Knowledge Graphs: Construction and Applications

by

Liang Tan

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Department of Electrical and Computer Engineering
University of Alberta

Abstract

Knowledge graphs become a de facto standard for representing data in situations where relations between individual pieces of information are important. These semantically rich data structures provide opportunities to develop methods and algorithms for new ways of analysing and utilizing data.

In this work, we illustrate benefits of representing data as knowledge graphs. We develop a number of algorithms that augment data via processing graph-based information. We show how these enriched structures can be utilized for further processing, and discovering of 'unseen' relations. In particular, we focus on two very different In particular, we focus on two very different domains of interest: power systems, and aspect-based sentiment analysis.

For a power system application, we construct a knowledge graph that combines topological data with information about technical details of electrical components, as well as past events that occurred in the system. We utilize the graph for extracting details needed for the analysis of different events. We develop methods that enable us to analyze an impact of events on different parts of the system. For a case of aspect-based sentiment analysis, we are interested in emotional-based representation of reviews. Firstly, we identify words and simple phrases that describe different aspects of the reviewed items. Secondly, we create a knowledge graph that combines the reviews with the description words and phrases, as well as with the Hourglass Model of Emotions. The graph allows us to identify emotions linked with the reviewed aspects. Further, we aggregate these emotions across all words and phrases that describe individual aspects to obtain an 'emotional summary' of the reviews.

Preface

This thesis was supervised by Professor Marek Reformat. Chapter 3 of this dissertation have been submitted to the " 2020 IEEE Electric Power and Energy Conference (EPEC) " by Yashar Kor, Liang Tan, Professor Marek Z Reformat, Professor Petr Musilek. The name of the conference paper is "GridKG: Knowledge Graph Representation of Distribution Grid Data" .

Acknowledgements

I would first like to thank my supervisor Dr Marek Reformat. His help and encouragements have supported me a lot whenever I ran into troubles in my Master research. Dr Marek Reformat also steered me into the right directions whenever I am confused. I am grateful for his supervision.

Secondly, I would like to thank my colleagues, Yashar Kor and Manpreet Kumar. They are professional and helpful and their advice is really helpful and enlightening. It is a pleasure for me to work with them together for our research projects.

I would also like to express my gratitude to my parents and friends for their unfailing support and encouragement throughout my years of the study. This accomplishment would not possible without them.

Finally, I would like to say thanks to the staffs of the Department of Electrical and Computer Engineering, who are supportive and friendly. With their constantly supports, I had an enjoyable and memorable study period.

Liang Tan

University of Alberta

January 2021

Table of Contents

Appendix B:

List of Tables

List of Figures

Chapter 1

Introduction

The easiness of generating data has created needs for developing new data storing formats that enable users to fully utilize collected information. It seems that current ways of storing information in a form of relational databases does not provide sufficient ways of extracting relevant information. Also, a process of finding new 'views' on facts and relations in the collected and stored information is a challenging task.

Currently, an access to and understanding of information from the point of view of connections and relations existing between individual pieces of data is rather limited. Yet, during data processing activities it is quite often that we are interested in finding and taking advantage of relationships – direct and indirect– that exist between collected data. It can be stated that the aspect of relations between individual pieces of data is of increasing importance.

1.1 Motivation

The growing amount of data available for processing leads to a number of new expectations regarding data processing. This, on the other hand, creates some challenges and requirements associated with data representation format. Let us take a look at a few of them:

- large quantities of 'hybrid' data, i.e., numerical, symbolic, domain specific, generic – this triggers a need for integration of variety of different type of data,

and methods that are able to take advantaged of such a diversity;

- importance of connections between pieces of information – it seems that data semantics (defined via interconnections between data pieces) promises discovery of new relations and dependencies not easy to spot especially in the case of large data.

To summarize, we can say that there is a strong need/demand for representing and storing data in more structural way where processing, utilization of existing and discovering new relations among data pieces is more natural and straightforward.

