPT* in this case, with the following words denoting the gazetteer entry itself, space separated. We did not observed improvement with this addition.

```
CONCEPT SMOOTHSORT
CONCEPT CUSTOMISED APPLICATIONS FOR MOBILE NETWORKS
CONCEPT XML DOCUMENTS
CONCEPT JOSEPHUS PROBLEM
CONCEPT RECOGNIZABLE
```

Table 3.4: Format in which Stanford's NER supports a gazetteer input.

The next step was to attempt to use the Stanford Relation Extractor (RE). The same small annotated sample by us in Brat would also contain the following relations: *Improves*, *Worsen*, *IsA*, *Uses*. The standoff2others custom library was then improved to be able to generate the more complex CoNLL format, accepted as an input for training of the Stanford's RE, denoted in Table 3.5 [45]. Also, the Java parser code from the Relation Extractor had to be changed in a few places to accept custom labels (classes) for NER.

The important columns of this format are: column 2 which denotes the entity tag, column 3 denotes the token ID in the sentence, column 5 contains

2	Concept	0	O	NNP/NNS	LSH/functions	O	O	O
2	O	1	O	VBP	are	O	O	O
2	O	2	O	NFP	fi	O	O	O
2	O	3	O	RB	rst	O	O	O
2	O	4	O	VBN	introduced	O	O	O
2	O	5	O	IN	for	O	O	O
2	O	6	O	NN	use	O	O	O
2	O	7	O	IN	in	O	O	O
2	Concept	8	O	NNP/NN	Hamming/space	O	O	O
2	O	9	O	IN	by	O	O	O
2	O	10	O	NNP	Indyk	O	O	O
2	O	11	O	CC	and	O	O	O
2	O	12	O	NNP	Motwani	O	O	O
2	O	13	O	-LRB-	[	O	O	O
2	O	14	O	CD	7	O	O	O
2	O	15	O	-RRB-	]	O	O	O
2	O	16	O	.	.	O	O	O
0	8	Uses						

Table 3.5: Format in which Stanford's Relation Extractor accepts its training input.

its Part-Of-Speech tag, and column 6 which contains the token itself. For this specific process, POS tags were obtained from the Google Syntaxnet Software [46, 3], which were generated in separated and then joined with the token for the final CoNLL output.

The results from the RE trained model, one of which is depicted in Figure 2.8, were much poorer compared to the NER output, and we failed to find interesting relations with confidences above 50%. In both cases, after analysing the models we were able to generate using the NER and the Relation Extractor software from Stanford, it was clear that much more annotated data would be needed as to achieve higher quality results.

Please refer to Section 4.1 for more information on tools mentioned in this section.

3.2 Open Information Extraction

Since we had no access to annotated data, we turned to a different approach called Open Information Extraction in an attempt for better results. This approach uses linguistic information from the text, among other techniques, as to attempt to extract the relations without the need of labelled data for a trained model.

Text	Extracted relation
We stress that our method improves a supervised baseline.	*improves*(our method ; supervised baseline)
(2008) demonstrate that adding part-of-speech tags to frequency counts substantially improves performance.	-
Experiments with an arc-standard parser showed that our method effectively improves parsing performance and we achieved the best accuracy for single-model transition-based parser.	*achieved*(we ; best accuracy for single-model transition-based parser) *is with* (Experiments , arc-standard parser)
(2007) revealed that adding non-minimal rules improves translation quality in this setting.	*adding*(translation quality ; rules) *is in*(translation quality ; setting)
(CBS Detroit, 2011-02-11) improves substantially over prior approaches.	*improves over*(CBS Detroit ; approaches) *improves substantially over*(CBS Detroit ; prior approaches) *improves over*(CBS Detroit ; prior approaches) *improves substantially over*(CBS Detroit ; approaches)

Table 3.6: Examples of results from the Open Information Extraction software from Stanford, Stanford OpenIE.

Stanford's OpenIE [4] is the first of these tools which we experimented with and works by utilising two classifiers, both applied on linguistic information from the text. The first one works at the text level and attempts to predict how to yield self-contained sentences from the text. As it processes the text, this classifier decides on three possible action: yield, which outputs a new sentence; recurse, which navigates further in the dependency tree arcs for the actual subject of the sentence; or stop, which decides then not to recurse further.

Comparison type	OpenIE sample parameters	NER sample output	Result
At least 1 full match	Exists(Entity One ; Entity Two Three)	Entity One	True
At least 1 full match	Exists(Entity One ; Entity Two Three)	Entity Four	False
At most 2-grams	Exists(Entity One ; Entity Two Three)	Entity Two	True
At most 2-grams	Exists(Entity One ; Entity Two Three)	Two	False
Exact match, both equal	Exists(Entity One ; Entity Two Three)	Entity Two Three	True
Exact match, both equal	Exists(Entity One ; Entity Two Three)	Two	False
1-gram	Exists(Entity One ; Entity Two Three)	Entity Two Three	True
1-gram	Exists(Entity One ; Entity Two Three)	Two	True
1-gram	Exists(Entity One ; Entity Two Three)	Four	False

Table 3.7: List of heuristics attempted when trying to combine OpenIE with NER results. Note that the *At least 1* comparison type is the only one that accepts that yields true by matching only 1 of the OpenIE parameters, all others are comparing both OpenIE parameters against NER resulted entities.

Once these sub-sentences are decided upon, its linguistic patterns are then further used to help a second classifier which will decide the format of the relation to be returned. It tries to yield the minimal meaningful patterns, or relation triplets, by carefully deciding with arcs to delete from the dependency tree, and which arcs are useful.

In some experiments, we observed that when applied to academic text, in the context of searching for the *Improves* relation (see Table 3.3 for a proposal of possible useful relations to be extracted), OpenIE can end up observing the pattern but not including in its output, or including it in a non-canonical form. For an example, Table 3.6 shows the output for a small range of sentences. Row 1 of this table shows a correct extraction, while row 2 shows a similar sentence that however yielded no result. Rows 3 and 4 present a situation where the *Improves* relation could be observed but it is not extracted, while row 5 shows a situation where this relation is present, but its non-canonical form is extracted with some other variations. Regarding this presented data, one observation is that OpenIE does not know what the researcher is after when extracting information from the text. While this might be interesting in several cases (i.e.: in early iterations with a corpus, as to observe what are the kinds of relations one could possibly find), the tool might not include relevant results once a specific type of relation is being sought after.

Comparison type	**Result**
At least 1	*is in*(Several research projects ; databases) *focuses on*(IVM ; xed query) *is in*(IVM ; DBToaster) *has*(IVM ; has developed) *aggressively pre-processing*(IVM ; query) *computing query over*(we ; database)
At most 2-grams	*utilizing constraints in*(IVM ; IVM) *hash*(k ; functions) *focuses on*(Association Queries Prior work ; association queries) *deploy*(RDF data ; own storage subsystem tailored to RDF) *using*(String Transformation ; Examples) *combining*(Samples ; samples)
Exact match	N/A
1-gram	*are*(Spatial kNN ; important queries worth of further studying) *are*(graph databases ; suitable for number of graph processing applications on non-changing static graphs) *have*(several graph algorithms ; With increase in graph size have proposed in literature) *compute*(I/O efficient algorithm ; components in graph) *builds on*(Leopard 's light-weight dynamic graph ; work on light-weight partitioners) *are related to*(Package queries ; queries)

Table 3.8: Sample of results of combining output from OpenIE with NER.

Further exploring OpenIE's potential, an experiment we did was to

attempt N-gram matching with the OpenIE results as to cross-analyse its output with the NER results from the model trained, as explained in Section 3.1. More precisely, given the relation and its 2 parameters extracted from the text, what are the relations in the output from Stanford OpenIE in which the parameters match an recognized entity from the output of Stanford NER. The types of comparisons done are depicted in Table 3.7 and some selected results are in Table 3.8. In general, we found this approach to yield only a very small number of the possible results, while also presenting inconsistencies (too much variation) in regards to the types of relations obtained.

As a similar tool, ClausIE, a Clause-Based Open Information Extraction [15] from the Max-Planck-Institute runs the sentences through a dependency parser, and use rules in order to find relations from constituents. ClauseIE starts finding clauses (candidate relations) by searching for subject dependencies (*nsubj*, *csubj*, *nsubjpass*, *csubjpass*), and then parse the entire sentence to get the contents of this relation. More precisely, in this final process it then attempts to detect the type of the sentence based on a sequence of decisions, as to match a known type of sentence. These types of phrases take into consideration all dependencies of the constituents of this clause, and then classify them as, e.g.:

- **SV**: Subject and Verb, such as: *Albert Eistein died*;
- **SVA**: Subject, Verb and Adverb, such as: *AE remained in Princeton*;
- **SVO**: Subject, Verb and Direct Object, such as: *AE has won the Nobel Prize*;
- Among others.

