Roskilde University

Computer Science and Informatics

Master Thesis

Building a knowledge base as a fundament for a personal assistant

A hands-on approach at extracting relation triples.

Authored By

Daniel Șerbănescu
Student no.64513

Supervisor: Troels Andreasen

January 3, 2022

Abstract

The key aspect of this project is to investigate the state-of-the-art techniques within the Natural Language Processing NLP realm for extracting knowledge from text and representing it in a form that facilitates question answering.

In order to facilitate question answering the knowledge base would be used as means to build a personal assistant, subject-predicate-object triplets would be extracted from each sentence provided by the input text and the triplet relationships would be stored in a graph database.

Once the knowledge base is ready it would be used for querying in order to find answers for a personal assistant software.

Keywords: Knowledge base; Knowledge graph; Graph Database;

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 Audience . . . 2
- 1.3 Problem formulation . . . 2
- 1.4 Work scope . . . 3
- 1.5 Thesis scope . . . 3

2 Theory **4**
- 2.1 Knowledge graph . . . 4
- 2.2 Knowledge Bases . . . 6
- 2.3 Natural language processing . . . 6
- 2.4 Feature engineering . . . 7
- 2.5 Triple extraction . . . 8
  - 2.5.1 Feature-Based Models . . . 9
  - 2.5.2 CNN-Based Neural Models . . . 9
  - 2.5.3 Attention-Based Neural Models . . . 9
  - 2.5.4 Dependency-Based Neural Models . . . 9
  - 2.5.5 Graph-Based Neural Models . . . 9
  - 2.5.6 Contextualized Embedding-Based Neural Models . . . 9
- 2.6 Text simplification . . . 13
- 2.7 Clause extraction . . . 13

3 Related research **14**
- 3.1 Standford NLP . . . 14
- 3.2 Vietnamese Triple Extraction . . . 15

4 Approach and Implementation **16**
- 4.1 Overall pipeline . . . 16
- 4.2 Dataset . . . 17
- 4.3 Tools and Libraries . . . 17
- 4.4 Implementation . . . 18
  - 4.4.1 Triple example . . . 22
- 4.5 Querying the graph database . . . 23
- 4.6 Further work . . . 26

5 Experiments **27**
- 5.1 Triple extraction with spacy . . . 28

6 Discussion **31**
- 6.1 Triple selection . . . 31
- 6.2 Co-reference . . . 31
- 6.3 Text simplification . . . 31

7 Conclusion **32**

Chapter 1

Introduction

Knowledge Graphs is one concept of NLP - Natural Language Processing that has appliances within question answering, text categorization, phrase identification and information retrieval.

A knowledge graph is a representation of facts that are represented as entities relationships and descriptions. Entities can be real or abstract concepts, relationships represent the relation between entities.

A common variant of knowledge graphs are property graphs or attributed graphs which are graphs with nodes that have attributes.

Throughout this project I will explore the process of extracting information from text in a way that could be stored as knowledge graphs.

1.1 Motivation

As the information quantity increases there is an increasing need of software programs that can deal with information and will support the user in finding, organizing and retrieving it.

A personal assistant software that has its main focus on assisting the user's knowledge-based decisions would be extremely helpful for persons dealing with a lot of information and knowledge. Those users can be students, teachers, researchers, journalists, and so on.

Therefore there is a need of extracting and structuring knowledge in order to be easily queried. The precision of the results must be high enough in order to be reliable and usable.

1.2 Audience

One of the main objectives of this tesis is to streamline the process of information extraction from text to a graph database. I will provide as many practical examples as possible, including diagrams, code excerpts and comparison tables.

Therefore the main audience of the thesis are persons within academia and computer scientists.

1.3 Problem formulation

This work aims at answerig the following research question:

Can text represented as a knowledge graph be used to feed knowledge to a computer-aided personal assistant?

Subquestions:

- Which techniques are best suited to extract new knowledge?
- How to represent the extracted knowledge as a graph?
- How to query the knowledge graph in order to answer questions?

1.4 Work scope

The scope of this paper revolves around building a structured graph of concepts within the domain of Earth Science. Earth science is an umbrella-term embracing all the siences related to planet earth. It includes the following fields of study:

- Geology - study of rocks and rock formations
- Geophysics - study of shapes of objects present on planet Earth
- Soil science - study of soil and its formation
- Oceanography - study of the oceans and their fauna
- Glaciology - study of the icy parts of the Earth
- Atmospheric sciences - study of the atmosphere and its composition
- Physical geography - study of the natural environment
- Biology - study of life and living creatures

Following are a couple of examples of query questions that would be sent to the graph-based system:

1. What is the capital of Japan? Answer: Tokyo
2. What is the Artic Circle? Answer: A Circle of latitude
3. When was the Earth's crust formed? Answer: 4.6 billion years ago

1.5 Thesis scope

The thesis focuses mainly of information extraction. The information comes as text structured in sentences. The sentences at hand are clean, they are peer reviewed, and therefore they do not need spell-checking.

However since information extraction is a big and rich field of study, I will not focus on anything beyond the processes needed to extract information from text for storage in a knowledge graph.

Chapter 2

Theory

2.1 Knowledge graph

Extracting human knowledge is one of the research directions of Artificial Intelligence (AI). Knowledge representation for problem solving is done to achieve unprecedented goals and solve complex tasks. The earliest attempts at extracting human knowledge were done in 1959 [1] when a general problem solver was attempted. During the recent years knowledge graphs gained momentum and were in attention of both researchers and industry.

A knowledge graph is a structured representation of facts. These facts are divided in entities, relationships and semantic descriptions. Entities are either abstract or real-world concepts, relationships are connections between entities and semantic descriptions. A widely used type of graph is the property graph which has elements with additional properties.

The elements of a graph are triplets in the form of subject-predicate-object which are simmilar to the gramatical elements of an english sentence.

Within the field of knowledge graph, research focus is split among:

- KRL - Knowledge Representation Learning
- Knowledge Aquisition
- Temporal Knowledge Graph
- Knowledge-aware applications

Knowledge Representation Learning is a research field focusing on four key aspects: representation space, scoring function, encoding models and auxiliary information.

- Representation space - where relations and entities are represented
- Scoring function - measuring the plausability of factual triples
- Encoding models - modeling the semantic interaction of facts
- Auxiliary information - external information to be incorporated into the embedding methods

KRL is also known under three other different names: KGE - Knowledge Graph Embedding, multi-relation learning and statistical relation learning.

Knowledge Aquisition aims at building knowledge graphs from unstructured text or other semi-structured sources. Then completing an existing knowledge graph by discovering and recognizing entities and relations. The resulted knowledge graph is useful for many downstream applications and AI implementations.

Knowledge Aquisition encompasses tasks divided into three categories: Knowledge Graph Completion (KGC), relation extraction and entity discovery. KGC is used for expanding knowledge graphs while the last two are used to discover new knowledge given a chunk of text.

Knowledge Graph Completion is used as a means to add new triples to existing knowledge graphs. As typical subtasks one can name link prediction, entity prediction and relation prediction.

At the beginning of KGC research the focus was on learning low-dimensional embedding for triple prediction. Ji et al. [2] refers to them as embedding-based methods.

Temporal Knowledge Graphs, also known as episodic or time-dependent knowledge graphs, include temporal information to be used for representation learning. An episodic knowledge graph can be thought of as a sequence of semantic knowledge graphs incorporated with timestamps. Therefore the relations stored in a temporal knowledge graph are quadruples comprised of the classic subject-predicate-object and a timestamp.

Knowledge-aware applications are part of Natural Language Understanding (NLU) and are used to inject knowledge with the purpose of improving representation learning.

Knowledge-aware applications include: Question Answering, Dialogue Systems, Recommender Systems and others.

The entire map of the research field was represented in a recent survey published in the IEEE journal by Ji et al. [2], and it looks at follows:

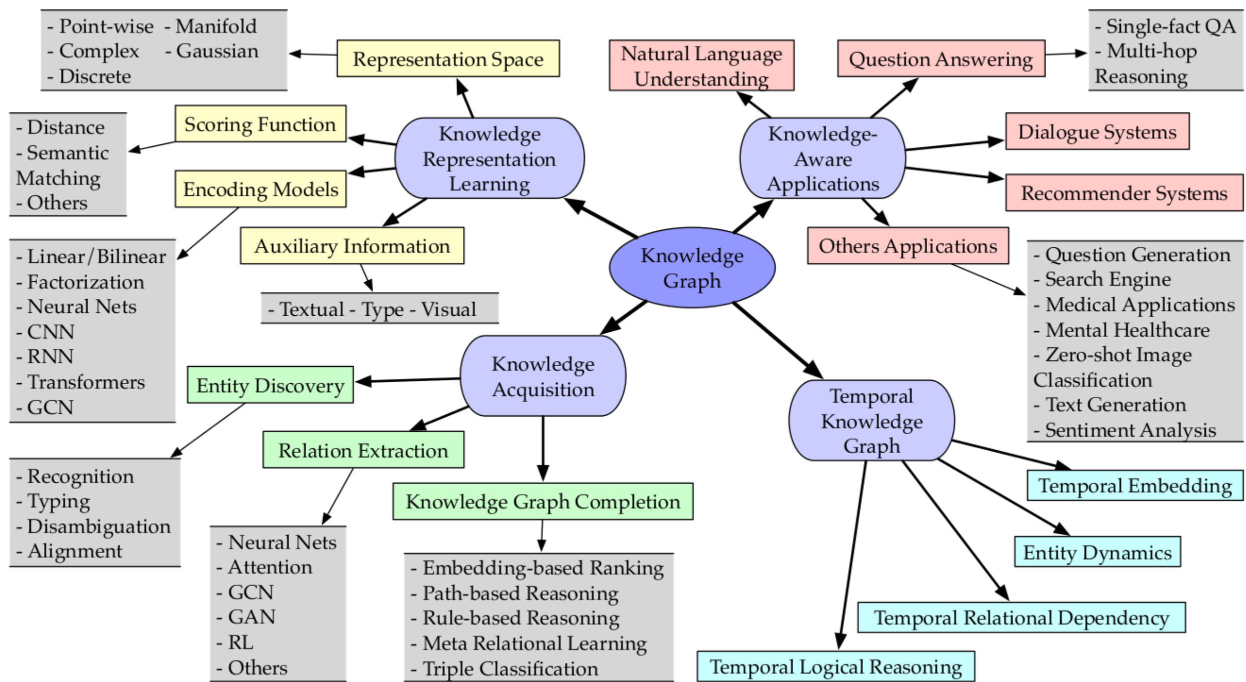

Figure 2.1: Map of the Knowledge graph research field as presented in the survey.