1.2 Objectives and Contributions

In the light of the above mentioned needs, we focus on graph-based data representation. An increased popularity of such a data format has resulted in development of a few standards for data graph formats. One that is of our interest is called Resource Description Framework (RDF)[1]. This format – standardized by W3C – provides a very simple yet efficient concept of representing data. The basic building block is called a $< triple >$. It is a 3-tuple $< subject, property, object >$ that expresses a simple fact about *subject*. It is accomplished via linking it with another piece of information via a *property*; the link is called $< property >$. For example, a statement *Ottawa is capital of Canada* is represented as the triple $< Ottawa, capital, Canada >$. A collection of such triples interconnected between each other is called *Knowledge Graph (KG)*.

In this thesis, we develop procedures that help constructing knowledge graphs, integrating multiple data sources, enabling data processing to 'generate' new data, and analysing graphs to gain new insight into phenomena described by the data.

In order to perform such activities , we have identified a number of sub-tasks:

- construction of vocabularies for graph databases that represent elements and relations characteristic to a domain of interest;

[1] https://www.w3.org/TR/rdf-primer/

- create procedures for 'translating' tabular data into graphs;
- design and develop multiple approaches to utilize relation-based view of data, and to generate new content based on the graph structure/relations.

In order to illustrate these activities, we look at two different domains of interest, i.e., power systems and aspect-based sentiment analysis of reviews. We show how data representation in a form of graph creates new possibilities in data processing and analysis.

For a power system application, we construct a knowledge graph that combines topological data with information about technical details of electrical components, as well as past events that occurred in the system. We utilize the graph for extracting details needed for the analysis of different events. We develop methods that enable us to analyze an impact of events on different parts of the system.

For a case of aspect-based sentiment analysis, we are interested in emotional-based representation of reviews. Firstly, we identify words and simple phrases that describe different aspects of the re-viewed items. Secondly, we create a knowledge graph that combines the reviews with the description words and phrases, as well as with the Hourglass Model of Emotions.The graph allows us to identify emotions linked with the reviewed aspects. Further,we aggregate these emotions across all words and phrases that describe individual aspects to obtain an 'emotional summary' of the reviews.

1.3 Thesis Outline

The thesis is divided into five sections. Section 2 introduces the background information of knowledge graph, emotion models, sentiment analysis and group decision making tasks. Section 3 proposes a knowledge graph representation of power grid ontology, named GridKG. We illustrate the methods of GridKG construction and the utilization of the Knowledge graph. Section 4 describes the emotion-based analysis

of reviews with NLP methods and create a Model called **REmModel**, and store the extracted information in a knowledge graph for aggregation computation. Section 5 provides the conclusions for the thesis.

Chapter 2

Background and Related Work

2.1 Knowledge Graph

A knowledge graph(KG) is basically a data management system which combines various types of data and utilizes graphs to represent information and knowledge [1]. Their intrinsic ability to denote relations between individual pieces of data/information makes them one of the most suitable forms for expressing data semantics. A W3C standard that provides a definition of graph-based data format is Resource Description Framework – RDF. Initially introduced by Google, the knowledge graph concept was utilized to optimize search engine performance with the information gathered from various sources. Knowledge graphs such as BabelNet [2], DBpedia [3], WordNet [4], Microsoft's Probase [5] and Google Vault [6] are created focusing on text-based extraction of data from the web content. The existing generic knowledge graphs prove their usefulness in applications such as semantic search, and information fusion from various sources.Knowledge graph has intrinsic ability to denote relations between individual pieces of data/information makes them one of the most suitable forms for expressing data semantics. A W3C standard that provides a definition of graph-based data format is Resource Description Framework – RDF.