With all this information at hand, it then yields relations by deciding the combinations of constituents that will form a relation. An on-line demo[2] exists in which its capabilities can be observed.

In contrast, AllenAI's OpenIE [17] utilises the text linguistic information in a different manner. As a first step it apply a Part-of-Speech tagging in the text and the NP-chunks of the sentence are then obtained through constituency parsing, both process are done using the Apache OpenNLP parser [5]. I then then utilises regular expressions on the result as to restrict the patterns to be treated. More specifically, it obtain the relations through searches for clauses in the format $V \mid VP \mid VW^*P$, where V is a verb or adverb, W is a noun, adjective, adverb, pronoun or determiner, and P is a preposition, particle or information marker. Once the clause is identified it uses a custom classifier called ARGLEARNER to find its arguments *Arg1* and *Arg2* and the left bound and the right bound of each argument.

[2] https://gate.d5.mpi-inf.mpg.de/ClausIEGate/ClausIEGate

Some approaches rely on human intervention as to control the quality of the extracted relations, or to guide the types of relations needed. Extreme Extraction [23] provides an interface where one can narrow sentences for a given relation; provides suggestions for words surrounded by similar context; and allows for extraction rules creation using logic entailments. AllenAI's IKE [13] is also a tool of similar nature, and provides its own query language which resembles regular expressions which apply at the Part-of-Speech level, or NP-chunks. It also provides powerful suggestions using probabilistic techniques as to narrow rules that are too general, or broadens rules that are too specific. IKE also provides a way to define a schema to store the items found by the rules from its query language for faster reuse as smaller parts of more complex conditions.

All tools described in this section are similar in nature to our tool, thus describing the related work.

3.3 Peculiarities of Academic Text

It was clear that existing model-based tools for IE, such as the ones shown in Section 2.2, do not come equipped to predict relation in Academic Text, mainly due to the different classes of entities presented. Academic text, however, has some characteristics that facilitate in some sense its parsing. More specifically, the language used in academia is more strict and precise, and does not contain attempts of inventive language or linguistic creative which would be common in romances or other type of written information such as literary books. We also did not observe academic text present difficult to understand notions, such as sarcasm or humour. We then attempt to remove complexity further by narrowing our scope to papers from the database area, such as noted in Section 3.1.

During our experimentation with manually tagging data, described in Section 2.2, this text also presented itself very difficult to tag by humans, as it would sometimes require domain knowledge on very advanced or narrow areas of the database topic.

The text also present a high number of coreference problems, e.g.: *'their work'*, *'the technique explained in [X]'*, or simply *'[X] facilitates this by using a certain algorithm'*. In these previous examples X would represent a number, generally a reference to another academic paper. One technique to alleviate these problems could be to parse the above references numbers and replace with the paper name or technique names from the papers.

Chapter 4

Developed Workflow

We developed an open information extraction tool that is capable of extracting information from academic text with the following characteristics:

- A verb-centric search query (normally a verb in the 3rd person singular);
- Exploiting linguistic properties of the text by obtaining this text Part-of-Speech tags and Dependency Tree using SpaCy [24, 42];
- Caching techniques for the parsed values for faster future re-runs, or iterations in adjusting rules in case one wants to customize the code further.
- Some other minor adjustments in the text parsing, away from defaults, to improve performance and quality.
- Local optimisations and adjustments within the tree from the verbs perspective as to prepare the relations for the output. More specifically, by local we mean near the nodes in the tree in which the verb is found to be in.
- Ability to export triplets for monotransitive verbs, or simpler relations for intransitive verbs, with optional parameters.
- The output of the relations in an HTML [25], and graphical way for easy grouping and visualisation using Graphviz[1].
- Or the output in a machine-readable way, the JSON [9] format.

With these characteristics, the tool is able to extract information from text for both analysis and further fine-tuning by a competent Python developer, or for down-the-line processing by another software.

[1] http://www.graphviz.org/

Our corpus was generated using an extraction process developed by Haojun Ma and Wei Wang at the University of New South Wales [28], which uses the pdftohtml[2] tool. All academic papers were downloaded into individual file-system directories, generally in the PDF [2] format. In each folder the conversion occurs by detecting the PDF file's layout and further detection and extraction of the 'Related Works' section. Files are then centralized in a single folder for parsing.

4.1 Tools

4.1.1 Programming Languages and Libraries

Java[3] is a programming language used in several of NLP tools, such as Stanford's CoreNLP [29] and Apache OpenNLP [5]. Java is an imperative, static-typed, compiled language and provides several utilities such a comprehensive standard library and strong Unicode[4] support.

Python[5] on the other hand is a scripting language that has been recently associated with Data Analysis[6] due to its powerful built-in idioms for data processing and its clean syntax. Although not as fast as a compiled language, it has the ability to have more low-level extensions through tools such as Cython[7], which is used for an example by the SpaCy [24, 42] parser.

Due to familiarity and above points, we have chosen utilising SpaCy and Python (version 3.4) to develop this tool. With that some modules (external libraries, or external dependencies) from the Python ecosystem that were used were: requests[8] and BeautifulSoup4[9] for downloading data from an Web Server; standoff2conll[10] for experiments with Brat and Stanford's Relation Extractor data transformation; and corpkit[11] for some text queries like concordance (e.g.: other words that appear in the same context, or surrounded by similar words) and lemma-based search instead of token-based during an early exploratory stage of this research. Note that corpkit utilises corenlp-xml[12] as a way to parse Stanfords CoreNLP output in Python. For generating HTML output we used Django[13] template engine in standalone mode.

[2] http://www.foolabs.com/xpdf/download.html
[3] https://docs.oracle.com/javase/specs/
[4] http://unicode.org/standard/standard.html
[5] https://docs.python.org/3.4/reference/
[6] https://www.quora.com/Why-is-Python-a-language-of-choice-for-data-scientists
[7] http://cython.org/
[8] http://docs.python-requests.org/en/master/
[9] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[10] https://github.com/spyysalo/standoff2conll
[11] https://interrogator.github.io/corpkit/
[12] https://github.com/relwell/corenlp-xml-lib
[13] https://www.djangoproject.com/

4.1.2 Stanford CoreNLP

Stanford CoreNLP [29] is an integrated framework of linguistic tools written in Java. As discussed and presented in previous section in more detail (Sections 2.1 and 2.1 and 2.2), it is done in this way with the intent of facilitate the creation of pipelines in which more fundamental. In the CoreNLP each of these tools are called annotators. It provides the following annotators out of the box: Tokenization; Sentence Splitting; Lemmatization; Parts of Speech; Named Entity Recognition (described further in this document in Section 2.1); RegexNER (Named Entity Recognition); Constituency Parsing; Dependency Parsing (also in Section 2.1); Coreference Resolution; Natural Logic; Open Information Extraction (Section 3.2); Sentiment; Relation Extraction (Section 2.2); Quote Annotator; CleanXML Annotator; True case Annotator; Entity Mentions Annotator.

4.1.3 NLTK

NLTK [7] is a popular Python toolkit, or set of libraries for NLP, generally associated with its companion book and popular in introductory NLP courses. NLTK provides interfaces to over 50 corpora and lexical resources such as WordNet [33], along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Stemming is a concept related to simplifying the handling of variations of words, such as plural or past tenses, in a more simple way of lemmatization. In stemming the root (or certain prefix range) of a word is kept while its varying part is removed.

NLTK also provides implementation of classification algorithms that can be trained for further text classification, and grammar parsers that can be defined and used, for example, to return a NP-chunking tree. It also has interfaces to the Stanford CoreNLP pipeline, so it can be used to externalise operations to it. While heavily used in an interactive manner together with Jupyter[14] in early exploratory stages of this research, only parts of its tree data structure remain used in some stages of the final developed workflow of this work.

4.1.4 Syntaxnet

The Syntaxnet parser [46] is a Tensorflow[15] implementation of the models described in [3]. TensorFlow is an Open Source Software Library for Machine Intelligence developed by Google. Syntaxnet parses a document or text feed through the standard input and outputs the annotated text in the

[14] https://jupyter.org/about.html
[15] https://www.tensorflow.org/

CoNLL format (see sample in Table 3.5), accepted as an input for training of the Stanford's RE.

In this project, Syntaxnet was used in the experiments described in Section 2.2 when adding Part-of-Speech tags for the Stanford Relation Extractor training input.