This project focuses mainly on knowledge aquisition which refers to constructing knowledge graphs from unstructured text. Well-constructed and large-scale knowledge graphs provide the building blocks for AI applications.

The tasks of knowledge aquisition include relation extraction, Knowledge Graph Completion (KGC), entity recognition, entity alignment, triple classification, entity classification and open knowledge enrichment.

2.2 Knowledge Bases

A Knowledge Base (KB) is a data storage hub that contains information about a product, service, topic or concept. It is used by organizations and can be targeted at employees or the wider public. The goal of a Knowledge Base is to provide logical information to its users in order to increase the understanding of the knowledge it provides.

There are many types of knowledge, but the main distinction is made between explicit and implicit knowledge.

Explicit knowledge is knowledge or skills that can be easily understood and transferred to others.

Implicit knowledge is knowledge with the potential to be codified but has not yet been transferred into the system, making it harder to teach.

There is another distinction between types of knowledge, factual and heuristic knowledge. Factual knowledge can be measured and verified by data while heuristic knowledge is determined by intelligent guesswork.

Knowledge can be extracted from the Knowledge Base using techniques like forward chaining and backward chaining.

Forward chaining is a reasoning method that solves problems by looking at the available data and makes use of inference rules to extract information. The inference rules applied are a collection of If-Then statements.

Backward chaining is a reasoning method applied to solve problems. It begins with a list of goals and identifies conditions that fulfill the goals (conclusion).

2.3 Natural language processing

In order to extract graph data from text we need to preprocess the text at hand.

The preprocessing includes spell checking, tokenization, normalization and other processes.

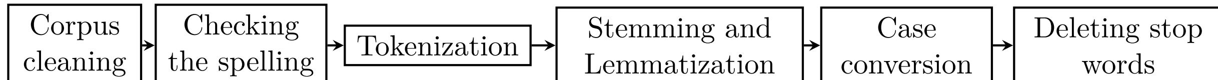

Figure 2.2: A typical preprocessing pipeline

Tasks:

1. Corpus cleaning - keeping valuable data while deleting noise data
2. Spell checking - finding spelling errors and improving spelling quality
3. Tokenization - breaking sentences into words
4. Stemming and Lemmatization - converting words to their basic form
5. Case conversion - converting all uppercase letters to lowercase
6. Deleting stop words - removing words that do not affect sentence understanding

Corpus cleaning is the task of removing irrelevant data from the corpus while retaining important data. The corpus may contain formatting of some sort (XML, HTML or markdown) which only complicates the process of triple extaction. Hence this formatting needs to be removed before we proceed with the next step of text processing.

Spell checking is another task that improves the quality of the corpus. Depending on the source of the text, the corpus may or may not need spell checking. For example if the data consists of user comments from a product review website we need to employ spell checking as the text was not checked before submission and the amount of spelling errors would be higher than for example a news article from a reputable newspaper website. Other sources with low spelling errors count could be Wikipedia articles, scientific papers or published books.

Tokenization is the task of finding tokens within the sentences of the text. These tokens can be single words (like "Athens") or compound words (like "New York") that represent named entities or concepts. Therefore the task of tokenization is closely connected with named entity recognition (NER).

Stemming is the task of reducing words to their root form, also called a stem. In order to achieve stemming the words are usually stripped of their suffixes as in "loving" becomes "lov" and "parses" becomes "parse".

Lemmatization reduces the inflected words properly ensuring that the root word belongs to the language. The root word for the lemmatization process is called lemma. The lemma is the dictionary form of the word, without inflections, plural or verb tense form.

Case conversion is the process of converting the entire corpus to lowercase as it is easyer to process under certain circumstances. This is useful when the corpus contains mix-case formations.

Deleting stop words is the process of removing irrelevant words as the article "the" or other short words such as "at", "which" and "on" that appear often in the corpus but do not convey any meaning.

2.4 Feature engineering

Feature engineering is the process of converting sentences in a computer-understandable form, usually a vector.

Common used models are bag-of-words (BOW) and word embeddings. BOW is usually used together with TF-IDF (Term Frequency-Inverse Document Frequency) model. Word embedding converts a word into a fixed-length vector in order to be mathematically processed. Examples of word embedding algorithms are one-hot encoding, word2vec and doc2vec.

One hot encoding is a process of converting categorical data variables in order to be provided to machine learning algorithms to include predictions. This is a crucial part of feature engineering for machine learning.

Categorical data is made up of variables that contain label values. As an example a "color" variable could have the values "red", "blue", and "green", much like differenet categories that could have a natural order.

Most machine learning algorithms require the inputs and outputs to be numbers so therefore categorical data is usually mapped to integers.

So one hot encoding is one of the methods used to convert data in a format that is better suited for improving prediction.

word2vec [3] is a two-layer neural net that processes text by converting words into vectors. Its input is a text corpus while the output is a set of vectors.

Word2vec is applied to genes, code, likes, playlists, social media graphs and so on. Word2vec facilitates detecting the likelihood of the words co-occuring.

The usefulness of word2vec is to group vectors of similar words togheter in a vectorspace, thus detecting similarities mathematically. With enough data, usage and context word2vec makes highly accurate guesses about meaning of the words based on past occurences.

doc2vec is used to create a numerical representation of a document. doc2vec uses the word2vec model with an extra vector for paragraphs which is document-unique.

2.5 Triple extraction

Triple extraction or Relation extraction (RE) is the task of extracting relationships from text. Relation extraction is a subfield of Information Extraction.

A relation triplet consists of two entities and a relationship between them. Such triples can be found in publicly available knowledge bases as Freebase [4], DBpedia [5], Wikidata [6] and others.

When we look at relation extraction there are two paradigms: open information extraction (Open IE) and supervised relation extraction.

Open information extraction is done with an open set of relations, meaning no restrictions on the amount of relations.

Where **supervised relation extraction** is based on rules that define the methods for extracting entities from noun phrases and relations from the verb phrases present in sentences.

Given the unlimited possibilities of relation triples, OpenIE will often lead to duplicate relations because it considers variations of the verbs to be different relations, hence leading to duplication of triplets.

The mentioned problems of Open IE can be addressed using supervised relation extraction that makes use of a fixed set of relations and a parallel corpus of text with relation triples for training.

Classification of triple extraction

Pipeline based extraction approaches fall into two types of tasks:

- entity recognition
- relation classification

Within **entity recognition** all the candidate entities are identified within the borders of a sentence. In **relation classification**, every possible ordered pair of candidate entities is determined, but the relation may not exist and be marked with "(None)".

Joint-extraction approaches look for entities and relationships at the same time and extract only the valid triples, so they do not need to extract None-triples.

Relation triples may share one or both entities between them, hence making the task challenging.

Based on entity overlap we can distinguish the following:

- No Entity Overlap (NEO) - one or more triples in a sentence, but no overlapping entities
- Entity Pair Overlap (EPO) - a sentence with at least two triples sharing both entities
- Single Entity Overlap (SEO) - a sentence with more than one triplet with two triplets sharing exactly one entity

The goal is extracting all relation triplets that are present in a sentence.

Pipeline Extraction Approaches:

- Feature-Based Models
- CNN-Based Neural Models
- Attention-Based Neural Models
- Dependency-Based Neural Models
- Graph-Based Neural Models
- Contextualized Embedding-Based Neural Models

2.5.1 Feature-Based Models

Mintz et. al. [7] proposed in 2009 a feature-based relation classification model for the task of extracting triples. Lexical features such as sequence of words between two entities and their part-of-speech (POS) tags along with a flag that indicates which entity appears first, were used. Syntactic features such as dependency between two entities and named entities types were also used in the presented model.

2.5.2 CNN-Based Neural Models

Word embeddings gave a major advantage to natural language processing tasks like Information Extraction (IE). Word2Vec and GloVe are the largest word embeddings publicly available. Their high dimensional distributed representation of words can encode crucial semantic information about the words themselves which are helpful in identifying relationships between entities within a sentence. At their inception, neural models also follow the pipeline approach to solve the task of relationship extraction. A Convolutional Neural Network (CNN) is a deep learning neural network designed for processing structured arrays of data. A convolutional neural network is a feed-forward neural network with up to 20-30 layers. Its power comes from a special layer called the convolutional layer.