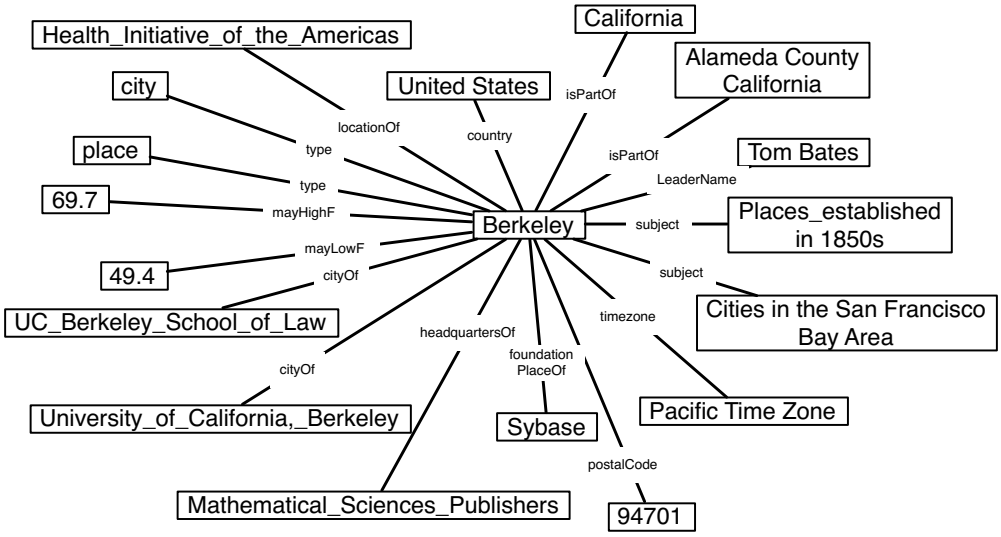


Figure 2.1: RDF: triples describing Berkeley

2.1.1 Resource Description Framework (RDF)

A single RDF triple is of the following form: $< subject - property - object >$. A component $< subject- >$ represents an element that a given triple is describing. Further, an $< -object >$ is an element that is in relation $< -property- >$ with the $< subject- >$ and contributes to its description. $< -Property >$ is used to express a relationship that exists between $< subject- >$ and $< -object >$. Subject, objects and properties can be of any type, but only objects can be alphanumeric literals.

Generally, multiple triples can have the same subject. Such a situation means that all those triples constitutes a description of a single, the same, entity. One of such triples, on the other hand, can be perceived as one feature of the entity/subject. Further, those triples are a definition of this entity. An illustration of this is shown in Figure 2.1. It is a definition of the city Berkeley.

When we look at a large number of RDF triples many of subjects and objects of some triples are subject and objects of other triples. That means that multiple triples are very interconnected between each other. So, RDF-based definitions of entities are quite interleaved: features are shared and many of them are subjects and centers of

other RDF-based definitions.

An increasing importance of graph-based representation of data and information, and an RDF in particularly, has led to multiple storing RDF triples. If we think of a process of collecting RDFs as a learning process, we should be aware that the same entity can be defined/described in multiple places and this can lead to some benefits as well as complications.

In general, accumulating descriptions of the same entity is equivalent to a repetitive process of acquiring information and eventually gaining confidence in gathered descriptions, i.e., RDF triples. At the same time, some discrepancies and inconsistencies can occur. A methodology of dealing with such a learning process is being proposed here. It enables a gradual learning, and leads to strong belief in individual RDF triples. As a result, the information about an entity is composed of triples associated with different levels of confidence. Those levels depend on how an assimilation of new information is being performed.

2.1.2 Graph Database: Neo4j

Neo4j [7] is an open-source NoSQL database written in Java and Scala, implements the generic graph models with full database characteristics such as ACID transaction compliance, cluster support, runtime failover, and query language called Cypher Query Language (CQL). It represents non-structural data by nodes with distinct labels and relationships between nodes are called paths. Path-oriented operations are highly efficient and suitable for production applications.

Figure 2.2, shows the main modules of the Neo4j's architecture, including interface APIs (Cypher, Traverse API, Core API), cache, transaction logs, records files and disks. In Neo4j Enterprise version, it also includes High Availability module with the master-slave pattern. If there is a Neo4j database cluster, there will be several Neo4j databases inside. The master instance uses its cluster management system to keep track of any replica instances joining or leaving. When the master instance broke