4.1.5 SpaCy

SpaCy [42] is Python/Cython NLP parser that provides Tokenizing, Sentence Segmentation, Part-of-Speech tagging and Dependency Parsing [24]. Although this encompasses less functionality in comparison with CoreNLP, the processing is done in a very fast manner, and conveniently into the Python language. SpaCy features a whole-document design: where CoreNLP for an example relies on sentence detection/segmentation as a pre-process step in the pipeline, spaCy reads the whole document at once and provides Object-oriented interfaces for reaching the data. A web interface DisplaCy[16] is also available for more impromptu checks on its dependency parser output.

More specifically, the hierarchy of Object-oriented classes are:

- **English**: the class that loads the language model for further parsing.
- **Doc**: it accepts a document as it is input, parses it, and then provides iterators for sentences, and tokens.
- **Span**: A group of tokens, e.g.: a sentence, or a noun-chunk.
- **Token**: A token. It contains its raw value, position in the document and in the sentence, POS tag information at different granularities (Table 2.1), and position in the dependency tree.

4.1.6 Brat

Brat [43] is a web-based tool, written in Python, for text annotation. It provides a method for define possible annotations, number of parameters, and possible relatinos between these annotation. Once this is defined, the interface allows for text selection and point-and-click, drag-and-drop interfaces to facilitate such annotation process. See Figure 4.1 for an example of annotated text.

Since Brat is designed in particular for structured annotation, the text or data observed is in free form text, but will then have a rigid structure for future machine interpretation. As noted in Section 2.2, we developed a data transformation tool from the standoff format used by Brat named standoff2others, extending the existing standoff2conll[17] library.

[16] https://demos.explosion.ai/displacy/
[17] https://github.com/spyysalo/standoff2conll

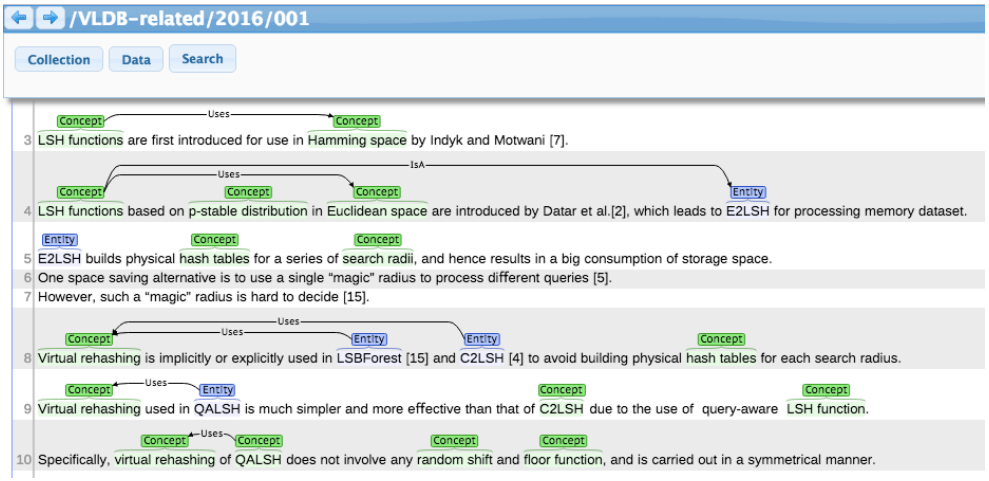


Figure 4.1: The Brat rapid annotation tool, an online environment for collaborative text annotation.

4.1.7 Graphviz

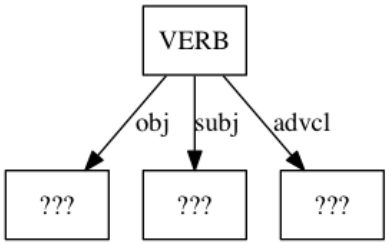


Figure 4.2: The resulting graphic from graphviz from the Figure 4.3 DOT specification.

Graphviz[18] is open source graph generation software. It utilises a graph describing language called DOT[19] which is used to generate the graphs using the distributed DOT binary that companions the Graphviz distribution.

In this project, we used its Python wrapper[20], which allowed for seamless generation of the DOT file straight from Python objects. As an example, Figure 4.3 shows the specification in DOT language that generated the graphic image in Figure 4.2

[18] http://www.graphviz.org/
[19] http://www.graphviz.org/doc/info/lang.html
[20] https://pypi.python.org/pypi/graphviz

```
digraph uses {
    node [shape=box]
    1376 [label=VERB]
    1378 [label="???"]
        1376 -> 1378 [label=obj]
    1379 [label="???"]
        1376 -> 1379 [label=subj]
    1377 [label="???"]
        1376 -> 1377 [label=advcl]
}
```

Figure 4.3: The DOT language.

4.2 Developed Program

Our developed program, namely *corpus_ analysis.py*, accepts as an input a file-system folder of raw text and a verb in its singular third person form, and it then outputs relations of the parameters surrounding that verb in the sentence, aiming for a triplet relation type such as the format: `Relation ( Argument1 ; Argument2 )`. Figure 4.4 presents the HTML output of the program rendered in a Web Browser.

To some extent, our current approach is similar to that of ClauseIE, in the sense that it completely relies on the dependency parsing tree, but with several key differences:

- Due to this verb-centric nature, since the verb is the relation being searched for and is part of the input, our tool tailors the extraction process for each different verb. It does so in the sense that ignores parts of the corpus that does not contain the token we are searching for, as long as there is a sufficiently large corpus to find typical usage of the verbs;
- Instead of heuristically determining whether or which PP-attachment, (named 'A' as in the *SVA* sentence type) to be used as object of the verb, we can do it more accurately given that the verb is given as an input. The same is true for 'O' in the *SVO* sentence type;
- Also, we extract more than binary relations with an optional typeless argument as ClauseIE did. The output is more in line with these semantic functional analysis of verbs, as in PropBank [38], or VerNet [39];
- ClauseIE classifies the sentence being extracted against a list of possible sentences using a decision tree, and then uses this information to

decide how to extract the information. In contrast, our method simply applies a sequence of rules in an arbitrary order that attempts to reach out for information in case it is missing in the nodes near the position of the verb in the dependency tree;

- Finally, it utilises SpaCy for dependency parsing instead of Stanford CoreNLP. This might affect technological choices as this process could more easily fit into a Python-based pipeline.

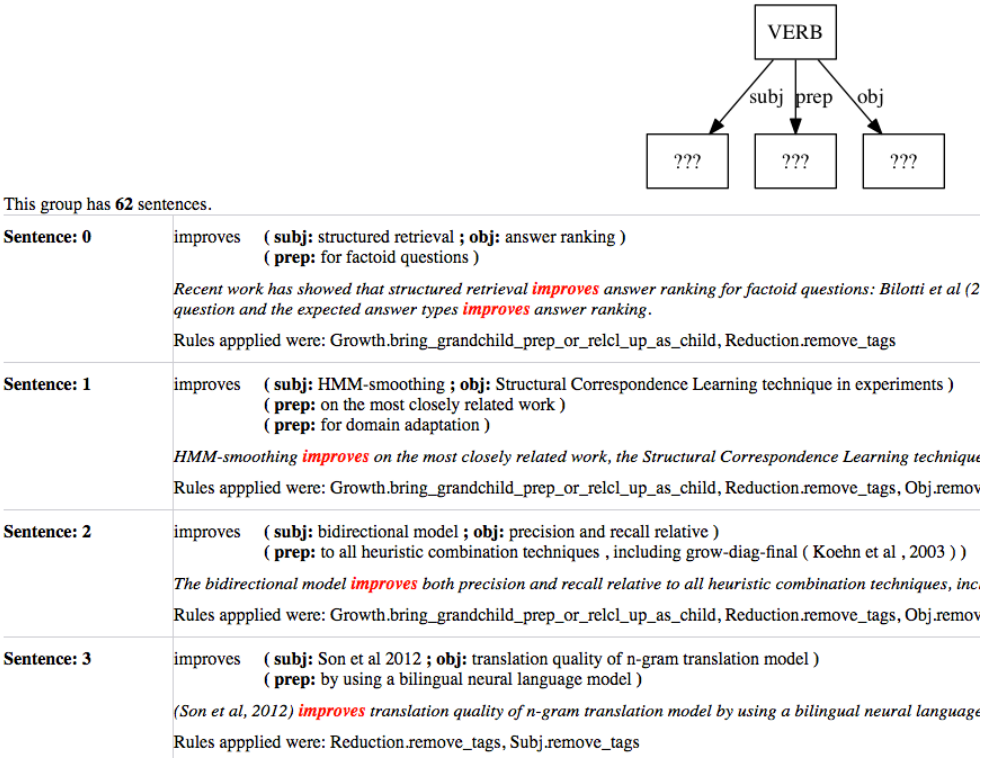


Figure 4.4: The HTML output generated by our program. Note how it organizes sentences orders by its grouping.

