BUILDING AN ALBANIAN KNOWLEDGE GRAPH FROM UNSTRUCTURED TEXT

Hakan Shehu

Follow this and additional works at: https://knowledgecenter.ubt-uni.net/etd

Part of the Computer Sciences Commons
BUILDING AN ALBANIAN KNOWLEDGE GRAPH FROM UNSTRUCTURED TEXT

Bachelor degree

Hakan Shehu

December / 2017

Prishtina
Diploma thesis
Academic year 2014 – 2015

Hakan Shehu

BUILDING AN ALBANIAN KNOWLEDGE GRAPH FROM UNSTRUCTURED TEXT

Mentor: Dr. Bertan Karahoda

December / 2017

This thesis is worked and submitted in fulfillment of partial requests for Bachelor Degree
ABSTRACT

We live in the golden area of information. The World Wide web contains a vast amount of unstructured text in different digital formats, including newswire, blogs, email communications, governmental documents, chat logs, and so on. Some of the biggest companies and organizations have created knowledge bases which represent a semantic network of facts, entities and relations between them. Even though this area has been well researched for a considerable time, there is a lack of implementation of such a knowledge extraction for Albanian language. In this thesis we will try to create an Albanian general knowledge graph from unstructured text. The existing state of the art proposals for relations extraction in other languages will be reviewed and used. We will present the process of creating a knowledge base using some of natural language processing techniques, graph modeling, storing and retrieving. Finally, we will discuss important potential applications of such a knowledge base in industry and academia.
ACKNOWLEDGEMENTS

This thesis would not be possible without the work of a large number of people in different fields, providing the tools and information that was used to implement such an idea.

I would like to thank all the contributors in Wikipedia articles, who voluntarily have provided a lot of facts and information in Albanian language, data which were the primary resource for creating the knowledge graph. Another big gratitude goes for all researchers and open source contributors who provided very important tools which made work so much easier and accurate.

Special thanks go for my mentor of the thesis Dr. Bertan Karahoda, his insights and advices helped me a lot during the research and implementation phase. Huge thanks go to my other mentor Dr. Ercan Canhasi, whose extended expertise and support was an important motivation in pursuing this project, and whose previous works in Albanian natural language processing were a key asset, without which this project could not be implemented.

Also, I would like to thank my parents and my brother for their unconditional love and support through my entire life.
**TABLE OF CONTENTS**

LIST OF FIGURES ........................................................................................................................................ IV
LIST OF TABLES ........................................................................................................................................ IV
LIST OF CODE SNIPPETS .......................................................................................................................... V
LIST OF ABBREVIATIONS ............................................................................................................................ VI

1. INTRODUCTION ........................................................................................................................................ 1

2. LITERATURE REVIEW ............................................................................................................................... 3
   2.1. Natural Language Processing .................................................................................................................. 3
      2.1.1. Part of speech tagging .................................................................................................................. 3
      2.1.2. Sentence breaking ......................................................................................................................... 4
      2.1.3. Stemming ....................................................................................................................................... 4
      2.1.4. Word segmentation ....................................................................................................................... 5
      2.1.5. Named Entity Recognizer (NER) .................................................................................................. 5
      2.1.6. Term Frequency Inverse Document Frequency (TF-IDF) .............................................................. 5
   2.2. Graph .................................................................................................................................................. 6
      2.2.1. Graph databases ........................................................................................................................... 6
   2.3. Relation extraction ............................................................................................................................... 7
   2.4. Knowledge base construction ............................................................................................................... 9

3. PROBLEM DEFINITION ............................................................................................................................ 11

4. METHODOLOGY ....................................................................................................................................... 12

5. IMPLEMENTATION .................................................................................................................................... 13
   5.1. Pre-processing .................................................................................................................................... 13
   5.2. Relation extraction ............................................................................................................................. 19
      5.2.1. Subject extraction .......................................................................................................................... 21
      5.2.2. Predicate extraction .................................................................................................................... 21
      5.2.3. Object extraction ........................................................................................................................ 23
   5.3. Storing nodes and relations .................................................................................................................. 26
   5.4. Retrieving information from knowledge graph .................................................................................... 28

6. APPLICATIONS ......................................................................................................................................... 33

7. CONCLUSIONS ......................................................................................................................................... 36

8. REFERENCES .............................................................................................................................................. 38
LIST OF FIGURES

Figure 1: Knowledge Graph Example (Source: Google) .......................................................... 1
Figure 2: An example of Named Entity Recognition in raw text ........................................... 5
Figure 3: Small Facebook social network ............................................................................. 6
Figure 4: Graph database example ......................................................................................... 7
Figure 5: Relation triplet illustration ....................................................................................... 8
Figure 6: Graph example from Freebase data .......................................................................... 10
Figure 7: Small part of Albanian Knowledge Graph (only 300 nodes) ........................................ 28
Figure 8: All nodes and relations for the article about "Lidhja e Prizrenit" ............................. 29
Figure 9: Simple is-a relation ................................................................................................. 29
Figure 10: Nodes that have a relation with the node Hashim Thaqi ........................................... 30
Figure 11: Another simple relation illustration ........................................................................... 30
Figure 12: Some of the capitals nodes related with Prishtina ................................................... 31
Figure 13: People born in 20th of May, displayed in a graph .................................................... 32
Figure 14: Google search results using knowledge graph ......................................................... 33
Figure 15: Text summarization illustration ............................................................................... 34

LIST OF TABLES

Table 1: Definition and example of commonly occurring parts of speech ................................. 4
Table 2: Most common node labels in knowledge graph ........................................................... 36
Table 3: Most common relations types in knowledge graph ....................................................... 37
LIST OF CODE SNIPPETS