2.5.3 Attention-Based Neural Models

Attention networks are very useful to different NLP tasks. They can be word-level attention model or sentence-level attention models. Rather than using all available information, attention mechanism aims to focus on the most pertinent information for the task at hand.

2.5.4 Dependency-Based Neural Models

The dependency structure information of sentences can be used in relation extraction. Including the dependency parse tree of an input sentence is beneficial for reordering because the dependency tree captures the relationships between words in a sentence, through the dependency relation label between two words.

2.5.5 Graph-Based Neural Models

Graph-based models work well with non-linear structures. Quirk [8] proposed a graph-based model where each word of the sentences was extracted as a node in the graph and edges were created based on the adjacency of the words, dependency tree relations and discourse relations. Each path was represented by lexical tokens, lemma of the tokens, part-of-speech tags, and so on. All the features were then used to find the relation between two entities.

Other similar graph-based approaches were used since then.

2.5.6 Contextualized Embedding-Based Neural Models

Contextualized word embeddings can be useful for relation extraction. These types of language models are trained on language corpora and will capture contextual meaning of words in their vector representations. Most of the neural models proposed for relation extraction use word representation such as Word2Vec and GloVe. The contextualized embeddings can be added in the embedding layer of the relation extraction models to further improve the performance.

Rule-based entity extraction

One can use a rule-based approach for extracting triples from the text. Rules can be defined based on part-of-speech tagging and dependency parsing. The results of rule-based triple extraction are promising but requires longer time for fine-tuning the rules.

```
def extractSubject(sentence) :
    parsed_sentence = nlp(sentence)
    matcher = Matcher(nlp.vocab)
    # Matches composite subject
    # (all modifiers and the punctuation + subject)
    pattern = [
        {"DEP" : {"IN" : ["compound", "nummod", "npadvmod"]},
        "OP": "+", "SPACY": True},
        {"DEP" : {"IN" : ["nsubj", "nsubjpass"] } }
    ]
    matcher.add("MultiWordSubj", [pattern])
    matches = matcher(parsed_sentence)
    # If we have no match for a composite subject we try a simple,
    # non-composite one
    if not matches:
        pattern2 = [
            {"DEP":"nsubj"}
        ]
        matcher.add("SingleWordSubj", [pattern2])
        matches = matcher(parsed_sentence)
    for match_id, start, end in matches :
        span = parsed_sentence[start:end]
        return span
```

Listing 2.1: Function that extracts simple and composite subject out of a sentence.

In the listing 2.1 we have a pattern for matching simple and compound subjects based ob dependency between words. We are searching for words that are compounds, numerical modifiers and adverbial modifiers for noun subjects.

Pipeline extraction approaches were popular at the very beginning of the study in the relation-extraction research. A pipeline approach typically consists of two steps: entity recognizer and classification model. The entity recognizer is used to identify named entities within a text while the classification model facilitates finding the relationship between two entities.

Relation extraction can be done in the following ways:

- Rule-based RE
- Weakly Supervised RE
- Supervised RE
- Distantly Supervised RE
- Unsupervised RE

Rule-based RE is done using a set of rules, usually with a regular expression that match them. Looking after keywords might result in many false positives. To reduce false positive named entity recognition can be applied to recognize named entities.

With this approach one does effectively *word sequence patterns* and this is limited by the sequence of words. If the words are not in sequence the pattern-approach will fail.

To improve the out-of-sequence triple extraction, gramatical dependency paths can be employed giving the pattern matcher information about the gramatical parts of speech and their dependencies.

Another approach that would improve Rule-based RE is to reword the sentence into a linear form before extracting triples.

Pros	Cons
Humans build a parttern with high precision. Can be tailor made to a specific domain.	Human-made patterns usually have low recall due to greater variation within languages. Increased manual labour for creating many rules.

Table 2.1: Pros and cons of using Rule-based RE.

Weakly Supervised RE

Is based on starting with a defined set of rules and automatically adding new ones through an iterative process.

An example of an algorithm which does this is "Snowball" [9]. Snowball starts with a set of seeds that contain tuples of related words, then tags them employing NER - Named Entitty Recognizer. With the data at hand it creates patterns of occurences and generates new tuples from the text. Then it can either iterate again from the employing NER step or finish the extraction.

Pros	Cons
More relations detected than rule-based RE. Less human effort is required.	Patterns are more error-prone at each iteration. Attention needed with NER-based patterns. New relation types require new seeds.

Table 2.2: Pros and cons of using Weakly Supervised RE.

Supervised RE

Relation Extraction can be done by training a binary classifier to determine if there is a relation between two entities. The training and extraction can be doneby:

- Manually label the data acoording to relevance for a specific type.
- Manually label relevant sentences as positive/negative if the relation is found.
- Teach the binary classifier to determine relevance for the relation type
- Teach the binary classifier to detect if the relation is present or not
- Use the classifier to detect relations in unseen text.

Pros	Cons
High quality supervision. Explicit negative examples are present.	Expensive to label examples. Difficult to add new relations. Does not generalize to new domains. Feasible for small set of relation types.

Table 2.3: Pros and cons of using Supervised RE.

Distantly Supervised RE

When we combine the method of using seed data from Weakly Supervised RE and a training classifier from Supervised RE, we obtain Distantly Supervised RE. However we do not provide a set of tuples ourselves but import them from a Knowledge Base like Wikipedia, DBpedia, Wikidata, Freebase or Yago.

1. For each relation type we get data from the Knowledge Base
2. For each tuple of this relation in the Knowledge Base
3. Select sentences from our text data that mach these tuples
4. Extract features from thes sentences (like POS, context words, etc)
5. Train a supervised classifier on these features

Pros	Cons
Less manual effort. Can scale to use large data with many relations. No iterations required	Noisy annotation of training corpus No explicit negative examples. Restricted to the Knowledge Base May require careful tuning.

Table 2.4: Pros and cons of using Distantly Supervised RE.

Unsupervised RE Unuspervised RE happens when we extract relations from the text without labeling and training data or provide a set of seed tuples or even writing rules to detect relations in the text. In place of all of the above we rely on a set of very general constraints and heuristics.

These systems require less supervision in general. Open Information Extraction (Open IE) usually referes at this approach, namely Unsupervised RE.

OpenIE 5.0 and Standford OpenIE are two open-source systems that do Unsupervised RE. These systems output many relationships types because they are not constrained on which types to return.

Pros	Cons
No/almost no training data required. Considers all types of relations.	System performance depends on heuristics. Relation types are not predefined.

Table 2.5: Pros and cons of using Unsupervised RE.

2.6 Text simplification

Text simplification is a process used in NLP (Natural Language Processing) in order to modify, enhance, classify and process an existing corpus of text in a way that the grammar structure is significantly simplified while the meaning and information conveyed by the text remains the same.

Text simplification would help, among others, triple extraction as the text at hand would be converted in simpler senteneces with easier-to-tackle grammar.

One key aspect of text simplification is clause extraction.

2.7 Clause extraction

A clause is a part of a sentence that expresses a coherent piece of information. A clause consists of a subject (S), a predicate or verb (V) and optionally an indirect object (O), a direct object (O), a complement (C), and one or several adverbials (A). Some of these components might be absent in the English language but present in other languages.

When we classify clauses according to their grammatical functions of their elements we obtain 7 types of clauses [10]. The types of clauses are represented in the following diagram.

	Pattern	Clause type	Example	Derived clauses
			Basic patterns	
S_1:	SV_i	SV	AE died.	(AE, died)
S_2:	SV_eA	SVA	AE remained in Princeton.	(AE, remained, in Princeton)
S_3:	SV_cC	SVC	AE is smart.	(AE, is, smart)
S_4:	$SV_{mt}O$	SVO	AE has won the Nobel Prize.	(AE, has won, the Nobel Prize)
S_5:	$SV_{dt}O_iO$	SVOO	RSAS gave AE the Nobel Prize.	(RSAS, gave, AE, the Nobel Prize)
S_6:	$SV_{ct}OA$	SVOA	The doorman showed AE to his office.	(The doorman, showed, AE, to his office)
S_7:	$SV_{ct}OC$	SVOC	AE declared the meeting open.	(AE, declared, the meeting, open)
			Some extended patterns	
S_8:	SV_iAA	SV	AE died in Princeton in 1955.	(AE, died)
				(AE, died, in Princeton)
				(AE, died, in 1955)
				(AE, died, in Princeton, in 1955)
S_9:	SV_eAA	SVA	AE remained in Princeton until his death.	(AE, remained, in Princeton)
				(AE, remained, in Princeton, until his death)
S_{10}:	SV_cCA	SVC	AE is a scientist of the 20th century.	(AE, is, a scientist)
				(AE, is, a scientist, of the 20th century)
S_{11}:	$SV_{mt}OA$	SVO	AE has won the Nobel Prize in 1921.	(AE, has won, the Nobel Prize)
				(AE, has won, the Nobel Prize, in 1921)
S_{12}:	$ASV_{mt}O$	SVO	In 1921, AE has won the Nobel Prize.	(AE, has won, the Nobel Prize)
				(AE, has won, the Nobel Prize, in 1921)

S: Subject, V: Verb, C: Complement, O: Direct object, O_i: Indirect object, A: Adverbial, V_i: Intransitive verb, V_c: Copular verb, V_c: Extended-copular verb, V_{mt}: Monotransitive verb, V_{dt}: Ditransitive verb, V_{ct}: Complex-transitive verb

Chapter 3

Related research

3.1 Standford NLP

The Natural Language Processing (NLP) Group [11] is a team of Standford faculty members and students that work together on algorithms that process, generate and understand human languages. The group works on simple as well as complex computational linguistics tasks. Example tasks can be sentence understanding, automatic question answering, machine translation, sytactic parsing and tagging, sentiment analysis, dialogue agents, models of text and visual scenes as well as applications of natural language processing for digital humanities and coputational social sciences.