Contrary to other tools, such as Stanford OpenIE, we are extracting only explicit information. There is no logical reasoning to better present the information obtained and that is implicit in the text. Moreover, we do not try to match the results of our tool against any knowledge database with the intention to compare what is being learned from the natural text with.

The main algorithm of our tool is simple in the sense that the goal is to process all entries found in the text containing the verb being looked after. Note that, in Algorithm 1, `token` is actually a node in the dependency tree, thus why rules are applied directly into the `token` variable. The main loop

Algorithm 1 Main loop

```
1: procedure SimplifiedGroup(files, verb)
2:     for sentence, token in GetTokens(files, verb) do
3:         ApplyGrowthRules(token)
4:         ApplyReductionRules(token)
5:         ApplyObjRules(token)
6:         ApplySubjRules(token)
7:         relations ← Extraction(token)
8:         AddToGroup(sentence, token, relations)
9:     end for
10:    GenerateOutput(groups)
11: end procedure
```

Algorithm 2 Iterator to tokens and sentences

```
1: procedure GetTokens(files, verb)
2:     finalList ← GetFromCache(files)
3:     if NOT finalList then
4:         list ← GetFilesWithVerb(files, verb)
5:         finalList ← []                          ▷ Empty list
6:         for text in list do
7:             rawParsed ← ParseRawText(text)
8:             spacyParsed ← EnglishSpacyModel(rawParsed)
9:             cacheableTreeNode ← TranformTree(spacyParsed)
10:            AppendToList(finalList, cacheableTreeNode)
11:        end for
12:        SaveCache(finalList)
13:    end if
14:    Yield each token, sentence from finalList
15: end procedure
```

then extract the relations and prepare the grouping presentation. After all this is done, the output is then generated. The goal of the HTML output is for human evaluation and analysis, while the goal of the JSON output is for down-the-line processing by other program.

Algorithm 2 implements a Python iterator[21] using the `yield` keyword. It starts by attempting to find a copy of the parsed tree already cached for performance purposes. If a cached version exists, it is used instead. Caching was implemented as follows: A cache entry has a key, which combines the verb being searched for and the date in which the input folder containing the raw text was last modified. This means that a cache entry can only be

[21] https://docs.python.org/3.4/reference/expressions.html#generator-iterator-methods

Algorithm 3 Accumulate sentences

```
1: procedure AddToGroup(sentence, token, relations)
2:     groupRepr ← GroupQTREERepr(token)    ▷ Group representation
3:     GenerateSentenceImage(sentence)
4:     GenerateGroupImage(groupRepr)
5:     if groupRepr not in groups then
6:         AppendToGroup(groups, groupRepr)
7:     end if
8:     AppendToGroup(groups[groupRepr], sentence, relations)
9: end procedure
```

found if the folder was not modified and the verb being searched for now was already searched for before. We had to implement our own tree data structure that mimics the SpaCy data structure since the SpaCy tree was not serializable with Pickle[22], the Python library responsible for serialization. In line 9 of Algorithm 2 we can see that the `cacheableTreeNode` variable is the same as the `token` variable, which is yielded later on by the function.

```
u [ a [ b c d]
    e [f [g h] i]
    k
  ]
}
```

Figure 4.5: A tree denoted using QTREE.

Algorithm 3 depicts the grouping of sentences by representation in a dictionary, which is the Python equivalent to a hash-table. In line 2, one can see the GroupQTREERepr method which, given the token data structure, generated the QTREE [41] representation of it for grouping purposes. Note that this is not the full tree, but only a smaller version used for analysis as described in Section 4.3. The QTREE representation was chosen to be the canonical representation of the tree data structure, and is then used as the key of the dictionary. This is the result of a performance optimisation on earlier versions of the tool, which instead compared the tree which already existing entry in a list before deciding if it already existed in it or not. This brings this part of the process asymptotically from $\mathcal{O}(n)$ time to a much faster in practice constant $\mathcal{O}(1)$ time.

In QTREE one uses the square brackets symbol to denote the edges and the hierarchy of the tree in text mode, resulting in a string representation of it. See an example in Figure 4.5.

[22] https://docs.python.org/3.4/library/pickle.html

Some specific adjustments were added to the procedure ParseRawText (in line 7 of Algorithm 2), as follows:

- Applies a regular expression to replace all *'et al.'* strings with an empty string. This is done to improve sentence segmentation in SpaCy which was in several occasions specifically confusing the term with a sentence boundary;
- Another small tweak was done in the SpaCy tokenizer as to not split words that contain a dash in the middle, such as *'data-mining'*. A file called `infix.txt`[23], which is part of the SpaCy data, contains a set of regular expressions for the tokenizer, and the `(?<=[a-zA-Z])-(?=[a-zA-z])` responsible for tokenizing words with a dash symbol was then deleted. This change made the tree simpler in some situations by reducing the amount of punctuation nodes;
- Removed unicode characters from the output by using Python filters to achieve so;
- Added a regular expression to remove citations of the type *'(Lenat, 1995)'*. This avoids SpaCy breaking these as nodes in the tree and diminishes the chances of misclassifications of the dependencies.

4.3 Grouping Sentence Types

This section further describes the purpose of the GroupQTREERepr method from Algorithm 3. The grouping of the sentences was done with the goal to facilitate human local analysis. There are four possibilities for grouping: based on the *verb* node (the original verb in which the relation is being searched for), based on the *subj* node, based on the *obj* node, and based on any other of the optional relations nodes.

The grouping works as follows. Given the node the grouping is based on, the immediate children are extracted and a new tree is formed only with the node plus its children. For an example, given the sentence *' MACK uses articulated graphical embodiment with ability to gesture.'*. Suppose we are searching for the *Uses* relation. The dependency tree from SpaCy is generated and presented in Figure 4.6. The `token` being analysed by the algorithm is then the word *uses*, at the top of the tree. The tree has four child nodes, however we disregard the actual child nodes values, and pay attention

[23] More precisely, when using virtualenv, it sits in in the following location: `.env/lib/python3.4/site-packages/spacy/data/en-1.1.0/tokenizer/infix.txt`. Virtualenv is a method of installing Python packages only to the local scope of a project, without affecting the traditional global folder where packages are installed (which affect all Python programs in the computer) - more information about virtualenv can be found at https://virtualenv.pypa.io/en/stable/.

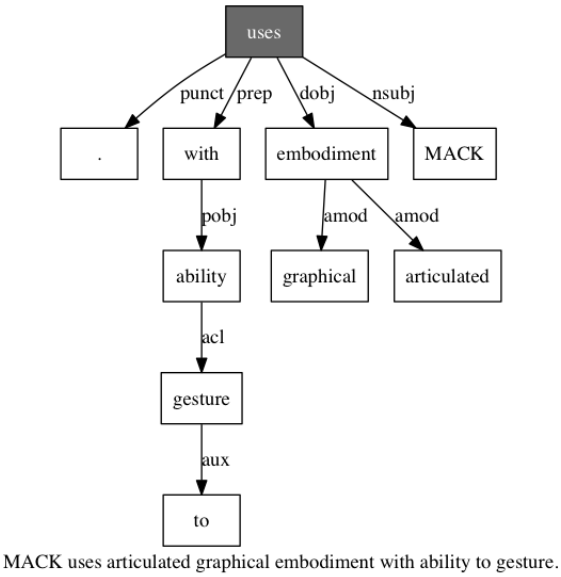


Figure 4.6: The Graphviz output generated by SpaCy dependency tree.

only to the actual dependencies, or edges values, between `token` and its children. This results in the summarized version of the tree presented in Figure 4.7. This is the tree that represents this sentence in this grouping.

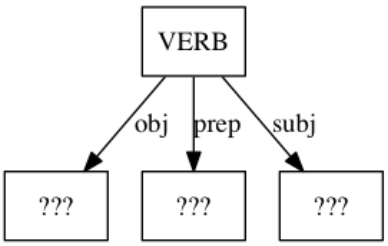


Figure 4.7: The group of the sentence from Figure 4.6.

Note that the *punct* dependency is missing in the final group, and this is due to the rules applied by Algorithm 1 to this resulting tree as to adjust and improve the grouping. More precisely, we wanted to ignore the *punct* dependency in the group, as it had no observed effect in our analysis of the tree and the location of the *subj*-like and *dobj* dependencies we are mostly after. This will be explained further in Section 4.4.