Code Snippet 1: Parsed Wikipedia Article JSON Format Example .................................................. 13
Code Snippet 2: Apache OpenNLP sentence detecting ................................................................. 14
Code Snippet 3: Method for finding named entities inside the content of an article using Stanford NER ............................................................................................................................................. 15
Code Snippet 4: Pre-processed article model in JSON ........................................................................ 16
Code Snippet 5: TFIDF Token class ................................................................................................ 17
Code Snippet 6: Method for calculating word frequencies and create TF-IDF tokens ...................... 18
Code Snippet 7: TF-IDF score calculation for each token ................................................................. 19
Code Snippet 8: Method for calculating TF-IDF max count for each article .................................... 19
Code Snippet 9: Graph node class representation ............................................................................. 20
Code Snippet 10: Graph relation class representation ................................................................. 20
Code Snippet 11: Triplet wrapper class representation .................................................................... 20
Code Snippet 12: Subject extraction method .................................................................................... 21
Code Snippet 13: Predicate extraction algorithm ............................................................................ 22
Code Snippet 14: TF-IDF score normalization based on distance .................................................. 23
Code Snippet 15: Object extraction method ..................................................................................... 24
Code Snippet 16: Final triplet extraction method ........................................................................ 25
Code Snippet 17: Inserting relation triplets in graph database from Java ........................................ 27
Code Snippet 18: Node inserting into Neo4j using jCypher for Java ............................................. 27
LIST OF ABBREVIATIONS

AI – Artificial Intelligence
NLP – Natural Language Processing
POS – Part of Speech Tagging
SQL – Structured Query Language
NER – Named Entity Recognizer
TF-IDF – Term Frequency – Inverse Document Frequency
CRUD – Create, Read, Update, Delete
OLTP - Online transaction processing
NoSQL - Non-Relational
KB – Knowledge base
XML - eXtensible Markup Language
JSON - JavaScript Object Notation
1. INTRODUCTION

A knowledge graph represents a collection of connected entities and their attributes (Gomez-Perez, Pan, Vetere, & Wu, 2017). The term Knowledge Graph became famous in 2012 since Google posted the famous blog article “Introducing the Knowledge Graph: things, not strings” (Singhal, 2012). But, this is not the beginning of it, because the idea of knowledge graph dates way back in 1956, when "Semantic Nets" were first invented for computers by Richard H. Richens as an "interlingua" for machine translation of natural languages (Lehmann & Rodin, 1992). A semantic network is a representation of semantic relations between different concepts or entities. This is often used as a form of knowledge representation. It is a directed or undirected graph consisting of vertices, which represent concepts, and edges, which represent semantic relations between concepts (Sowa, 1987). The basic unit of a knowledge graph is (the representation of) a singular entity, such as a football match you are watching, a city you will visit soon or anything you would like to describe. Each entity might have various attributes  (Gomez-Perez, Pan, Vetere, & Wu, 2017).

Figure 1: Knowledge Graph Example (Source: Google)
The goal of this thesis is to try and create a general Albanian Knowledge Graph which would include entities of various types (people, cities, states, institutions, important dates etc.). These entities are extracted from Albanian Wikipedia corpus, using natural language processing algorithms for creating triplets and storing them in a graph database. This process is called relation extraction. The triplets are generated in the form (subject) - [predicate] - (object).

The implementation is done using Java programming language and Neo4j as a graph database with its own Cypher declarative SQL-like query language for creating nodes and relations between them.

In the first chapter we will do a review for natural language processing and its main tasks and a review for relation extraction in academic area, list some of the latest projects in this area, their benefits and how we will use some of the ideas in our project. We will also have a look at graph as data structures and graph databases as data management systems. In the third chapter we will define the problem of knowledge case construction and the importance of implementing on in Albanian language. In the fourth chapter we briefly discuss the methodology used in solving the problem of knowledge base construction using relation extraction from unstructured text. In the fifth chapter we will describe in detail the process of coding and implementing a knowledge graph, from the unstructured text to the graph itself.
2. LITERATURE REVIEW

In this chapter we will present literature review of the current state of the art algorithms that were used through this thesis, for constructing a knowledge graph.

2.1. Natural Language Processing

Natural Language Processing (NLP) is a field of computer science and artificial intelligence specifically, whose focus is creating different algorithms and software for processing natural language corpora. It represents a range of computational techniques for analyzing and representing raw texts at one or more levels of linguistic analysis, for the purpose of achieving human-like language processing (Liddy, 2001).

NLP is divided in many tasks, some of which we will use for generating entities and relations from raw text. The NLP tasks that are used in this thesis are:

- Syntax
  - Part-of-speech tagging
  - Sentence breaking (also known as sentence boundary disambiguation)
  - Stemming
  - Word segmentation
- Semantics
  - Named entity recognition (NER)
  - Relationship extraction

2.1.1. Part of speech tagging

Part of speech tagging helps natural language processing software to better determine the importance and the context of the word in a document. It can also help find different usages of the same word, in different documents (e.g. the word "close" can be used as a verb in the sentence "Close the door behind you when you leave" or as an adjective in the sentence "He is a close friend.") (Ingersoll, Morton, & Farris, 2013).
For Albanian part of speech tagging this thesis used a POS Tagger implemented by Dr. Ercan Canhasi.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Part of speech</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Noun</td>
<td>Janar, Futboll, Planet</td>
</tr>
<tr>
<td>V</td>
<td>Verb</td>
<td>Është, ishte, shkoi</td>
</tr>
<tr>
<td>ADJ</td>
<td>Adjective</td>
<td>Ngjashme, njohur</td>
</tr>
<tr>
<td>ART</td>
<td>Articulate</td>
<td>Në, të, mbi</td>
</tr>
<tr>
<td>NUM</td>
<td>Numeric</td>
<td>Një, dy, tre</td>
</tr>
<tr>
<td>ADV</td>
<td>Adverb</td>
<td>Më, edhe, si</td>
</tr>
</tbody>
</table>

Table 1: Definition and example of commonly occurring parts of speech

2.1.2. Sentence breaking

Sentence breaking is the process of splitting raw text into sentences. One of the most common methods of sentence breaking are based on Regular Expressions pattern for matching end of sentences characters. Computing sentence boundaries can help reduce erroneous phrase matches as well as provide a means to identify structural relationships between words and phrases and sentences to other sentences. With these relationships, you can then attempt to find meaningful pieces of information in the text (Ingersoll, Morton, & Farris, 2013). In this thesis we will use an open source sentence detection algorithm implemented in Apache Open NLP, using code samples from (Ingersoll, Morton, & Farris, 2013).