The Standford NLP Group provides a widely used integrated NLP toolkit, the Standford CoreNLP toolkit [12].

The areas from Standford NLP that are interesting for this project are automatic question answering and syntactic parsing and tagging. Another tool provided by Standford NLP Group is Open Information Extraction, OpenIE for short.

Standford OpenIE [13] is a tool for extracting relation tuples, usually binary relations from plain text. The main difference between open information extractors and others is that the open IE does not need a schema specified in advance in order for relations to be found. Typically the relation name is just the text linking two arguments. As an example "Barack Obama was born in Hawaii" would result in a triple (Barack Obama; was born in; Hawaii) which corresponds to the open domain relation was-born-in(Barack-Obama, Hawaii).

Standford OpenIE is a java implementation of an open IE system based on the paper of Angeli et. al. [14]. The system first splits each sentence into a set of entitled clauses. Then each of the clauses is maximally shortened producing sentence fragments. Afterwards these fragments are reduced to OpenIE triples and output by the system.

An example of dependency parsing and triple extraction is shown in the following diagram.

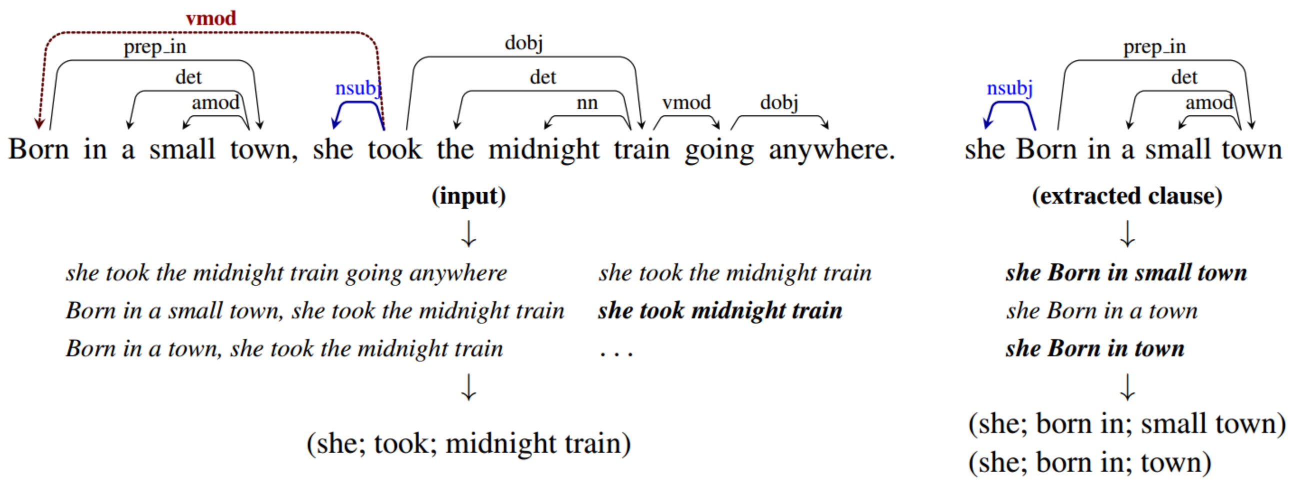

Figure 3.1: An example of OpenIE triple extraction [13]. From left to right: A sentence results in a number of indepented clauses. From top to bottom: each clause produces a set of shorter segments which in turn match a triple.

3.2 Vietnamese Triple Extraction

Another paper that deals with triple extraction is: «*Extracting triples from Vietnamese text to create knowledge graph*» by Huong Duong To and Phuc Do [15]. This paper aims to fill the gap of missing knowledge graphs for the Vietnamese language. Their approach is to scrape vietnamese websites for input text then employing NER - Named-entity-recognition combined with POS - Part-of-speech tagging to recognize the relation verbs and the rest of the triple in the sentences of a paragraph. In the end the resulted triple was loaded in a neo4j graph database.

From parsing almost 1000 sentences the extracting triple accuracy was 84.20And they have encountered the same drawbacks as I did during this project. Namely the triplet extraction thrives only on simple sentences. Duplicate information sneaks into the Knowledge Graph. There is a need for checking the reliability of extracted triples.

Overall this paper gives good insights into the process of extracting triples for a new language / new domain.

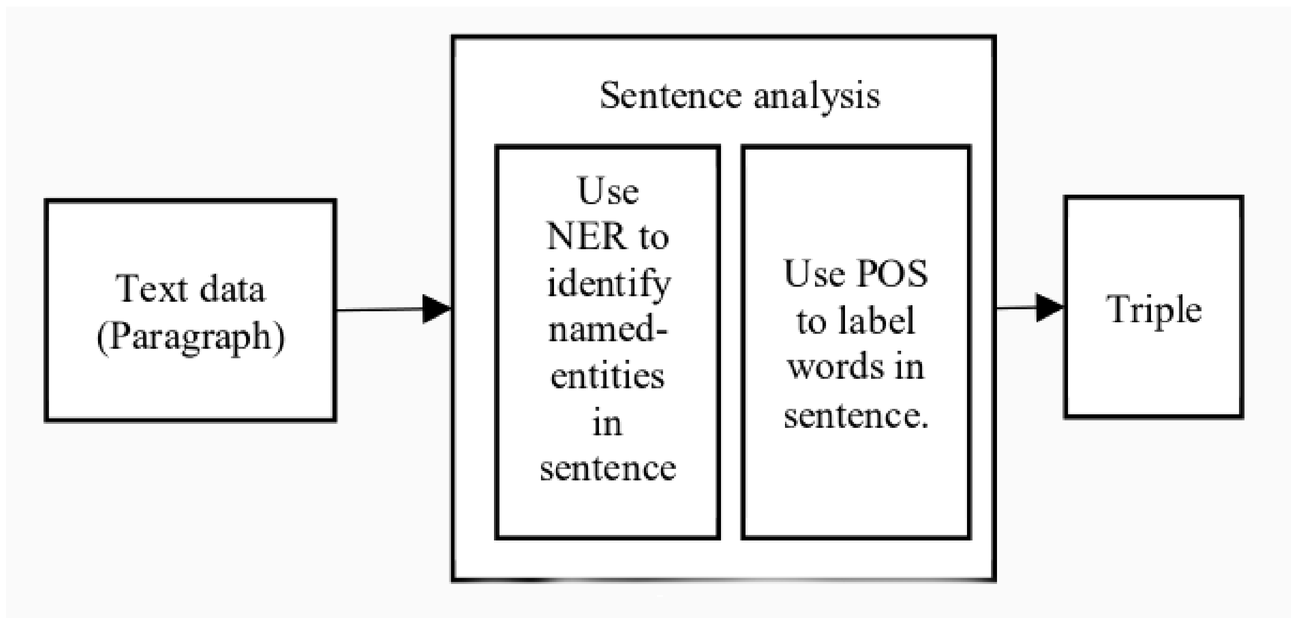

Figure 3.2: The approach to extract vietnamese triples.

Chapter 4

Approach and Implementation

This chapter's purpose is to explain the overall method of extracting relevant information from the Corpus imported from Wikipedia and the processing of the text all the way to SVO-triples stored in a graph database. Choosing the dataset and explaining the tools and code used will follow in this chapter.

4.1 Overall pipeline

In order to prepare the corpora text for triple extraction a text processing pipeline needs to be built.

The pipeline consists of the following steps:

1. Sentence splitter - divide the text into sentences.
2. Dependency parsing - extract the gramatical dependencies of words
3. Extract triplets - extract subject-predicate-object triples using the defined rules
4. Store the triplets - storing triplets in the graph database

In order to improve the pipeline extra steps could be added:

- Sentence segmentation / clause extraction. This is further splitting of long sentences or phrases in more simple sentences while retaining the meaning.

In the end, when we extract the structured knowledge and make use of it we will query the newly created knowledge base.

1. Ask a question - send a question to the chat-bot that queries the knowledge base
2. An answer is retrieved - the chat-bot would respond according to its findings or a generic answer will be provided

The implementation of the pipeline looks like this:

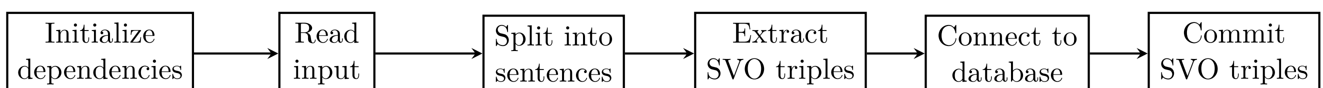

Figure 4.1: Overview over the processing pipeline

In the 4.1 the overall architecture of the prototype built for demonstration purposes is represented.