We also used a similar grouping in an attempt to observe the other parameters that are optionally part of the relation, assuming they would be attached to the `token` node by SpaCy. This would be more in line with efforts such as PropBank [38], or VerNet [39]. A more complex example is

the sentence *'Our work not only improves the CPU efficiency by three orders of magnitude, but also reduces the memory consumption'* which, through the same process, produces the group representation in Figure 4.8.

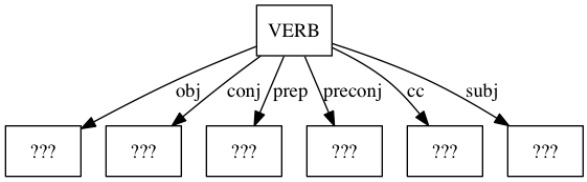


Figure 4.8: A more complex example of sentence grouping.

4.4 Dependency Tree Manipulation Rules

This section further describes the purpose of the rule application methods in the main loop of Algorithm 1. During the process of obtaining the trees for the searched verb, several rules are applied as to organise the tree in a way that facilitates the extraction method. These rules are applied by four different Python classes: `Growth`, `Reduction`, `Obj`, `Subj`, in this order. We use a custom annotator to identify which methods of these classes are actual rules to be applied. The rules are applied in the order they appear in these classes. The addition of new rules, in an investigative setting, simply requires the addition of an annotated method in any of these classes.

The `Growth` and `Reduction` classes apply the rules from the perspective of the node which represents the verb being looked for, i.e., the method receives the verb as the node to do the analysis on. The `Growth` class intends to have rules which cause currently unavailable information to be obtained from other parts of the tree, while the `Reduction` class intends to have rules that remove irrelevant information.

Moreover, in the `Obj` and `Subj` classes, the rules are applied on the node that currently represents respectively:

- *obj*-like relations: Direct Object (*dobj*); Object of Preposition (*pobj*), Indirect Object (*iobj*); or
- *subj*-like relations: Nominal Subject (*nsubj*), Clausal Subject (*csubj*), Passive Nominal Subject (*nsubjpass*), Passive Clausal Subject (*csubjpass*) [31].

Further describing the tree structure, it is important to note the strong characteristic that every node of the dependency tree can have from *0* to n children, however exactly *1* head (or parent) node. For each actual node dependency tree, we also generate a separate tree representation which is used

for grouping. In some cases (which will be denoted), these rules apply only in the representation and not to the original tree. This means that, although we want some trees to be grouped together to facilitate analysis, the original version might still be used for the rule extraction.

A final class called `Extraction` apply a single extraction method which obtains the parts of the relation after all rules above are applied. The extraction method trivially outputs all the child nodes from the node that represents the verb which is the relation being looked for. The rules are defined as follows.

As a baseline example to start with, note the sentence *'We stress that our method improves a supervised baseline'* where the tree generated by the dependency parser is already 'optimal', in the sense that the information is ready to be extracted without any tree manipulation. See Figure 4.9 for the dependency tree. The extracted relation by our tool is `improves ( our method ; supervised baseline )`, which is basically the text form of the verb sub-trees. Consequently, we list now the rules created for our system to increase its ability to extract information in other 'non-optimal', more complex trees.

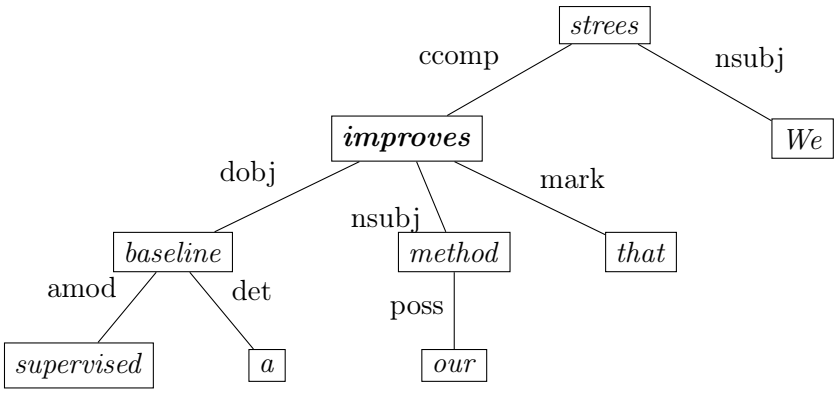


Figure 4.9: A sentence whose relation can be obtained without tree manipulation.

Rule 1 (`Growth`)**.** *If the edge to the head node is of the type* relcl *or* ccomp*, and the existing* subj*-like child node does not have the POS tag* NOUN*,* PROPN*,* VERB*,* NUM*,* PRON*, or* X*, replace the* subj*-like child node with the immediate head node. If there is no* subj*-like child node, simply move the head node as to be its* subj*-like child.*

In Rule 1, we replace the subject when in a relative clause or clausal complement. In this setting, it is common that the verb does not have a *subj*-like child node, or that it has a non-meaningful one (such as *'which'*, or *'that'*). Note how this situation occurs in the sentence *'Calvin [21] is a*

distributed main-memory database system that uses a deterministic execution strategy', when searching for the *'uses'* relation. Figure 4.10 shows the raw tree from the SpaCy dependency parser, and Figure 4.11 shows it after the rule application. The relation extracted in this case is: `uses ( distributed main-memory database system ; deterministic execution strategy )`.

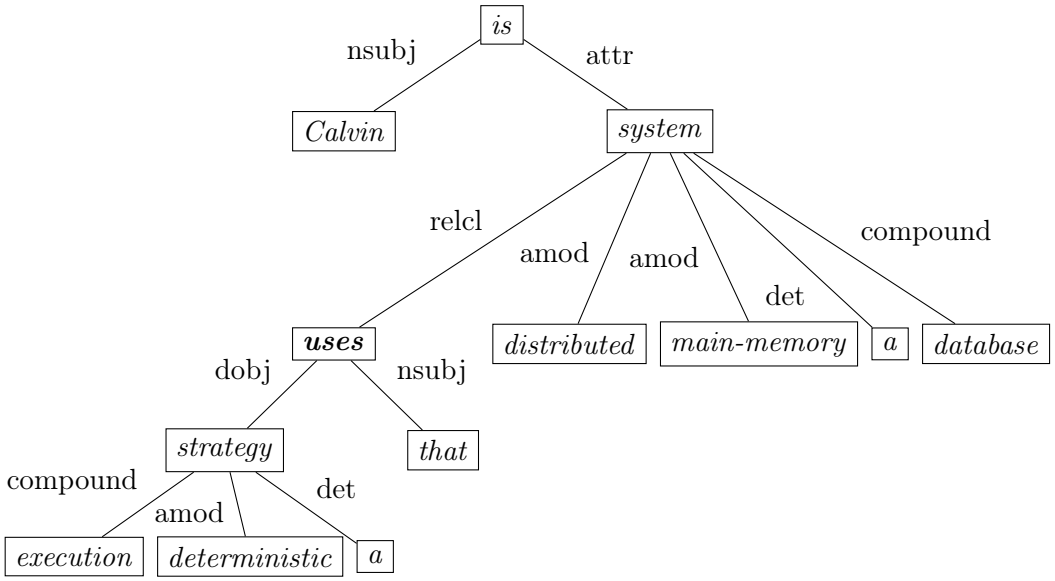


Figure 4.10: A sentence that depicts a tree in which the application of Rule 1 is possible (before).

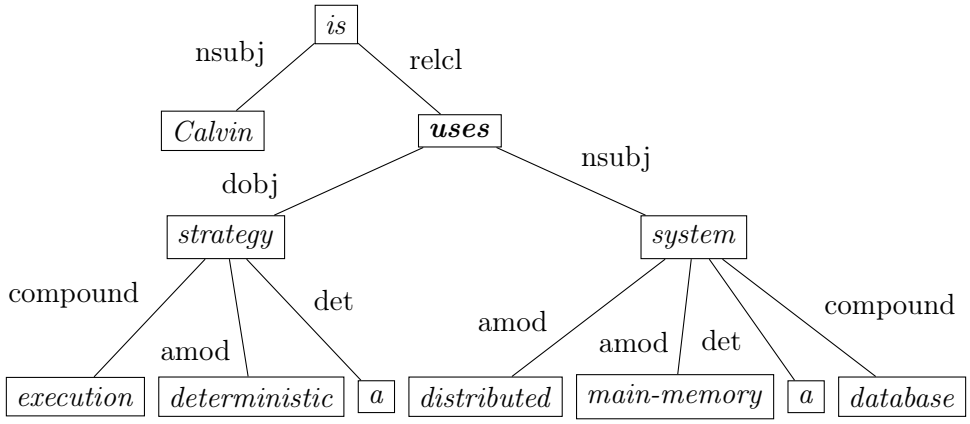


Figure 4.11: The sentence from Figure 4.10 after application of Rule 1.