2.1.3. Stemming

Stemming is the process of reducing a word to its root, by removing additional prefixes or postfixes (e.g. for the word “fjalive” the root is “fjali” hence we remove the postfix “ve”). In Albanian language, words can take different forms based on context, so stemming is an important part of the processing, for identifying same words on different forms, which results in a better-quality graph.
2.1.4. Word segmentation

Word segmentation is the process of splitting a sentence into specific single words. Given that the Albanian language, same as the English language are Latin based, most of the rules of word segmentation apply to both of the languages, which means that an English tokenizer would work for Albanian sentences too. This thesis uses Apache Open NLP simple tokenizer, using code samples from (Ingersoll, Morton, & Farris, 2013).

2.1.5. Named Entity Recognizer (NER)

Named-entity recognition (NER) is a task of natural language processing which tries to find different entities inside a document or a set of documents (Morwal, Jahan, & Chopra, 2012). The process of entity extraction is based in pre-trained rules and patterns using machine learning statistical models. Based on different implementations, entities can be of various types: persons, cities, states, countries, numbers, dates, money, company names etc. Different software has been implemented for automatic named entity recognition from raw text.

![Figure 2: An example of Named Entity Recognition in raw text](image)

2.1.6. Term Frequency Inverse Document Frequency (TF-IDF)

TF-IDF which is short for Term Frequency – Inverse Document Frequency is a mathematical model for finding the importance of a word inside a specific document of a set (Leskovec, Rajaraman, & Ullman, 2014). Its evaluation is based on the frequency of a word inside a document and inside a set of documents. If a word is frequently mentioned in a document, that increases the probability that this particular word is important to that document. But, if the same word is frequently repeated in every other document that means that word is not important in that context.
(like the verb “is” that is used in almost every document). TF-IDF is a very popular statistical model in information retrieval and data mining, it is estimated that this formula is used in as much as 83% of text-based recommender systems (Breitinger, Gipp, & Langer, 2015).

2.2. Graph

In computer science a graph is a data structure which contains vertices that represent an object with attributes and edges that connect different nodes together (Gibbons, 1985). For example, Facebook’s data is easily represented as a graph. In the Facebook graph people are represented as nodes, and friendships between users are represented as edges or relations. Except users, a node can be a post, page, group, location etc. and a relation can be a like, a follow, check-in etc. In Figure 2 we see an example of a network of users in Facebook.

![Small Facebook social network](image)

**Figure 3** Small Facebook social network

2.2.1. Graph databases

A graph database is a database management system that can operate main database CRUD operations (Create, Read, Update, Delete) and is backed up by a graph model data storing (Ribinson, Webber, & Eifrem, 2015). Unlike relation databases where data is stored in discrete standards, and perform heavy joins in query time, graph databases use connections as primary model of storing information, thus allowing fast join query performance. Graph databases are part of NoSQL storages, which means they are schema free and you can store multiple properties inside a specific node. Each node has a label which can be used to divide nodes in different groups like
People, Company, Page, City etc. You can connect two different nodes together by creating a relation between them. Same as nodes relations can contain multiple properties, which can be used to better describe a relation (e.g. score, weight etc.). The connected model of storing data, gives graph databases the ability to store a large number of nodes and relations without affecting the overall performance. One of the most famous graph databases is Neo4j, which is open source NoSQL graph database implemented in Java and Scala.¹

![Graph database example](image)

**Figure 4: Graph database example**

### 2.3. Relation extraction

Relation extraction is the process of automatically retrieving entities and their relations from unstructured text. The whole relation extraction process is not a trivial task. The computer needs to know how to recognize a piece of text having a semantic property of interest in order to make a correct annotation. Thus, extracting semantic relations between entities in natural language text is a crucial step towards natural language understanding applications (Bach & Badaskar).

¹ More about Neo4j: [https://neo4j.com/developer/graph-database/](https://neo4j.com/developer/graph-database/)
There are two main approaches in relation extraction (Bach & Badaskar)

- Supervised approach
- Semi-supervised approach

Supervised approach is based on two methods:

- Feature based methods: given a set of positive and negative relation examples, syntactic and semantic features can be extracted from the text. These extracted features serve as cues for deciding whether the entities in a sentence are related or not (Bach & Badaskar)
- Kernel Methods - The kernels used for relation-extraction (or relation-detection) are based on string-kernels described in (Lodhi, Saunders, Shawe-Taylor, & Cristianini, 2002). An understanding of the workings of string-kernels is essential for interpreting the kernels used for relation extraction (Bach & Badaskar)

Semi-supervised learning has become an important topic in computational linguistics. For many language-processing tasks including relation extraction, there is an abundance of unlabeled data, but labeled data is lacking and too expensive to create in large quantities, therefore making bootstrapping techniques desirable (Bach & Badaskar).
Some of the most important existing relation extraction systems are:

- **DIPRE** (Dual Iterative Pattern Relation Expansion) is a relation extraction system proposed by (Brin, 1998)
- **Snowball** (Agichtein & Gravano, 2000) which is similar as DIPRE, but the main task is to identify (organization, location) relation on regular text.
- **KnowItAll** (Etzioni, et al., 2005) represents a large-scale Web IE system which uses only a small set of extraction patterns to label its own training examples.
- **TextRunner** (Banko, Cafarella, Soderland, Broadhead, & Etzioni, 2007) which represents a self-supervised relation extraction system
- **ReVerb** (Etzioni, Fader, Christensen, Soderland, & Mausam, 2011) which takes as an input a POS-tagged and NP-chunked sentences and returns a triplet of type (x, r, y)

### 2.4. Knowledge base construction

Completeness, accuracy, and data quality are important parameters that determine the usefulness of knowledge bases and are influenced by the way knowledge bases are constructed. We can classify KB construction methods into four main groups (Nickel, Murphy, Tresp, & Gabrilovich, 2015):

- In curated approaches, triples are created manually by a closed group of experts.
- In collaborative approaches, triples are created manually by an open group of volunteers.
- In automated semi-structured approaches, triples are extracted automatically from semi-structured text (e.g. info boxes in Wikipedia) via hand-crafted rules, learned rules, or regular expressions.
- In automated unstructured approaches, triples are extracted automatically from unstructured text via machine learning and natural language processing techniques.