As we can see we start by initialization of program dependencies then we proceed with reading the input text. Afterwards the next step is to split the text at hand into sentences. Once that is done the SVO triples are extracted from each sentence. There might be multiple triples per sentence. At the end of the pipeline we connect to the Graph database and submit our triples.

Once the triples reside in the graph database they are ready for querying directly or by any other program.

4.2 Dataset

The dataset used is based on the Simple English variant of Wikipedia. As this project aims to provide a proof-of-concept only a handfull of Wikipedia articles will be parsed, leaving the rest of the dataset unprocessed. The parsed articles will be related to the topic of Earth Science.

Dataset extraction from Wikipedia is done through the fetching of the latest monthly package available for download on Wikipedia Dumps page. The downloaded dump is in compressed xml format and needs to be preprocessed before it can be used.

Dataset preprocessing refers to converting the xml file downloaded from Wikipedia into a plain-text file. For this purpose I make use of the WikiExtractor tool developed by the Apertium project.

4.3 Tools and Libraries

WikiExtractor is a tool used for extracting text data from dumps of wikipedia formatted data. This tool removes formatting and links and outputs wikipedia articles as close to raw text as possible. However it does not split articles into separate files, so all articles end in a huge text file.

spaCy is an open-source library for advanced natural language processing. It is written in Python and Cython and published under the MIT license. Unlike other libraries within NLP which focuses on teaching and research, spaCy is designed for production usage. spaCy features convolutional neural network (CNN) models for part-of-speech tagging, dependency parsing, text categorization and named entity recognition (NER). It also comes with prebuilt statistical neural network models and NER models for 17 languages and tokenization for 65 languages

In the beginning, in order to extract the triples for the knowledge graph I used the spaCy library.

Spacy can be used to build information extraction systems or natural language understanding systems. However spacy is not a platform nor an API. It is not a chat bot engine and is not designed spechifically for chat purposes. And at last spacy is not research software, therefore it takes different design decisions than its competitors NLTK and CoreNLP. [16]

From the miriad of features of spacy the following are worth using for this project:

Feature	**Description**
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) Tagging	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Sentence Boundary Detection	Finding and segmenting individual sentences.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions

Table 4.1: Features of spacy with potential for this project.

Spacy offers trained pipelines for a variety of languages. The languages can be installed on a as-needed basis. The language packages vary in size, speed, memory usage, amount of data and accuracy.

The trained pipelines include:

- Binary weights - estimated values based on the model examples
- Lexical entries - words from vocabulary and their context-independent attributes like shape and spelling
- Data files - lemmatization rules and lookup tables
- Word vectors - multi-dimensional meaning representation of words used to determine similarity
- Configuration options - used to configure spacy for the best outcome

displacy is an open source dependency parse tree visualizer that helps with drawing dependencies between words in a sentence. This framework is built on technologies like JavaScript, CSS and SVG.

textacy is a python library that helps in performing many natural language processing tasks, built on top of the spaCy library. Among its features it includes functionality for cleaning, normalizing and exploring raw text before processing it with spaCy; extracting structured information from processed documents, n-grams, entities, acronyms, keyterms, and SVO triples; tokenize and vectorize documents; etc.

clauCy is a python implementation of the ClausIE [10] that works with spacy. ClausIE is a clause-based information extraction that extracts relations and clauses from natural language text. It exploits linguistic knowledge about the English grammar to detect clauses in the input sentences.

stanza is a python library that wraps around Standford CoreNLP Java Toolkit. Stanza is acting as a client while CoreNLP is the server. Using this architecture Stanza directs python calls to the CoreNLP server and gets the response back to the python environment.

CoreNLP is a Natural Language Processing toolkit [17] developed at Standford University in California. It is simmilar to spacy but it has its main focus on the academic world.

Neo4j is a graph database management system featuring and ACID-compliant transactional database with native graph storage and processing. Neo4j is implemented in Java and offers an API for use with other languages, namely the Cypher Query Language, over a transactional HTTP endpoint or through the binary bolt protocol.

Jupyter Notebook is a community driven project that aims at producing open-source software, open-standards and services for interactive computing over a variety of programming language. I make use of the python version of Jupyter Notebook in order to interactively access and edit the code used in this project.

4.4 Implementation

At the beginning we initialize the dependencies of our little triple-extractor program. We need spacy and its english language model for segmenting text into sentences and stanza with its CoreNLPClient in order to extract triples from sentences.

```
1  # IMPORT PACKAGE DEPENDENCIES.
2  # Import the spacy language library
3  import spacy
4
5  # Import the english language module
6  from spacy.lang.en import English
7
8  # The language model used.
9  nlp = spacy.load('en_core_web_lg')
10
11 # Stanza - client fore Standford CoreNLP
12 from stanza.server import CoreNLPClient
```

Listing 4.1: The import definitions at the start of the program

Next we define the sentence extraction function that makes use of the spacy library.

Note that because of extra whitespace between sentences the spacy sentencizer produces empty sentences which are removed in 4.2.

```
def extract_sentences(text) :
    nlp = English()
    nlp.add_pipe('sentencizer')
    doc = nlp(text)
    sentences = [str(sent).strip() for sent in doc.sents]
    # using remove() to perform removal of empty sentences.
    while("" in sentences) :
        sentences.remove("")
    return sentences
```

Listing 4.2: The sentence extraction function.

Then we have the raw triple extraction using stanza connected to CoreNLP's OpenIE using the included CoreNLPClient.

```

# Extacts SVO-triples from a given sentence.
def extractGraphElements(sentence) :
    with CoreNLPClient(annotators="openie,coref,ner", be_quiet=False) as client:
        ann = client.annotate(sentence, properties={"openie.triple.strict":"true", "
   openie.resolve_coref":"true"})
        triples = []
        # Extracting all matching triples from the phrase.
        for annotated_sentence in ann.sentence:
            for triple_candidate in annotated_sentence.openieTriple:
                triples.append({"confidence": triple_candidate.confidence, "subj" :
   triple_candidate.subject, "rel": triple_candidate.relation, "obj":
   triple_candidate.object})
    return triples
```

Listing 4.3: The raw triple extraction function.

So at line 4 in 4.3 we initialize a CoreNLPClient using the following modules: "openie", "coref" and "ner".

OpenIE stands for Open Information Extraction and deals mostly with extracting SVO-triples from text.

Coref stands for "co-reference" and helps with identifying pronouns and the noun they refer to. For example in the sentence "Obama is the president of USA, he was born in Hawaii." with the help of "coref" module the pronoun "he" would be replaced with "Obama".

NER stands for Named Entity Recognition and is a module for recognizing and extracting named entities like proper names of persons and companies.

Furthermore in 4.3 we take a sentence and extract all the triple candidates (multiple triple pairs for each sentence clause). The triple candidates are reorganized in a list of dictionaries structures. We keep the subject, relation, object, and the confidence score.

```
1 # Score triple elements based on the frequency of occurence.
2 # Then select one relevant triple per sentence.
3 def scoreAndSelectTriples(triples) :
4     # Initialize variables
5     subj = {}
6     rel = {}
7     obj = {}
8     elements = ['subj', 'rel', 'obj']
9     scored_triples = []
10    selected_triple = { 'score' : 0, 'obj': '' }
11    selected_triples = []
12    # Giving each element of the triple a score
13    for index, triple in enumerate(triples):
14        triples[index]['subj_score'] = 0
15        triples[index]['rel_score'] = 0
16        triples[index]['obj_score'] = 0
17        # Iterate through each triple element (subj,rel and obj)
18        for element in elements:
19            score_key = element + '_score'
20            if not hasattr(locals()[element], triple[element]):
21                # score is defined as the amount of identical elements present in the
     triple array
22                score = len([item for item in triples if item[element] == triple[
    element]])
23                locals()[element][triple[element]] = score
24                triples[index][score_key] = score
25                # total score is the sum of subj, rel and obj scores
26                triples[index]['score'] = triples[index]['subj_score'] + triples[
    index]['rel_score'] + triples[index]['obj_score']
27                # selecting one triple per sentence based on total score, the
    difference between rel and obj score, length of obj and confidence score
28                if(selected_triple['score'] <= triples[index]['score'] and triples[
    index]['rel_score'] >= triples[index]['obj_score'] and len(triples[index]['obj'])
     > len(selected_triple['obj']) and triples[index]['confidence'] == 1):
29                    selected_triple = triples[index]
30    # filter empty triples from the selected ones.
31    if selected_triple['score'] > 0:
32        selected_triples.append(selected_triple)
33
34    return selected_triples
```

Listing 4.4: The triple selection function.

In 4.4 we take the previously extracted triples, and we score them according to the frequency of occurrence of every element of the triple within the entire set of the triples.