Rule 2 (`Growth`)**.** *If the current node is part of a* conj *relation through its head edge, and no* subj*-like child node exists, search for a* subj*-like child*

node in the parent (a sibling node). Recurse in case this is not found and the head edge is again a conj.

In Rule 2, we obtain subject from parent if in a conjunct relation. This normally occurs once the parser decides that the relation being searched for is part of a bigger set of relations the subject of the sentence is part of. For an example, note in Figure 4.12 how the sentence *'SemTag uses the TAP knowledge base5, and employs'* depicts the subject *'SemTag'* being further away from the verb *'employs'*, the relation being searched for. In this case, before the rule application *'SemTag'* is a sibling node of *'employs'*, both being child nodes of *'uses'*.

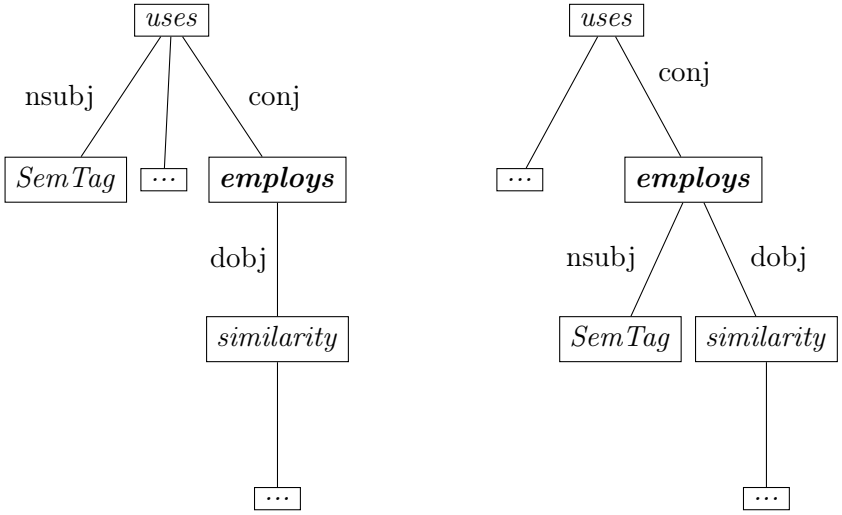


Figure 4.12: A partial tree of a sentence that depicts a situation in which the application of Rule 2 is possible (before on the left, and after the application on the right).

Rule 3 (`Growth`)**.** *If no* obj*-like child node exists, transform nodes* xcomp *or* ccomp *in a* dobj*. If no* subj*-like child node exists, transform nodes* xcomp *or* ccomp *in a* nsubj*.*

Rule 4 (`Growth`)**.** *If no* obj*-like child node exists, transform* prep *relation whose preposition word is* 'in' *in a* dobj *node.*

Rules 3 and 4, handle further transformations, or edge renaming, on existing child nodes to improve relation extractions. In Rule 3 the clausal complements with both internal or external subjects, which often contain the missing part of a relation are renamed to be the subject of or object of the sentence.

An example for Rule 4 is the partial sentence *'matrix co-factorization helps to handle multiple aspects of the data and improves in predicting individual decisions'*, when searching for the *'improves'* relation. Normally, the parser annotates *'in predicting (...)'* as a sub clause with a *prep* edge relation, however, in this case, this clause does contain the object being improved by the subject.

Rule 5 (`Growth`)**.** *If no* obj*-like child edge exists, a* subj*-like child edge exists, and the head edge is of the* subj*-like type, move the head node as to be its* dobj*-like child.*

Another rule that does tree manipulation, Rule 5 caters for situations where the relation being searched for is itself found in a *subj*-like edge connected with its head node. Figure 4.13 notes this rule being applied in the sentence *'This work uses materialized views to further benefit from commonalities across queries'*, when searching for the *'uses'* relation.

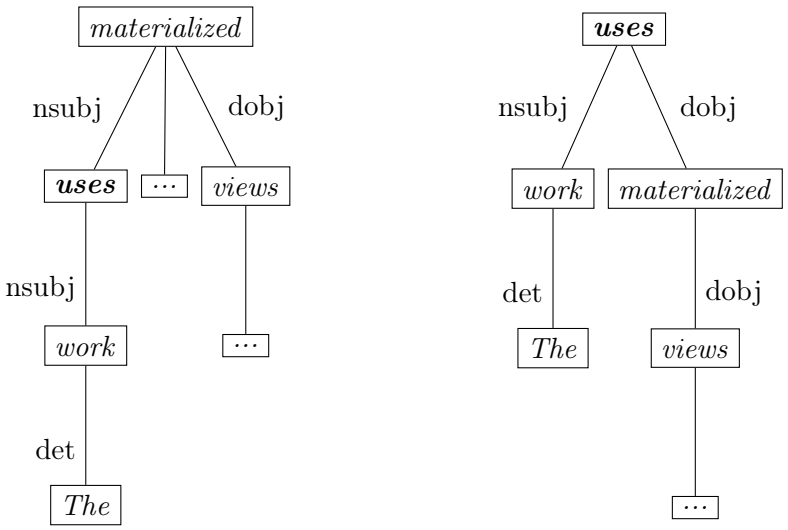


Figure 4.13: A partial tree of a sentence that depicts a situation in which the application of Rule 5 is possible (before on the left, and after the application on the right).

Rule 6 (`Reduction`, representation only)**.** *For any two child with same incoming edge type, remove the duplicate edge.*

Rule 7 (`Reduction`)**.** *Remove tags of type* punct, mark, ' ' *(empty space),* meta.

Rule 8 (`Reduction`)**.** *Transform specific edge types of child nodes into a more general version. More specifically, transform all* obj*-like relations into* obj*, all* subj*-like relations into* subj*, and all* mod*-like relations into* mod.

Rule 6 is the first one we describe of the `Reduction` type, and together with Rules 7 and 8 serve a main purpose of simplifying the tree representation for grouping and analysis purposes. Rule 6 removes duplicates only in the representation and causes the analysis of a node with two *prep* child nodes to be the same as a node with only one.

Rule 9 (`Reduction`)**.** *Merge all* obj*-like relations into one single* obj *node, and all* subj*-like relations into one* subj *node.*

To describe Rule 9, it is important to note the definition of *mod*-like relations, as per the following:

- *mod*-like relations: Noun Phrase Adverbial Modifier (*npadvmod*), Adjectival Modifier (*amod*), Adverbial Modifier (*advmod*), Numeric Modifier (*nummod*), Quantifier Modifier (*quantmod*), Relative Clause Modifier (*rcmod*), Temporal Modifier (*tmod*), Reduced Non-finite Verbal Modifier (*vmod*) [31].

Rule 10 (`Subj` and `Obj`, representation only)**.** *For any two child with same incoming edge type, remove the duplicate edge.*

Rule 11 (`Subj` and `Obj`)**.** *Remove tags of type* det *and* ‘ ’ *(empty space).*

Rule 12 (`Subj` and `Obj`)**.** *Transform specific edge types of child nodes into a more general version. More specifically, transform all* obj*-like relations into* obj*, all* subj*-like relations into* subj*, and all* mod*-like relations into* mod*.*

Rules 10, 11, 12 behave similarly to the `Reduction` rules, but at the `Subj`, `Obj` level - these rules intend to facilitate grouping and analysis.

Furthermore, we observed that, in several situations, the subject or object sentences were too long, mainly due to containing extra information beyond the subject/object concept definition. With information extraction, it is reasonable to assume that the tool should return the information as granular as possible, while still maintaining the possibility for the user to use extra context if needed. In an attempt to alleviate this situation, Rule 13 was created.

Figure 4.14 shows the modification done by Rule 13 in the sentence *‘It uses the exponential mechanism to recursively bisect each interval into subintervals’*, when searching for the *‘uses’* relation. The relation extracted in this case has an extra parameter *mod*: `uses ( subj: It ; obj: exponential mechanism ; mod: to recursively bisect each interval into subintervals )`.