A curated approach is the best and most accurate way of extracting triplets, but it is limited to scaling because it need human work for manually extracting triplets. The second collaborative
approach is used by various English knowledge bases like Freebase\(^2\), but given the lack of such initiative for Albanian language, it couldn’t be used for creating a knowledge graph in this language. Semi-structured approach too, requires a lot of manual work for extracting hand-crafted rules and given the structure of Albanian language it was a hard approach. So, the approach used for creating an Albanian Knowledge Graph in this thesis is based on automated approaches using machine learning and natural language processing techniques.

Figure 6: Graph example from Freebase data

\(^2\) More about freebase: https://developers.google.com/freebase/
3. PROBLEM DEFINITION

Big data is one of the most important and challenging problem in modern days of technology. The exponential increase of data size has brought the need for more and more data accuracy, completeness, intelligent applications and compatibility.

One of the main problems we face today is that we have a lot of data, but very little useful information and knowledge. Much of these pieces of information are separated in different databases or other storage entities. They are stored in different formats and in discrete standards, being useful for only a specific range of applications. Over time, many different standards have been created, each trying to solve a tight range of problems, thus producing complex domains and complex relationships of same information.

This has produced a high level of complexity and difficulty in trying to gather information from different sources about a particular topic or person. So, given our size of the data that we have access today, and given the state of the art algorithms for data mining and natural language processing, how can we make a central knowledge base that could connect all these different entities and relations between them. All the data has to be stored in one common format, where each entity needs to have a global identifier allowing us to make dynamic advanced queries for retrieving new information on top of existing one.

Top global companies have tried to solve this problem by creating knowledge graphs. One of the most famous one is Google Knowledge Graph, which is the secret behind the side boxes that appear when you make a query about a topic or entity. A lot of algorithms have been developed in automatic and semi-automatic relation extraction as we have seen in the previous chapter. But, there hasn’t been any serious attempt on creating such a knowledge base for Albanian language data. One of the main reasons is the small size of data in that language compared to others one (e.g. English) and lack of initiative on structuring this data in semantic formats.

An Albanian knowledge base would allow us creating a central semantic network of different entities and topics, would allow us make better searches through existing information, would allow us to know when the context of the word “Mars” is about the month and not the planet and vice versa and would allow us to know if an entity is a person, a company, a state, a city, a country, a historic date etc. This is the problem that we are trying to address in this thesis.
4. METHODOLOGY

The methodology that this thesis uses for building an Albanian Knowledge graph is based in using open source free data and tools that can be used by everyone. We got the data from Wikipedia open corpus for Albanian language which can be downloaded from Wikipedia media sites. We have used the open source Java programming language for implementation, Apache OpenNLP for natural language processing and Neo4j as a graph storage, which is open source too. The relation extraction algorithm is based on state of the art techniques which were explained in chapters before and grammatical rules of Albanian language. An important factor of triplet extraction were part of speech tagging which were used for determining the relation types between the subject and the object.
5. IMPLEMENTATION

The whole process of building the knowledge graph is divided in three main parts

- Pre-processing
- Relation extraction
- Storing nodes and relations

5.1. Pre-processing

As we mentioned earlier we have used the Wikipedia Albanian corpus, so the first step is to download the latest corpus dump from Wikimedia dumps³. The data dump is a single file in XML format, containing all of the Albanian Wiki Articles.

For parsing the XML file, we have used an open source Python script named Wikiextractor⁴. WikiExtractor.py is a Python script that extracts and cleans text from a Wikipedia database dump. The extractor parses the XML file and transforms it in different formats. We used the extractor to parse articles in JSON format using the following command line options:

    WikiExtractor.py -o OUTPUT_DIRECTORY --json INPUT_XML_FILE

After processing the entire corpus file, the script splits articles in separate JSON object with the following attributes:

    
    
    
    Code Snippet 1: Parsed Wikipedia Article JSON Format Example

³ Wikipedia Dumps link: https://dumps.wikimedia.org/sqwiki/
⁴ Library GitHub repository: https://github.com/attardi/wikiextractor
When parsing process is done for every article, we continue by splitting the article texts into separate sentences using the Apache OpenNLP\(^5\) sentence detector. The detector uses a pre-trained sentence model, which can be downloaded for free from the official library resources\(^6\). The sentence detection is done using the following method, which returns an array of the detected sentences:

```java
public static String[] splitSentences(String text) throws IOException {
    InputStream sentenceModelIn = new FileInputStream(SENT_MODEL_FILE);
    SentenceModel sentenceModel = new SentenceModel(sentenceModelIn);
    SentenceDetectorME sentenceDetector =
        new SentenceDetectorME(sentenceModel);

    return sentenceDetector.sentDetect(text);
}
```

Code Snippet 2: Apache OpenNLP sentence detecting

For each sentence we have applied a part of speech tagging, to find the most important parts of the sentence. If we take for example the previous sentence\(^7\):

```
“Janari është muaji i parë i vitit nëë Kalendarin Gregorian dhe ka 31 ditë.”,
```

the part of speech tagging result would be like this:

```
“janari|N është|V muaji|N i|ART parë|ADJ i|ART viiti|N në|PREP kalendarin|N
gregorian|ADJ dhe|C ka|V 31|NUM ditë|N”.
```

With some basic regular expression patterns, each response is parsed and split into specific part of speech tokens.

Another iteration is done over each article for named entity recognition, using the open source Stanford NER in Java\(^8\). The name finder takes as input the original text of article and collects entities that are returned by NERClassifierCombiner class. A HashSet is used for eliminating duplicate entities for the same article. We use NERClassifierCombiner because it can run additional classifiers, which allows to recognize numeric and date/time entities\(^9\).

---

\(^5\) Apache OpenNLP official site: https://opennlp.apache.org/

\(^6\) Apache OpenNLP tools models: http://opennlp.sourceforge.net/models-1.5/

\(^7\) The sentence is in Albanian language which in English means: January is the first month of the year in Gregorian calendar and it has 31 days.

\(^8\) Official site for Stanford Named Entity Recognizer: https://nlp.stanford.edu/software/CRF-NER.shtml

\(^9\) Documentation of NERClassifierCombiner: https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ie/NERClassifierCombiner.html
The third iteration over articles is done for getting the stemma for each word in the article. If we take for example the word “shqiptarëve” which means “albanians”, the stemming process would split the word in two parts: “shqiptar|ROOT eve|POSTFIX”. In this thesis we use only the roots of each word and ignore the postfix part.