For example from the sentence: "Earth science is an all-embracing term for the sciences related to the planet Earth." the following triples are extracted:

```
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is term for', 'obj': 'sciences related'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is', 'obj': 'embracing term'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is all embracing term for', 'obj': 'sciences'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is term for', 'obj': 'sciences related to planet Earth'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is embracing term for', 'obj': 'sciences related to planet Earth'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is', 'obj': 'all embracing term'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is', 'obj': 'term'},
{'confidence': 1.0, 'subj': 'science', 'rel': 'is embracing', 'obj': 'sciences'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is embracing term for', 'obj': 'sciences'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is embracing term for', 'obj': 'sciences related'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is all embracing term for', 'obj': 'sciences related'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is term for', 'obj': 'sciences'},
{'confidence': 1.0, 'subj': 'Earth science', 'rel': 'is all embracing term for', 'obj': 'sciences related to planet Earth'}
```

Listing 4.5: An example of extracted triples.

From all of these triples only one will be selected and stored in the database. To find the most relevant one we score each of the triple in relation with the others. So in the aforementioned example we count how many time a subject like "Earth science" appears in the list of triples and we give it a score. In the example the subject score will be 12. We do the same for each relation and each object.

In order to store relations in a graph database we use the following functions:

```python

from neo4j import GraphDatabase

driver = GraphDatabase.driver("neo4j://localhost:7687", auth=("user", "password"))

def add_triple(tx, sentence_subject, sentence_predicate, sentence_object):
    query = """
            MERGE (a:Concept {name: $concept})
            MERGE (b:Concept {name: $explaination})
            MERGE (a)-[:#relation#]->(b)
            """
    # Insert relation into the query.
    relation = sentence_predicate.upper()
    relation = relation.replace(" ", "_")
    relation = ''.join([i for i in relation if i.isalpha()])
    query = query.replace("#relation#", relation)
    tx.run(query, concept=sentence_subject, explaination=sentence_object)


def save_triple(triple):
    print(triple)
    with driver.session() as session:
        session.write_transaction(add_triple, triple['subj'], triple['rel'], triple['obj'])
```

Listing 4.6: Connecting to the neo4j database.

So firstly we use the driver to connect to the database then we have have a function (add_triple) that prepares a database query and then another fuction (save_triple) that commits the query to the database.

When reading data from text files all the code mentioned in this section is ran for each file. The glue code that takes care of calling each function is as follows:

```

fileObject = open("Earth science.txt", "r")
data = fileObject.read()
sentences = extract_sentences(data)
for sentence in sentences:
    triples = extractGraphElements(sentence)
    triples = scoreAndSelectTriples(triples)
    for triple in triples:
        if triple:
            print(triple['subj'], '-', triple['rel'], '-', triple['obj'])
            save_triple(triple)

driver.close()
```

Listing 4.7: The main code that starts every function.

So first we open the input file for reading, then we extract sentences from the text within the file. Afterwards we extract and score triples for each sentence, then we commit each of the triples to the Neo4j database. And in the end we close the connection to the database.

4.4.1 Triple example

Here we will closely follow the extraction of several triples from the original sentence to their final form as stored in the graph database.

```
Earth science is an all-embracing term for the sciences related to the planet Earth.
```

Listing 4.8: Example sentence from the "Earth science" wikipedia article.

The sentence from listing 4.8 is read from a text file that was extracted from the Simple English wikipedia. It was then parsed with the Standford CoreNLP Open IE and the following triples were proposed:

```

'Earth science', 'is term for', 'sciences related'
'Earth science', 'is', 'embracing term'
'Earth science', 'is all embracing term for', 'sciences'
'Earth science', 'is term for', 'sciences related to planet Earth'
'Earth science', 'is embracing term for', 'sciences related to planet Earth'
'Earth science', 'is', 'all embracing term'
'Earth science', 'is', 'term'
'science', 'is embracing', 'sciences'
'Earth science', 'is embracing term for', 'sciences'
'Earth science', 'is embracing term for', 'sciences related'
'Earth science', 'is all embracing term for', 'sciences related'
'Earth science', 'is term for', 'sciences'
'Earth science', 'is all embracing term for', 'sciences related to planet Earth'
```

Listing 4.9: The proposed triples as extracted by Standford CoreNLP OpenIE

After we have the triples returned from Standford CoreNLP we put them through the scoring function and the result is as follows:

```

'Earth science': 12, 'is term for': 3, 'obj': 'sciences related': 3, 'score': 18
'Earth science': 12, 'is': 3, 'embracing term': 1, 'score': 16
'Earth science': 12, 'is all embracing term for': 3, 'sciences': 4, 'score': 19
'Earth science': 12, 'is term for': 3, 'sciences related to planet Earth': 3, 'score
    ': 18
'Earth science': 12, 'is embracing term for': 3, 'sciences related to planet Earth':
    3, 'score': 18
'Earth science': 12, 'is': 3, 'all embracing term': 1, 'score': 16
'Earth science': 12, 'is': 3, 'term': 1, 'score': 16
'science': 1, 'is embracing': 1, 'sciences': 4, 'score': 6
'Earth science': 12, 'is embracing term for': 3, 'sciences': 4, 'score': 19}, 'Earth
    science': 12, 'is embracing term for': 3, 'sciences related': 3, 'score': 18
'Earth science': 12, 'is all embracing term for': 3, 'sciences related': 3, 'score':
    18
'Earth science': 12, 'is term for': 3, 'sciences': 4, 'score': 19
'Earth science': 12, 'is all embracing term for': 3, 'sciences related to planet
    Earth': 3, 'score': 18
```

Listing 4.10: The scored triples

As we can see there are 3 pair of triples that have the maximum score of 19:

```

'Earth science': 12, 'is all embracing term for': 3, 'sciences': 4, 'score': 19
'Earth science': 12, 'is embracing term for': 3, 'sciences': 4, 'score': 19,
'Earth science': 12, 'is term for': 3, 'sciences': 4, 'score': 19
```

Listing 4.11: The triples with the highest scores.

4.5 Querying the graph database

As the database used is a neo4j graph database we need to use the cipher query language in order to extract information from the database.

One could start as simple as checking if the database contains an entity. This is done with the following query:

```
  MATCH (concept:Concept {name: "Japan"})
  RETURN concept
```

Listing 4.12: Retrieving an entity from the database (if it exists)

This query will return the matching entity if it exists.

```
__________________
|"concept"       |
|________________|
|{"name":"Japan"}|
|________________|
```

Listing 4.13: An example of entity result)

Then in order to return all entities we have a simple query as follows:

```
  MATCH (n)
  RETURN n
```

Listing 4.14: Retrieving all entities from the database

Figure 4.2: The graph representation of the result of retrieving query

{"name":"Japan"}
{"name":"war"}
{"name":"small fishing village"}
{"name":"Edo"}
{"name":"old Musashi Province"}
{"name":"walls"}
{"name":"century"}

Figure 4.3: The table representation of the result of retrieving query

The result can either be viewed as graph, as shown in figure 4.2, or as a classic table, as shown in figure 4.3.

Some more advanced queries would be finding the relations between two nodes. For this example we would take the entities "Japan" and "Olympic Games".

```
MATCH (a:Concept{name:'Japan'})-[r]-(b:Concept{name:'Olympic Games'})
RETURN r
```

Listing 4.15: Retrieving a relationship between two entities

The result is as follows

```
{
  "identity": 161,
  "start": 4,
  "end": 251,
  "type": "HAS_TAKEN_PART_IN",
  "properties": {
  }
}
```

Listing 4.16: The resulting relationship

In order to find the nodes between two nodes a more complex query needs to be created.

```
MATCH p1 = (n1:Concept{name: "Japan"})-[r1*..5]-(n2:Concept),
      p2 = (n2:Concept)-[r2*..5]-(n3:Concept {name:"Tokyo"})
RETURN n1,n2,n3,r1,r2
```