Rule 13 (`Subj` and `Obj`)**.** *Search for the sub-tree rooted by the current node being analysed (either* subj*-like or a* obj*-like) for certain types of nodes and then split the sub-tree in the following way: the found node is removed from the current sub-tree, and moved to be a child node (sub-tree) of the node that*

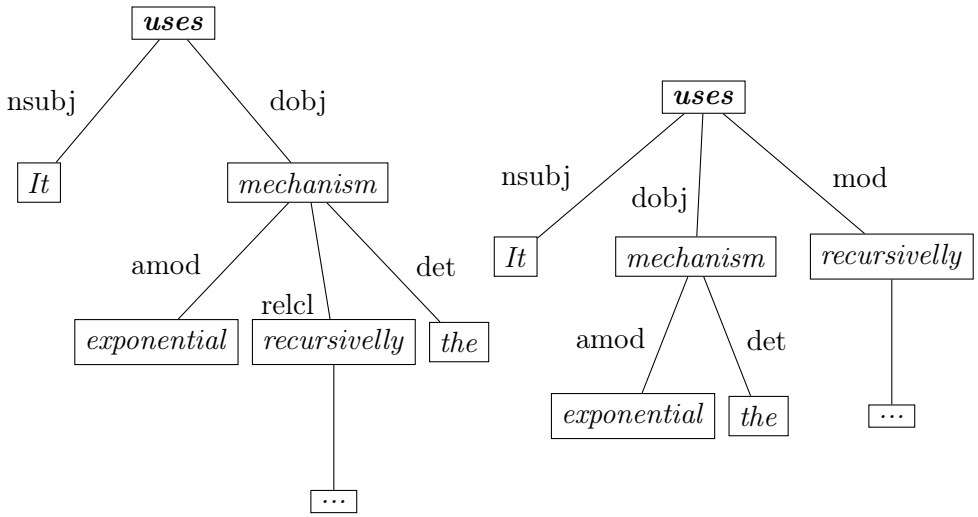


Figure 4.14: A partial tree of a sentence that depicts a situation in which the application of Rule 13 is possible (before on the left, and after the application on the right).

Rule #	**Python method name**
1	Growth.replace_subj_if_dep_is_relcl_or_ccomp
2	Growth.recurse_on_dep_conj_if_no_subj
3	Growth.transform_xcomp_to_dobj_or_sub_if_doesnt_exists
4	Growth.transform_prep_in_to_dobj
5	Growth.add_dobj_if_dep_is_subj
6	Reduction.remove_duplicates
7	Reduction.remove_tags
8	Reduction.transform_tags
9	Reduction.merge_multiple_subj_or_dobj
10	Obj.remove_duplicates; Subj.remove_duplicates
11	Obj.remove_tags; Subj.remove_tags
12	Obj.tranform_tags; Subj.tranform_tags
13	Obj.bring_grandchild_prep_or_relcl_up_as_child; Subj.bring_grandchild_prep_or_relcl_up_as_child

Table 4.1: Rules from this document and the Python method names.

represents the relation (the verb). The node that represents the relation is the head (parent) of the current node being analysed. This rule also renames the node as per the below:

- relcl, acl, advcl *with any token: split and rename to* mod.
- prep *with tokens* 'by', 'to', 'for', 'with', 'whereby'*: split and rename to* prep.

In the HTML output, the tool presents the Python method name of the rules applied to a given sentence. Table 4.1 presents the relation between the rules in this document, and their Python method names.

Finally, after continuous revision, some rules were adjusted to, by omission, also cater for a certain number of cases such as:

- Appositional modifier: once an apposition is found attached to a subject through an *appos* edge, this will be output in the output as part of the relation.
- Punctuation: it is in general also added to the output given, in this corpus, an excess of situations where square brackets or symbols are used to point to extra information around a concept, such as in a references.

Chapter 5

Results

This section describes the experiments and comparisons done of this tool with similar existing ones. To prepare for the experiment, we modified the HTML output to:

- Include the output of three other similar tools: Stanford OpenIE [4], Max Planck Institute ClauseIE [15], AllenAI OpenIE [17].
- Modified the interface as to provide an input for comments, and ability to save and load comments using the JSON format.

5.1 Experiments

To be added: further results of the evaluation.

5.2 Observations

We observed that in some cases the limits of the decision process os this tool, where the linguistic syntactical information from the text might not be enough, and further semantic knowledge might be needed. Note for an example the sentence *'SemTag uses the TAP knowledge base5 , and employs the cosine similarity with TF-IDF weighting scheme to compute the match degree between a mention and an entity, achieving an accuracy of around 82%'*. As a result it has the following main structure, mainly due to Rule 13:

- *obj*: *'SemTag'*
- *sub*: *'cosine similarity'*
- *prep*: *'with TF-IDF weighting scheme, achieving an accuracy of around 82%'*

In this domain, *'cosine similarity with TF-IDF weighting scheme'* would represent a single concept instead, since it is a specific type of *'cosine similarity'*, contrary to what was the output of the rule. One then observes that, for improved correctness Rule 13 should rely on more information and apply reasoning in order to break the sub-tree more appropriately.

Moreover, it was also possible to note the incapacity of the rules to be applied together, or chained, as to output the correct answers. Note, for an example, the sentence *'LSD is an extensible framework, which employs several schema-based matchers'*. A new rule could be developed and named Rule A, which processes the *'is'* relation and follows the *attr* edge as to get the definition for the proper noun *'LSD'* in this case (Figure 5.1). At this moment, this rule would then yield the relation `is ( LSD ; an extensible framework )`. Suppose now the *'employs'* relation is the one actually being searched for. Observing the dependency tree, one could see that Rule 1 would be triggered and cause the head node to be moved and replace the existing *nsubj* child node, yielding `employs ( extensible framework ; several schema-based matchers )`. At this point, the ability to chain both these rules would yield a more complete relation `employs ( LSD ; several schema-based matchers )` since the system would already know what *'LSD'* actually is. In addition, another challenge would be how to have the tool being capable of this decision: when to chain rules, or when knowing that the current result is already optimal.

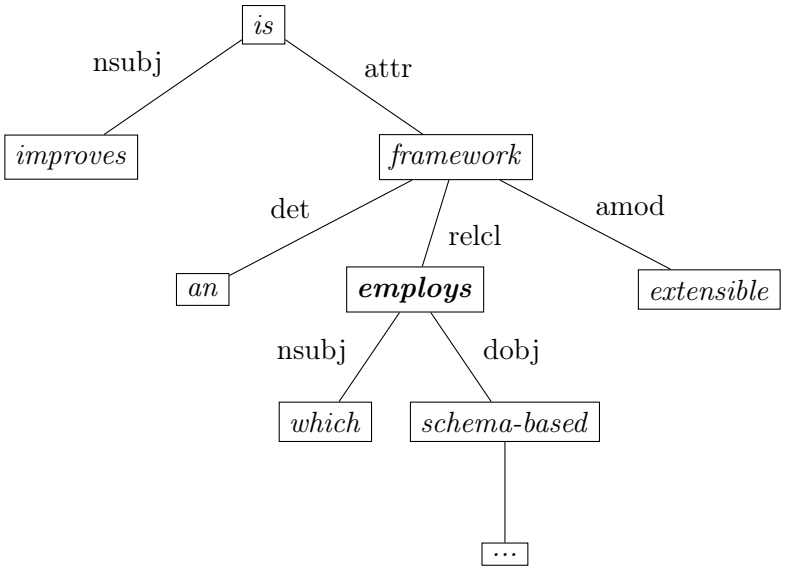


Figure 5.1: A sentence that could benefit from rule chaining.

Another observation comes from the simplicity of the `Extraction` class. In certain situations, multiple relations could have been extracted instead of

one. The first case can be seen in sentence *'As PAS analysis widely employs global and sentence-wide features, it is computationally expensive to integrate'* which in the current tool yields the relation `employs ( PAS analysis ; global and sentence-wide features )`. A more advanced `Extraction` rule could attempt to yield two relations instead:

- `employs ( PAS analysis ; global features )`; and
- `employs ( PAS analysis ; sentence-wide features )`.

The challenge then sits on deciding when to yield multiple sentences, and what are the tokens that compose them. Note that, in this case, we made the non-trivial decision to repeat the token *'features'* in both relations.

The second case being, as previously mentioned, regarding the *appos* edge, or appositional modifier. This appears in situations such as in the sentence *'A similar technique, LightLDA, employs cycle-based Metropolis Hastings mixing'*. While our tool yields one relation `employs ( similar technique LightLDA ; cycle-based Metropolis Hastings mixing )`, a more advanced `Extraction` rule could attempt to yield two relations instead:

- `employs ( similar technique ; cycle-based Metropolis Hastings mixing )`; and
- `employs ( LightLDA ; cycle-based Metropolis Hastings mixing )`.

To be added: further observations done after the experiment.

Chapter 6

Conclusion and Future Work

The maturity and fast-pacing on current development of NLP algorithms and frameworks is very positive and provides advanced linguistic information for tackling problems such as information extraction. We observed that the developed tool was reasonably successful, but as the previous section notes that are room for future work on improving the details of its operation.

The addition of semantic information for reasoning in certain rules application would certainly improve the ability of the system to better decide what to do in certain situations, it is unclear however at this point how this would be done. The entities that are part of the relations would benefit from a good disambiguation system and the development of canonical representations of them.