In this part, we split the article content into sentences, assigned the part of speech tag for each word of each sentence, replaced each word with the stemma root of that word, and found named entities in each original text of the article. Now, each article is represented as an array of objects,

```java
public static List<String> findNames(String content) throws IOException {
    String serializedClassifier = "DIRECTORY_PATH/english.all.3class.distsim.crf.ser.gz";
    String serializedClassifier2 = "DIRECTORY_PATH/english.muc.7class.distsim.crf.ser.gz";
    NERClassifierCombiner classifier = new NERClassifierCombiner(false, false, false, serializedClassifier, serializedClassifier2);
    Set<String> uniqueNames = new HashSet<String>();
    List<List<CoreLabel>> out = classifier.classify(content);
    for (List<CoreLabel> lcl : out) {
        StringBuilder builder = new StringBuilder();
        int lastPosition = 0;
        String lastText = "";
        for (CoreLabel cl : lcl) {
            if (!cl.ner().toLowerCase().equals("o")) {
                if (!lastText.isEmpty() && cl.beginPosition() > lastPosition + lastText.length() + 1) {
                    String result = builder.toString().trim();
                    if (!result.isEmpty()) {
                        uniqueNames.add(result);
                        builder = new StringBuilder();
                    }
                    lastText = cl.originalText();
                    lastPosition = cl.beginPosition();
                    builder.append(cl.originalText().append(" ");
                }
            }
        }
        String result = builder.toString().trim();
        if (!result.isEmpty()) {
            uniqueNames.add(result);
        }
    }
    return uniqueNames.stream().collect(Collectors.toList());
}
```

Code Snippet 3: Method for finding named entities inside the content of an article using Stanford NER
each of whom contains the original text of the sentence and the POS tagged result of the sentence. The article object also contains an array of entities found by the Names Entity Recognizer. A simple article in this part of the processing looks like the following object represented in JSON format:

```
{
  "preProcessedSentences": [
    {
      "articleId": 156235,
      "originalText": "Mikel Tarabulluzi qe klerik dhe mësues i gjuhës shqipe në shkollën e Stubllës.",
      "posResult": "mikel|N tarabulluzi|ADV qe|V klerik|N dhe|C mësues|N i|ART gjuhës|N shqipe|ADJ në|PREP shkollën|N e|ART stubllës.|N"
    },
    {
      "articleId": 156235,
      "originalText": "Leu më 4 prill 1868 në Prizren, i biri i Zefit dhe Stanës."
    }
  ],
  "entities": [
    "Mikel Tarabulluzi",
    "Stubllës",
    "4 prill 1868",
    "Prizren",
    "Zefit",
    "Stanës"
  ]
}
```

Code Snippet 4: Pre-processed article model in JSON

The fourth and final iteration is done for calculating the TF-IDF score for each word of each article. We iterate through the POS result of each sentence of the article, split it in white spaces, and take only the word value without the POS Tag. For each unique word we create a unique TF-IDF token which contains these properties:
public class TfIdfToken {
    // id of the wiki article
    private int articleId;

    // value of the word
    private String word;

    // number of occurrences of the specific word in the specific article
    private int count;

    // maximum count of any word in the specific article
    private int max;

    // term frequency in all articles
    private int tf;

    // inverse document frequency
    private double idf;

    // final score value
    private double tfidf;

    public TfIdfToken setArticleId(int articleId) {
        this.articleId = articleId;
        return this;
    }

    public TfIdfToken setWord(String word) {
        this.word = word;
        return this;
    }
}

Code Snippet 5: TFIDF Token class

Given the large number of documents and words, we had to make some optimization for calculating the TF-IDF of each word, which includes pre-calculating the word frequencies and token Max values. This means that we made two iterations through the articles before calculating the actual score of TF-IDF.

In the first iteration we calculated the word frequencies and created TFIDF-tokens for each word.
In the second iteration we calculated the maximum word frequency for each article.
In the third and last iteration we calculated the score of TF-IDF for each token of articles.

In the third and last iteration we calculated the score of TF-IDF for each token of articles.

In the third and last iteration we calculated the score of TF-IDF for each token of articles.

In the third and last iteration we calculated the score of TF-IDF for each token of articles.

5.2. Relation extraction

The most important part of the knowledge graph construction is the relation extraction process. Here an iteration of each preprocessed articles is done, focusing in three main parts of relation extraction: subject extraction, predicate extraction and then object extraction.
Before we continue with the extraction process, we first must define models for representing a graph node and a graph relation. As has been mentioned in section 1, a node can contain different parameters. In this thesis we use this parameters for a graph node representation:

```java
public class GraphNode {
    private String id;
    private String name;
    private String original;
    private int articleId;
    private String nodeType;
}
```

Code Snippet 9: Graph node class representation

We also use a class for graph relation representation. In each relation of the graph we store the type of the relation, the original text of the sentence from which the relation was extracted and the article id. These properties will be used later for querying information from the knowledge graph.

```java
public class GraphRelation {
    private String relationType;
    private String posTag;
    private String originalText;
    private int articleId;
}
```

Code Snippet 10: Graph relation class representation

We also have a triplet wrapper class, which has as a property the references of the first, second and relation instances of that triplet extracted from a specific sentence. We use this triplet for later storing them in the graph database.

```java
public class TripletWrapper {
    private GraphNode fromNode;
    private GraphNode toNode;
    private GraphRelation relation;
}
```

Code Snippet 11: Triplet wrapper class representation
5.2.1. Subject extraction

Iterate through each word and check if the given word is the subject of the article, which usually is part of the title or a frequent personal pronoun. If the subject is found, a check is made whether the previous word (if there is any) is a number. The check is performed by looking if the POS tag of the previous word is “NUM”. If the result is true then the number is prepended with the subject value. After we extract the subject, we create a graph node instance, giving as an id the lower-case value of the subject itself and a node type of “Article”.

```java
for (int i = 0; i < sentenceTokens.length; i++) {
    PosToken pT = sentenceTokens[i];
    if (isSubject(pT, article.getTitle())) {
        String subject = Stemming.getRoot(pT.getWord());

        //check whether the previous word is num
        if (i > 0) {
            PosToken prev = sentenceTokens[i - 1];
            if (prev.getTag().equals("NUM")) {
                subject = prev.getWord() + " " + subject;
            }
        }

        //create the graph node
        GraphNode subjectNode = new GraphNode()
            .setId(subject.toLowerCase().trim())
            .setName(subject)
            .setOriginal(pT.getWord())
            .setNodeType("Article");
    }
}
```