Listing 4.17: Query to find entities between two nodes

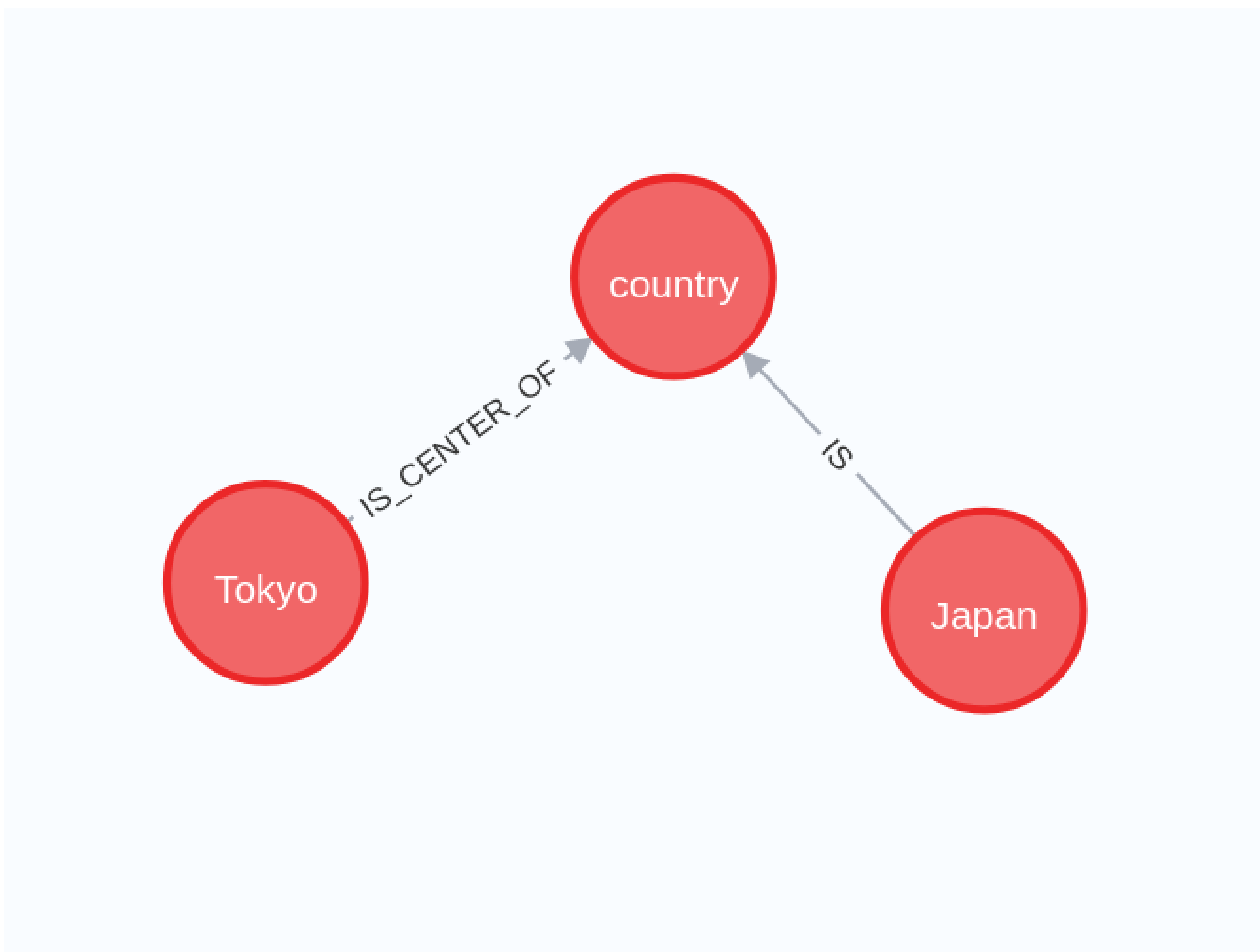

Figure 4.4: The graph representation of entities between two nodes

4.6 Further work

There are multiple ways to improve the triple extraction process. Among them are:

- lemmatize before triple extraction
- employing NER - Named Entity Recognition
- scoring function - improve the triple selection algorithm
- coref - replace pronouns with their counterparts

Lemmatization

With lemmatization we reduce the word forms at their dictionary forms by getting rid of inflections. Having this will improve the results of Standford CoreNLP which sometimes has a hard time to distinguish between inflected verbs and nouns.

For example with the triple ("Japan", "was", "opened for visitors again") after applying lemmatization we should opbtain a triple as follows ("Japan", "is open", "for visitiors again"). So the verbs from the object side of the triple would be moved to the relation.

There would be cases where the lemmatization would not be advised such as the words "writing system" should not be converted to "write system". If the dictionary used is comprehensive enough this should not be a problem.

NER - Named entity recognition will help defining precise objects or subjects as multiple words would be represented as one token. For example "New York" will stand as one token and will not be split in two by the OpenIE algorithm.

Scoring function selects which one of the proposed triples should be chosen as the final triple stored in the database. The scoring function should have more criteria for selecting triples. At the moment the triple selection is solely based on statistical measurements of occurence.

Coref - co-reference takes care of substituding the nouns with the entities they are replacing. The co-reference module used with Standford OpenIE can only substitute nouns that are present in the same sentence or phrase. A better co-reference module will do this job spanning across a paragraph or maybe even an entire document.

Chapter 5

Experiments

The experiments I made in the beginning of the project were trying different libraries for extracting triples. One of them was textacy which is built on top of spacy. What I found was that it refused to extract triples from certain complex sentences. I believe the reason for that was because there were multiple predicates in those sentences and hece the main predicate was difficult to extract. I don't know how textacy works under the hood, but my experience was unsatisfactory so therefore I decided to avoid the library altoghether.

Another thing I tried was a library used for extracting chunks. The library was NLTK - Natural Language Toolkit - which is able to perform sentence splitting into chunks. Chunks are not the same as triples, chunks are tokens that are related to eachother like named entities.

After I tested more complex sentences with my own implementation for extracting triples I noticed that the quality of extracted triples was not satisfactory and then I tried to simplify the text at hand with a library called claucy.

Claucy is a clause extractor and sentence simplifier. The latest version of claucy at the moment of writing was built on top of spacy 2.3.0 which is an older version of the spacy library. Spacy is at 3.1.4 at the time of writing and there were significant changes from spacy2 to spacy3.

However, theres is a differenece in the implementation of claucy for spacy2 and the implementation of claucy for spacy3. For example with the model sentence *«A cat, hearing that the birds in a certain aviary were ailing dressed himself up as a physician, and, taking his cane and a bag of instruments becoming his profession, went to call on them.»* there is a difference in the extracted sentences. Claucy for spacy2 extracts the following:

```
['The birds were ailing.']
['A cat dressed himself as a physician.', 'A cat dressed himself.']
['A cat took his cane.', 'A cat took a bag.']
['A cat became his profession.']
['A cat went.']
['A cat called on them.']
```

Listing 5.1: The output of the model sentence after it was split in clauses by claucy for spacy2

Whie Claucy for spacy3 extracts the following:

```
['the birds were ailing']
['the birds dressed himself as a physician', 'the birds dressed himself']
['A cat went']
['A cat called on them']
```

Listing 5.2: The output of the model sentence after it was split in clauses by claucy for spacy3

As we can see when we compare the output from 5.1 to 5.2 we see some missing clauses and clause number 2 and number 3 are wrong in the output of the library for spacy3. The subject was not determined correctly with the latter library.

5.1 Triple extraction with spacy

At first I prepared all the needed libraries by installing them on the system at hand and then importing them into the python environment. I use python because it is the most common language/environment within the academia and study of NLP matters.

```
# IMPORT PACKAGE DEPENDENCIES.
# The main library.
import spacy
# A library for drawing dependencies.
from spacy import displacy
# Pattern matcher library-
from spacy.matcher import Matcher

# The language model used.
nlp = spacy.load('en_core_web_lg')

# The clause extraction library used.
import claucy
claucy.add_to_pipe(nlp)
```

Listing 5.3: The import definitions at the start of the program

So in listing 5.3 we import spacy with its submodules displacy and Matcher so we have them ready for later use. Then we load the Large English Web Models. And in the end we load claucy and add it to the pipeline.

Then in listing 5.4 we prepare the Matcher with our language vocabulary.

```
# Provide the loaded vocabulary to the matcher library.
matcher = Matcher(nlp.vocab)
```

Listing 5.4: Preparing the Matcher by initializing it with our language vocabulary.

Next in listing 5.5 we are doing sentence segmentation. We divide the text into sentences by using spacy functionality.

This is done by loading the English model and adding the sentencizer module to the pipeline. Note that we remove empty sentences that occur when we have double spaces or new lines at the end of other sentences.

```

def extract_sentences(text) :
    nlp = English()
    nlp.add_pipe('sentencizer')
    doc = nlp(text)
    sentences = [str(sent).strip() for sent in doc.sents]
    # using remove() to perform removal of empty sentences.
    while("" in sentences) :
        sentences.remove("")
    return sentences
```

Listing 5.5: Function for extracting sentences using the spacy library.

Next we define utility functions for extracting the subject-predicate-object triples from the sentences.

```python

# Extracts just the subject of a given sentence.
def extractSubject(sentence) :
    parsed_sentence = nlp(sentence)
    matcher = Matcher(nlp.vocab)
    # Matches composite subject
    # (all modifiers and the punctuation + subject)
    pattern = [
        {"DEP" : {"IN" : ["compound", "nummod", "npadvmod"]}, "OP": "+",
         "SPACY": True},
        {"DEP" : {"IN" : ["nsubj", "nsubjpass"] } }
    ]
    matcher.add("MultiWordSubj", [pattern])
    matches = matcher(parsed_sentence)
    # If we have no match for a composite subject we try a simple,
    # non-composite one
    if not matches:
        pattern2 = [
            {"DEP":"nsubj"}
        ]
        matcher.add("SingleWordSubj", [pattern2])
        matches = matcher(parsed_sentence)
    #displacy.render(parsed_sentence, style='dep',jupyter=True)
    for tok in parsed_sentence:
        print(tok.text, "...", tok.dep_)
    for match_id, start, end in matches :
        span = parsed_sentence[start:end]
        return span
```

Listing 5.6: Function for extracting subject of the sentence using the Matcher library.

In listing 5.6 we start by initializing the sentence in regards to pos-tagging and syntactic dependencies and matcher. Then we define a pattern that matches compound subjects based on the syntactic dependencies that we have available from the dependency parsing step.

Then if we are unsucessful at finding a composite subject we extract a simple one.

In the end we return the part of the sentence that contains the simple or composite subject.

Then we proceed by extracting the predicate of the sentence using the same Matcher library.

```python
# Extracts just the verb of a given sentence.
def extractVerb(sentence) :
    parsed_sentence = nlp(sentence)
    matcher = Matcher(nlp.vocab)
    # Extract only the main verb of the sentence.
    pattern = [
        {"DEP" : "ROOT"}
    ]

    matcher.add("SingleWordVerb", [pattern])
    matches = matcher(parsed_sentence)
    for match_id, start, end in matches :
        span = parsed_sentence[start:end]
    return span
```

Listing 5.7: Function for extracting predicate of the sentence using the Matcher library.