Extra meta-data from the papers, and the entirety of the paper itself, could start being considered. With this one could attempt to answers questions such as:

- Research relations through time. You could have, e.g., certain historical insights into which algorithm was more popular for a certain task during certain periods;
- Explore coreference resolution more deeply, not only within a paper but across papers and the references between them;
- Events, or introductions of new algorithms or concepts in certain years and how it changes further outputs;
- Building and using a database of the extracted concepts and the relations between them (Knowledge Graph).

To be added: further notes from the experiments

Bibliography

[1] *About SIGMOD*. https://sigmod.org/about-sigmod/. [Online; accessed 09-October-2016]. 2016.

[2] *Adobe: What is PDF?* https://acrobat.adobe.com/us/en/why-adobe/about-adobe-pdf.html. [Online; accessed 14-August-2016]. 2016.

[3] Daniel Andor et al. 'Globally Normalized Transition-Based Neural Networks'. In: *CoRR* abs/1603.06042 (2016).

[4] Gabor Angeli, Melvin Jose Johnson Premkumar and Christopher D. Manning. 'Leveraging Linguistic Structure For Open Domain Information Extraction'. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, 2015, pp. 344–354.

[5] *Apache OpenNLP*. https://opennlp.apache.org/. [Online; accessed 17-October-2016]. 2010.

[6] *Bing Knowledge and Action Graph*. https://www.bing.com/partners/knowledgegraph. [Online; accessed 14-August-2016]. 2016.

[7] Steven Bird, Ewan Klein and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[8] Kurt Bollacker et al. 'Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge'. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1247–1250.

[9] Timothy William Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. http://www.rfc-editor.org/info/rfc7159. [Online; accessed 17-October-2016]. 2014.

[10] Danqi Chen and Christopher Manning. 'A Fast and Accurate Dependency Parser using Neural Networks'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 740–750.

[11] *CiteSeerX*. http://csxstatic.ist.psu.edu/about. [Online; accessed 27-September-2016]. 2016.

[12] Kevin Clark and Christopher D. Manning. 'Entity-Centric Coreference Resolution with Model Stacking'. In: *Association for Computational Linguistics (ACL)*. 2015.

[13] Bhavana Dalvi et al. 'IKE - An Interactive Tool for Knowledge Extraction'. In: *Proceedings of the 5th Workshop on Automated Knowledge Base Construction, AKBC@NAACL-HLT 2016, San Diego, CA, USA, June 17, 2016*. 2016, pp. 12–17.

[14] *Data Engineering, International Conference on*. http://ieeexplore.ieee.org/xpl/conhome.jsp?reload=true&punumber=1000178. [Online; accessed 09-October-2016]. 2016.

[15] Luciano Del Corro and Rainer Gemulla. 'ClausIE: Clause-based Open Information Extraction'. In: *Proceedings of the 22Nd International Conference on World Wide Web*. WWW '13. Rio de Janeiro, Brazil: ACM, 2013, pp. 355–366. ISBN: 978-1-4503-2035-1.

[16] *Empirical Methods in Natural Language Processing*. https://www.aclweb.org/website/emnlp. [Online; accessed 09-October-2016]. 2016.

[17] Oren Etzioni et al. 'Open Information Extraction: The Second Generation'. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 3–10. ISBN: 978-1-57735-513-7. DOI: 10.5591/978-1-57735-516-8/IJCAI11-012. URL: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-012.

[18] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. https://www.w3.org/TR/REC-xml/. [Online; accessed 09-October-2016]. 2008.

[19] Jenny Rose Finkel, Trond Grenager and Christopher Manning. 'Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling'. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. ACL '05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, pp. 363–370.

[20] *Google Knowledge Graph*. https://www.google.com/insidesearch/features/search/knowledge.html. [Online; accessed 14-August-2016]. 2016.

[21] *Google Scholar*. https://scholar.google.com/intl/en/scholar/about.html. [Online; accessed 27-September-2016]. 2016.

[22] Ben Hixon, Peter Clark and Hannaneh Hajishirzi. 'Learning Knowledge Graphs for Question Answering through Conversational Dialog'. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, 2015, pp. 851–861.

[23] Raphael Hoffmann, Luke S. Zettlemoyer and Daniel S. Weld. 'Extreme Extraction: Only One Hour per Relation'. In: *CoRR* abs/1506.06418 (2015).

[24] Matthew Honnibal and Mark Johnson. 'An Improved Non-monotonic Transition System for Dependency Parsing'. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, 2015, pp. 1373–1378.

[25] *Hypertext Markup Language (HTML) 5.1 W3C Proposed Recommendation*. https://www.w3.org/TR/html51/. [Online; accessed 17-October-2016]. 2016.

[26] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[27] Jens Lehmann et al. 'DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia'. In: *Semantic Web Journal* 6.2 (2015), pp. 167–195.

[28] Haojun Ma and Wei Wang. *Academic PDF Content Extraction*. UNSW University of New South Wales, Technical Report. 2016.

[29] Christopher D. Manning et al. 'The Stanford CoreNLP Natural Language Processing Toolkit'. In: *Association for Computational Linguistics (ACL) System Demonstrations*. 2014, pp. 55–60.

[30] Mitchell P. Marcus, Mary Ann Marcinkiewicz and Beatrice Santorini. 'Building a Large Annotated Corpus of English: The Penn Treebank'. In: *Comput. Linguist.* 19.2 (June 1993), pp. 313–330.

[31] Marie-Catherine De Marneffe and Christopher D. Manning. *Stanford typed dependencies manual*. 2008.

[32] *Microsoft Academic Graph*. https://www.microsoft.com/cognitive-services/en-us/academic-knowledge-api. [Online; accessed 27-September-2016]. 2016.

[33] George A. Miller. 'WordNet: A Lexical Database for English'. In: *Commun. ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782.

[34] Marie-Francine Moens. *Information Extraction: Algorithms and Prospects in a Retrieval Context (The Information Retrieval Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[35] Ndapandula Nakashole, Martin Theobald and Gerhard Weikum. 'Scalable Knowledge Harvesting with High Precision and High Recall'. In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. WSDM '11. New York, NY, USA: ACM, 2011, pp. 227–236.

[36] Ndapandula Nakashole, Gerhard Weikum and Fabian Suchanek. 'PATTY: A Taxonomy of Relational Patterns with Semantic Types'. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 1135–1145.

[37] Joakim Nivre et al. *Universal Dependencies 1.3*. http://universaldependencies.org/. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University in Prague. 2016.

[38] Martha Palmer, Daniel Gildea and Paul Kingsbury. 'The Proposition Bank: An Annotated Corpus of Semantic Roles'. In: *Comput. Linguist.* 31.1 (Mar. 2005), pp. 71–106. ISSN: 0891-2017.

[39] Karin Kipper Schuler. 'Verbnet: A Broad-coverage, Comprehensive Verb Lexicon'. PhD thesis. Philadelphia, PA, USA, 2005.

[40] *Semantic Scholar*. http://allenai.org/semantic-scholar/. [Online; accessed 20-August-2016]. 2016.

[41] Jeffrey Mark Siskind and Alexis Dimitriadis. *Qtree, a LATEX tree-drawing package*. University of Pennsylvania. Philadelphia, PA, USA.

[42] *spaCy: Industrial-strength Natural Language Processing*. https://spacy.io/. [Online; accessed 27-September-2016]. 2016.

[43] Pontus Stenetorp et al. 'BRAT: A Web-based Tool for NLP-assisted Text Annotation'. In: *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. EACL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 102–107.

[44] Fabian M. Suchanek, Gjergji Kasneci and Gerhard Weikum. 'Yago: A Core of Semantic Knowledge'. In: *Proceedings of the 16th International Conference on World Wide Web*. WWW '07. New York, NY, USA: ACM, 2007, pp. 697–706.

[45] Mihai Surdeanu et al. 'Customizing an Information Extraction System to a New Domain'. In: *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics*. RELMS '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 2–10.

[46] *SyntaxNet: Neural Models of Syntax*. https://github.com/tensorflow/models/tree/master/syntaxnet. [Online; accessed 01-October-2016]. 2016.

[47] Kristina Toutanova et al. 'Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network'. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*. NAACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 173–180.

[48] *Very Large Data Base Endowment Inc. (VLDB Endowment)*. http://www.vldb.org/. [Online; accessed 09-October-2016]. 2016.

[49] *What is the ACL and what is Computational Linguistics?* https://www.aclweb.org/website/what-is-cl. [Online; accessed 09-October-2016]. 2016.

[50] *Wikipedia, The Free Encyclopedia*. http://www.wikipedia.org/. [Online; accessed 21-August-2016]. 2010.