Code Snippet 12: Subject extraction method

5.2.2. Predicate extraction

Given the position of the subject, find the nearest verb after, which correlates with the subject and mark it as a predicate. If the relation is part of title (hence part of the subject), skip to the other word, until another verb is found. The maximum distance\(^{10}\) allowed between the subject and the predicate is 5. A predicate is valid only if there is another word after it.

\(^{10}\) Distance is measured by the count of words that are between the subject and the next verb
If the next word after predicate, is a preposition, a grammar article or a number, it is concatenated with the original predicate. This process is repeated if there is another same occurrence. For example, let’s analyze the following sentence “Naim Frashëri është një shkrimtar” which is in the content of an Wikipedia article about Naim Frashëri, a famous Albanian writer. The POS tags for this sentence are: “[Naim|N Frashëri|N është|V një|NUM shkrimtar|N|]”. The subject extraction algorithm will extract the first two nouns as the subject of the sentence. The next word, “është” is a verb hence will be marked as a predicate. But, also the next word after the verb is a number which completes the meaning of the verb, so the algorithm will concatenate the NUM “një” with the verb “është”, thus giving us the complete predicate of the sentence “është një”, which in English means “is a”. After the process is completed, we create a graph relation instance where the relation type is the value of the predicate, and we also store the original sentence of the predicate and the article id. The snippet code below demonstrates the predicate extraction from the sentence, and is inside the previous for loop from the Code Snippet 12.

```java
for (int j = i + 1; j < sentenceTokens.length; j++) {
    PosToken pT2 = sentenceTokens[j];
    if (isSubject(pT2, article.getTitle()))
        continue;
    if (pT2.getTag().equals("V")) {
        int dist = i - j;
        if (dist > 5)
            break;
        if (j > sentenceTokens.length - 1)
            break;
        String predicate = Stemming.getRoot(pT2.getWord());
        String tag = pT2.getTag();
        PosToken oT = sentenceTokens[++j];
        while (isHelper(oT) && j < sentenceTokens.length - 1) {
            predicate += " " + Stemming.getRoot(oT.getWord());
            tag += " " + oT.getTag();
            oT = sentenceTokens[++j];
        }
        GraphRelation relationNode = new GraphRelation()
            .setRelationType(predicate.toUpperCase())
            .setPosTag(tag)
            .setOriginalText(sentence.getOriginal())
            .setArticleId(article.getId());
    }
}
```

Code Snippet 13: Predicate extraction algorithm

---

11 This is another Albanian sentence which in English means: Naim Frasher is a writer.
5.2.3. Object extraction

After the predicate extraction is completed, we iterate through each word left and find the most important one, based on TF-IDF score, marking it as the object. However, this can cause many false objects if the word with the highest score is in the end of the sentence and does not have direct correlation with the predicate. To avoid this problem, we put into consideration the distance between the predicate and the object, giving more priority to words which are closer to the predicate. The implementation of this priority is done by lowering the word score with the increase of distance from predicate. We divide the TF-IDF score of each word that has a distance of three or more with the natural logarithm of the distance. In this way words that are more distant from the predicate have a slightly lower chance to become object of the triplet, but still enough chance to become so, given known logarithmic scale. In the following snippet we show the method used for normalizing TF-IDF scores of words, based in their distance from the predicate.

```java
public double getNormalizedScore(
    int predicateIndex,
    int wordIndex,
    double tfIdfScore)
{
    int difference = wordIndex - predicateIndex;
    if (difference > 2) {
        double normalizedScore =
            tfIdfScore / Math.log(difference);
        return normalizedScore;
    }
    return tfIdfScore;
}
```

Code Snippet 14: TF-IDF score normalization based on distance

Same as with predicate, if the word after extracted object is a preposition, a grammar article or a number we process the following:

- Let’s name this word with R
- We get the next word after R, and name it with P
- We create a relation between the subject and the P using R as a relation
- We create a new object by concatenating the previous object with R and P, thus creating a multi-word object and naming it RP
- We create a new relation between the subject and the new multi-word object RP
When we extract the object, before creating the graph node instance we perform a check whether the object is a named entity. If the result is true, we give to the node a label of “Entity”.

```java
int index = 0;
double highest = 0;

//find the most important node
for (int k = j + 1; k < sentenceTokens.length; k++) {
    PosToken tK = sentenceTokens[k];
    double tfIdfScore =
        tokensMap.get(article.getId())
            .get(tK.getWord()).getTfIdf();
    double normalizedScore = getNormalizedScore(j, k, tfIdfScore);
    if (index == 0 || normalizedScore > highest) {
        index = k;
        highest = normalizedScore;
    }
}
if (index > 0) {
    PosToken tK = sentenceTokens[index];
    String obj = Stemming.getRoot(tK.getWord());
    String objTag = tK.getTag();
    if (isEntity(obj, article.getEntities())){
        objTag = "Entity";
    }

    //create the graph node
    GraphNode objectNode = new GraphNode()
        .setId(obj.toLowerCase().trim())
        .setName(obj)
        .setOriginal(tK.getWord())
        .setNodeType(objTag);
}
```

Code Snippet 15: Object extraction method

In the above method we implement the object extraction algorithm which is based in finding the most important word after the predicate, and mark it as an object of the triplet. If the iteration can’t find any word with positive score then that sentence is passed and no relations are formed. If a word is found then we put all the nodes and relations in specific lists and check whether there are additional words to append in the object node based in previous steps.
List<GraphNode> nodes = new ArrayList<GraphNode>();
List<TripletWrapper> relations = new ArrayList<TripletWrapper>();
nodes.add(subjectNode);
nodes.add(objectNode);