Listing 5.7 is a simpler version of the subject extraction. Here the pattern matcher only maches the verb (root of the sentence).

And finally we extract the object of the sentence using the Matcher library.

```
# Extracts the object of a given sentence.
def extractObject(sentence):
    parsed_sentence = nlp(sentence)
    matcher = Matcher(nlp.vocab)

    # Matches composite object (modifiers and objects close to eachother)
    pattern = [
        {"DEP" : "amod", "OP" : "*"},
        {"DEP" : {"IN" : [ "pobj", "dobj", "oprd"] } }
    ]
    matcher.add("MultiWordObj", [pattern])
    matches = matcher(parsed_sentence)
    span = ""
    for match_id, start, end in matches :
        span = parsed_sentence[start:end]
    return span
```

Listing 5.8: Function for extracting the object of the sentence using the Matcher library.

The latest extraction based on pattern matching is done to find either a composite object or a simple one. This is done in listing 5.8.

And in the end we have a function that extracts the triple by using the aforementioned functions.

```
# Extacts a triple SVO from a given sentence.
def extractGraphElements(sentence) :
    sentenceSubject = extractSubject(sentence)
    sentenceVerb = extractVerb(sentence)
    sentenceObject = extractObject(sentence)
    if (not sentenceSubject) or (not sentenceVerb)
        or (not sentenceObject):
        return []
    else:
        return [sentenceSubject, sentenceVerb, sentenceObject]
```

Listing 5.9: Function for extracting the triple of the sentence.

In listing 5.9 we call the previous functions that extract a subject, predicate and an object and then a triple is returned.

Chapter 6

Discussion

6.1 Triple selection

Triple selection is the process of filtering the triples resulted from triple extraction. It is a difficult and inconsistent process.

In a perfect world one would use a specificity index that would select the most specific triple from the matched triples. But since a specificity index does not yet exist one could use various statistical methods to choose the most relevant triple.

In this paper I make use of an statistical approach by scoring the triples according to their element frequency. An element is a part of the triple, thus either a subject, relation or object. The triple with the highest score is usually selected.

However problems arise when there are multiple triples with the same highest score. When this happens another strategy needs to be employed.

6.2 Co-reference

Co-reference is the task that takes care of substituting pronouns with the entity they represent in a sentence. When applying coref together with OpenIE there is a limitation to the boundaries of sentences. But in English texts pronouns usually are not constrained by the boundaries of the sentences. The pronoun and its replacement could be present several sentences apart, usually in the same paragraph. Sometimes within the same document.

One would obtain better results if the task of coref would be applied before doing OpenIE and with a larger scope. In the case at hand where the documents are wikipedia articles I would have obtained better results If I had a coref task applied on the entire wikipedia article before starting open information extraction.

6.3 Text simplification

Text simplification turns to be a hard task, but if performed well it would significantly improve the precision of the extracted triples.

Rule-based Relation Extraction would have the most gain from simplified text as the rules tend to be based on sequence patterns.

Chapter 7

Conclusion

Knowledge graphs help with structuring information in a way that can be understood by machines. Extracting information from text is a difficult problem within the realm of Natural Language Processing.

The problem at hand can be solved in numerous ways and Open Information Extraction is one of them. As we see throughout this paper, the accuracy of the results depends on the pipeline used and the simplicity of the text at hand.

Two processes that would help enhance the solution further are clause extractionn and text simplification but their implementation is not an easy and problem free task.

The problem domain, Earth science, has its complexity that directly impacts the result of triple extraction and hence the information that reaches the graph database.

Now I will answer the questions I started with in Problem Formulation.

- **Which techniques are best suited to extract new knowledge?**

There are multiple teqniques and steps that need to be completed in order to extract knowledge from text. At the incipient stage of the process one needs to complete corpora cleaning tasks then text preprocessing (stemming, lemmatization, etc.) and finally relationship extraction techniques. Among the techniques employed in relationship extraction each one has pros and cons as shown in 2.5 so one has to choose what is best suited for the project at hand.

- **How to represent the extracted knowledge as a graph?**

When using a graph database the graph representation is usually provided by the database software. One needs only to provide the data represented as triple of the form (subject, predicate, object).

- **How to query the knowledge graph in order to answer questions?**

In section 4.5 querying the knowledge graph has been explored. In order to answer questions one has to craft a query according to the question. In simple questions two elements of the triple would be present and the third one can be found with a single query. Whith more complex question, answering strategies need to be defined.

And finally the main question of the thesis.

Can text represented as a knowledge graph be used to feed knowledge to a computer-aided personal assistant?

While the compter-aided personal assistant software was not yet explored during this project, one could either build one or use an existing one to extract data from a knowledge graph.

As shown in section 4.5 it is easy to query the graph database. In this case I used neo4j database and the cipher query language, but any other graph database/query language can be used.

Bibliography

[1] Allen Newell, J. C. Shaw, and Herbert A. Simon. "Report on a general problem-solving program". In: (1959), pp. 256–264.

[2] Shaoxiong Ji, Shirui Pan, Erik Cambria, et al. "A Survey on Knowledge Graphs: Representation, Acquisition and Applications". In: *CoRR* abs/2002.00388 (2020). arXiv: 2002.00388. URL: https://arxiv.org/abs/2002.00388.

[3] Quoc V. Le and Tomás Mikolov. "Distributed Representations of Sentences and Documents". In: *CoRR* abs/1405.4053 (2014). arXiv: 1405.4053. URL: http://arxiv.org/abs/1405.4053.

[4] Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, et al. "Freebase: a collaboratively created graph database for structuring human knowledge". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. Ed. by Jason Tsong-Li Wang. ACM, 2008, pp. 1247–1250. DOI: 10.1145/1376616.1376746. URL: https://doi.org/10.1145/1376616.1376746.

[5] Christian Bizer, Jens Lehmann, Georgi Kobilarov, et al. "DBpedia - A crystallization point for the Web of Data". In: *J. Web Semant.* 7.3 (2009), pp. 154–165. DOI: 10.1016/j.websem.2009.07.002. URL: https://doi.org/10.1016/j.websem.2009.07.002.

[6] Denny Vrandecic and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase". In: *Commun. ACM* 57.10 (2014), pp. 78–85. DOI: 10.1145/2629489. URL: https://doi.org/10.1145/2629489.

[7] Mike Mintz, Steven Bills, Rion Snow, et al. "Distant supervision for relation extraction without labeled data". In: *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*. Ed. by Keh-Yih Su, Jian Su, and Janyce Wiebe. The Association for Computer Linguistics, 2009, pp. 1003–1011. URL: https://aclanthology.org/P09-1113/.

[8] Chris Quirk and Hoifung Poon. "Distant Supervision for Relation Extraction beyond the Sentence Boundary". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*. Ed. by Mirella Lapata, Phil Blunsom, and Alexander Koller. Association for Computational Linguistics, 2017, pp. 1171–1182. DOI: 10.18653/v1/e17-1110. URL: https://doi.org/10.18653/v1/e17-1110.

[9] Eugene Agichtein and Luis Gravano. "*Snowball*: extracting relations from large plain-text collections". In: *Proceedings of the Fifth ACM Conference on Digital Libraries, June 2-7, 2000, San Antonio, TX, USA*. ACM, 2000, pp. 85–94. DOI: 10.1145/336597.336644. URL: https://doi.org/10.1145/336597.336644.

[10] Luciano Del Corro and Rainer Gemulla. "ClausIE: clause-based open information extraction". In: *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*. Ed. by Daniel Schwabe, Virgilio A. F. Almeida, Hartmut Glaser, et al. International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 355–366. DOI: 10.1145/2488388.2488420. URL: https://doi.org/10.1145/2488388.2488420.

[11] *The Standford Natural Language Processing Group*. URL: https://nlp.stanford.edu/.

[12] Christopher D. Manning, Mihai Surdeanu, John Bauer, et al. “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*. The Association for Computer Linguistics, 2014, pp. 55–60. DOI: 10.3115/v1/p14-5010. URL: https://doi.org/10.3115/v1/p14-5010.

[13] *The Standford OpenIE*. URL: https://nlp.stanford.edu/software/openie.html.

[14] 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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. The Association for Computer Linguistics, 2015, pp. 344–354. DOI: 10.3115/v1/p15-1034. URL: https://doi.org/10.3115/v1/p15-1034.

[15] Huong Duong To and Phuc Do. “Extracting triples from Vietnamese text to create knowledge graph”. In: *12th International Conference on Knowledge and Systems Engineering, KSE 2020, Can Tho City, Vietnam, November 12-14, 2020*. IEEE, 2020, pp. 219–223. DOI: 10.1109/KSE50997.2020.9287471. URL: https://doi.org/10.1109/KSE50997.2020.9287471.

[16] *spaCy 101: Everything you need to Know - SPACY usage documentation*. URL: https://spacy.io/usage/spacy-101#whats-spacy.

[17] Christopher D. Manning, Mihai Surdeanu, John Bauer, et al. “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*. The Association for Computer Linguistics, 2014, pp. 55–60. DOI: 10.3115/v1/p14-5010. URL: https://doi.org/10.3115/v1/p14-5010.