TripletWrapper relationWrapper = new TripletWrapper()
    .setFromNode(subjectNode)
    .setToNode(objectNode)
    .setRelation(relationNode);
relations.add(relationWrapper);

int k = index++;
if (k < sentenceTokens.length) {
    while (isHelper(sentenceTokens[k]) && k < sentenceTokens.length - 2) {
        PosToken aT1 = sentenceTokens[++k];
        PosToken aT2 = sentenceTokens[++k];

        String additionalObject = Stemming.getRoot(aT2.getWord());
        String additionalRelation = Stemming.getRoot(aT1.getWord());

        GraphNode additionalNode = new GraphNode()
            .setId(additionalObject.toLowerCase().trim())
            .setName(additionalObject)
            .setOriginal(aT2.getWord())
            .setNodeType(aT2.getTag());

        String multiWord = obj
            + " " + additionalRelation
            + " " + additionalObject;
        GraphNode multiObjectNode = new GraphNode()
            .setId(multiWord.toLowerCase().trim())
            .setName(multiWord)
            .setOriginal(multiWord)
            .setNodeType(objTag + "_" + aT1.getTag() + "_" + aT2.getTag());

        GraphRelation additionalRelationNode = new GraphRelation()
            .setRelationType(additionalRelation.toUpperCase())
            .setPosTag(aT1.getTag())
            .setOriginalText(sentence.getOriginal())
            .setArticleId(article.getId());
        TripletWrapper additionalWrapper = new TripletWrapper()
            .setFromNode(objectNode)
            .setToNode(additionalNode)
            .setRelation(additionalRelationNode);
        relations.add(additionalWrapper);

        TripletWrapper multiWordWrapper = new TripletWrapper()
            .setFromNode(subjectNode)
            .setToNode(multiObjectNode)
            .setRelation(relationNode);
        relations.add(multiWordWrapper);
    }
}
5.3. Storing nodes and relations

The last part of knowledge graph construction is storing nodes and relations in a graph database. From the previous snippet code, we have a list of nodes and relations that are going to be store in the graph database. As we mentioned earlier in the beginning of the thesis, we are using Neo4j Community edition as a graph database management system. Neo4j uses Cypher as a declarative language for querying information.

Before storing nodes and relations, we perform a quick iteration in order to remove any duplicate triplet, even though Neo4j can store two identical nodes or can store only unique nodes.

The cypher syntax for creating a node, in this case an article node is:

```
CREATE (node:Article { Id: id, Name: 'Title of article' })
```

if we want to create unique nodes, thus removing duplicates we use the following command:

```
MERGE (node:Article { Id: id })
ON CREATE article
SET article.Name = 'Title of article'
```

After creating all the nodes extracted from the previous step, we then connect these nodes using relations create command:

```
MATCH (node1:Article), (node2:Article)
WHERE node1.Id = id1, node2.Id = id2
CREATE UNIQUE node1[:VERB]->node2
```

As a channel of communication between Neo4j database and Java we will use the open source library named jCypher\(^\text{12}\).

---

\(^\text{12}\) jCypher GitHub repository: https://github.com/Wolfgang-Schuetzelhofer/jcypher
For storing nodes in graph database, we use the following method:

```java
public void insertTriplets(List<TripletWrapper> triplets) {
    // create a new graph model
    Graph graph = Graph.create(dbAccess);
    for (TripletWrapper triplet : triplets) {
        // create and populate nodes
        GrNode fromNode = graph.createNode();
        GrNode toNode = graph.createNode();
        populateGrNode(triplet.getFromNode(), fromNode);
        populateGrNode(triplet.getToNode(), toNode);

        // create and populate a relation
        GrRelation rel =
            graph.createRelation(
                triplet.getRelation().getRelationType(),
                fromNode, toNode);
        populateGrRelation(triplet.getRelation(), rel);
    }
    graph.store();
}
```

**Code Snippet 18: Node inserting into Neo4j using jCypher for Java**

```java
private void populateGrNode(GraphNode graphNode, GrNode grNode) {
    grNode.addLabel(graphNode.getNodeType());
    // add properties
    grNode.addProperty("id", graphNode.getId());
    grNode.addProperty("name", graphNode.getName());
    grNode.addProperty("original", graphNode.getOriginal());
    grNode.addProperty("articleId", graphNode.getArticleId());
}
```

**Code Snippet 17: Inserting relation triplets in graph database from Java**

```java
private void populateGrRelation(GraphRelation relation, GrRelation grRelation) {
    grRelation.addProperty("tag", relation.getPosTag());
    grRelation.addProperty("originalText", relation.getOriginalText());
    grRelation.addProperty("articleId", relation.getArticleId());
}
5.4. Retrieving information from knowledge graph

After inserting all nodes and relations in graph database, in this part we demonstrate some information retrieval from the new constructed knowledge graph. We will use some cypher queries for returning specific nodes and relations and will use Neo4j browsing user interface for visualization of the graph results.

First let’s match all nodes and relations that exist in the database. Because of the limits of Neo4j user interface only random 300 nodes and relations are returned. We execute the following cypher query: “MATCH (N) RETURN N;” and get the following result:

![Figure 7: Small part of Albanian Knowledge Graph (only 300 nodes)](image)

Each circle is a node and each line represents a relation between two nodes.

If we want to get all nodes and relations of a specific Wikipedia article we use the following cypher query:

```
MATCH (N:Article { ArticleId:4869 })-[R]-(M) return N, R, M
```

The id 4869 is for the article “Lidhja e Prizrenit”, which has these nodes:
As we can see above the yellow circle is the subject of the article, and all other circles are objects extracted from that article. Nodes with green color contain nouns, with blue color contain entities and those with purple color contain adjectives. Other circles are combination of different words.

If we analyze the article for our current president Hashim Thaqi, which has the id of 17392, among other relations we see that there is a relation between him and the word president with an ėshtë (which in English means “is a”) relation.
If we query all nodes that also have a relation with the node president, then we get all the Albanian articles that are about current or previous presidents:

![Diagram of nodes related to the node president](image1.png)

**Figure 10:** Nodes that have a relation with the node Hashim Thaçi

Another similar example we could take by analyzing the article about Prishtina. If we make a similar query we can see a relation between “Prishtina” and the word “kryeqytet”, and also other nodes that are or have been capitals and have an Albanian Wikipedia article about them.

![Diagram of nodes related to Prishtina and kryeqytet](image2.png)

**Figure 11:** Another simple relation illustration
Figure 12: Some of the capitals nodes related with Prishtina

Another example can be used by date of birth, for example if we want all users that are connected with the date 20th of May by using the following query:

"MATCH (n:Article)-[r]-(m { Id: 'mē_20_maj'}) RETURN n,r,m;", 

We can all connected articles that are related with this date node. Same method can be applied to get for example all users born in a specific year, month or day. If we perform the query above, then we get the result that is displayed in the next page:
These are just some examples of potential information retrieval queries from knowledge graph. A complete module can be implemented, that could parse user natural queries and transform them in cypher queries for answering different questions, but that is outside the scope of this thesis. Some of the applications that can use knowledge graph are discussed in the next chapter.
6. APPLICATIONS

Given the nature of graphs, a knowledge graph base could be helpful for a range of applications and services. In this chapter we provide some of the most important applications that can be built on top of a knowledge graph.

Searching

Since 2012, when you make a search in Google, a little information/summarization box started to appear in the right side of search results. This information box is powered by the Knowledge Graph that Google launched in that year. The need for such a feature comes because users more often search for a quick answer without wanting to read long articles. An additional cause of this behavior from users came from the rise of mobile usage. People browsing on mobile wanted immediate answers for their questions without having to surf through different web-pages.

Figure 14: Google search results using knowledge graph

The Albanian knowledge graph would improve an Albanian search engine, by allowing to make sophisticated semantic queries and retrieving detailed information about a specific query.
Question Answering

The main reason that people use internet is to search about information, or put in a different perspective to find answers. Automatic question answering systems are among the most advanced innovations in modern artificial intelligence. Some of the most famous question answering systems are powered by huge knowledge graphs, which are used to retrieve information based on user questions. An Albanian knowledge graph would be the first step towards building an Albanian question answering system, which can provide information upon existing data about different entities. Based on nodes and relations, the system could answer user questions like: “Who is Hashim Thaqi?”, “What is the capital of Kosovo?”, “Who is the prime minister of Kosovo?” etc.

Text summarization

The task of text summarization is to produce a short, concise and comprehensive summary to provide the main information about a collection of documents. This collection of documents could be news articles, blog posts, research articles, user reviews, comments etc. By using a knowledge graph these documents could be linked together, thus allowing us to find the main entities or topics of the collection, and by that try to extract the main concepts, analyze the relation between the documents and produce a qualitative summary for them.

Figure 15: Text summarization illustration
Recommender Systems

By using users browsing history and having a knowledge graph, we could build a reading recommender system based on user interests. For example, if a user read an article about Lorik Cana, and by knowledge graph we know that Lorik Cana is an Albanian football player then we can assume that the reader is interested in sport, or Albanian football players particularly in this case. With this information we can recommend other articles about football or Albanian football players that might interest the user. This can be used in targeted advertising based on user interests and entity relations info’s about the topics and user intents.

Other applications

Beside applications mentioned above, some other applications that could use knowledge graphs are:

- Automated fraud detection
- Intelligent chatbots
- Knowledge management systems
- Fake news detection
7. CONCLUSIONS

Knowledge graph are one of the hottest trends in artificial intelligence. An Albanian knowledge graph is the first step in building Albanian systems for semantic searches, automatic question answering and other digital assistants.

We provided a review of natural language processing with its tasks (part-of-speech tagging, sentence breaking, word segmentation, stemming, named entity recognition, word importance) and their open source implementation that were used in constructing an Albanian knowledge graph. We also provided a review for graph data structures, graph databases and their existing management systems.

In third chapter we defined the problem of knowledge extraction, focused on the needs for an Albanian knowledge base. We emphasized the lack of effort for building such a solution and the importance of having a knowledge graph for Albanian language data.

In fourth chapter we explained in detail the step-by-step process of building an Albanian knowledge graph using snippet code for most important parts of the implementation. All the tools that have been used were explained and links for more information were provided. The snippet codes were in Java programming language.

As a result of this thesis, we implemented an automatic relation extraction algorithm and created a knowledge graph using extracted triplets. The final knowledge graph contains the total of 126,766 nodes, linked together by 249,928 different relations.

A table of the most common node labels is given below:

<table>
<thead>
<tr>
<th>Node Label</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article</td>
<td>40,041</td>
</tr>
<tr>
<td>Entity</td>
<td>13,123</td>
</tr>
<tr>
<td>Noun</td>
<td>14,935</td>
</tr>
<tr>
<td>Adjective</td>
<td>2,308</td>
</tr>
<tr>
<td>Adverb</td>
<td>715</td>
</tr>
</tbody>
</table>

Table 2: Most common node labels in knowledge graph
Another table of most common relation types is give below:

<table>
<thead>
<tr>
<th>Relation Type</th>
<th>Number of relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eshtë</td>
<td>61,540</td>
</tr>
<tr>
<td>Ka</td>
<td>18,338</td>
</tr>
<tr>
<td>Ishte</td>
<td>11,530</td>
</tr>
<tr>
<td>Të</td>
<td>7,532</td>
</tr>
<tr>
<td>Janë</td>
<td>3,634</td>
</tr>
<tr>
<td>Me</td>
<td>2,969</td>
</tr>
</tbody>
</table>

Table 3: Most common relations types in knowledge graph

In the last part of the implementation chapter we provided some examples of information retrieving using Cypher queries and different relation combinations.

However, there are many improvements that could be done in the implementation process. The subject extraction can be used in other nouns of the sentences, not only to the subject that are directly correlated with the article subject as were used in this thesis. This could increase the number of extracted relations. Another object scoring mechanism, that could take into consideration the context of the sentence and other linguistic properties can be implemented that could potentially increase the quality of triplets.

This thesis uses only Wikipedia articles, but another approach can be done by using news articles or different blog posts, that would keep the graph updated by day to day events and developments.

In the last chapter we discussed different applications for an Albanian knowledge graph, focusing in search, question answering and recommender systems.
8. REFERENCES