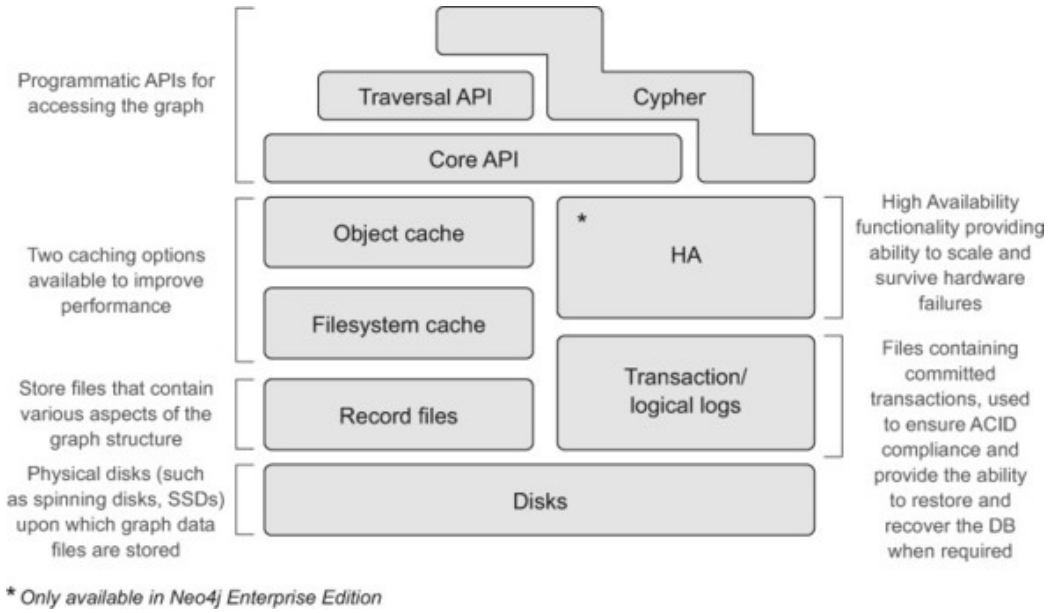


Figure 2.2: Neo4j Architecture[1]

down, the cluster will perform a leader election algorithm to ensure that the new master is elected consistently.

Different types of APIs provide programmers with the flexibility to access the database. Callback interface provided by Traversal API allows programmers to implement his/her approach to traverse the graph. Cypher Query Language (CQL), is an easy-to-use SQL-like language to query the database, providing similar functionalities to other standard data access languages. Neo4j also provides core APIs such that one can use JVM-based languages like Java and Scala to call Kernel APIs to interact with the graph. The performance of reading from the database and writing into the database highly relies on the caching options. Neo4j provides two different types of caching layers. One is the File System cache which uses off-heap memory to cache data stored on disk. The other one is the object cache. All writes and reads are performed through these caches to improve throughput. All data in these caches will not flush to durable storage until the logical logs are rotated. Transaction logs, also known as logical logs can ensure ACID properties and provide the ability

[1]https://apprize.best/javascript/neo4j/13.html

to restore and recover database when needed. Record files are the files that contain the information on nodes, relationships and properties.

2.2 SenticNet and the Hourglass Model of Emotions

SenticNet5 [8] leverages the generalization power of recurrent neural networks, specifically Bi-direction Long Short Term Memory networks (Bi-LSTM) to automatically discover concept primitives for sentiment analysis. After that, the concept primitives are used to build a large commonsense knowledge graph by multi-dimensional scaling. SenticNet tries to merge symbolic and sub-symbolic AI in the context of sentiment analysis. It contains 100,000 concepts as a commonsense knowledge which are represented in an XML repository on the SenticNet website.

Emotions, rather than the classical sentiment classification tasks, are difficult to define and modelling. Emotion categorization modelling is an intensive research area so that there is a large amount of literature on emotion categorization model.

There is no such universal agreement on emotion modelling. Cambria [9] proposed a model, called Hourglass Model of Emotions, Figure 2.3, for emotion categorization recognition. The Hourglass of Emotions model utilize the empirical evidence in the context of sentiment analysis, so it's optimized for polarity detection and has powerful effects for sentiment analysis [10].

The Hourglass Model of Emotions measures every emotion in four dimensions in the context of Human-Computer Interaction (HCI), which are Pleasantness (how much a person is amused by interaction modalities), Attention (how much a person is interested in interaction contents), Sensitivity (how much a person is comfortable with interaction dynamics) and Aptitude (how much a person is confident in interaction benefits).

For each dimension, the strength of the emotion is classified into six-levels, terms $'sentic\ level'$. Therefore, there are a set of 24 emotions in the Hourglass Model

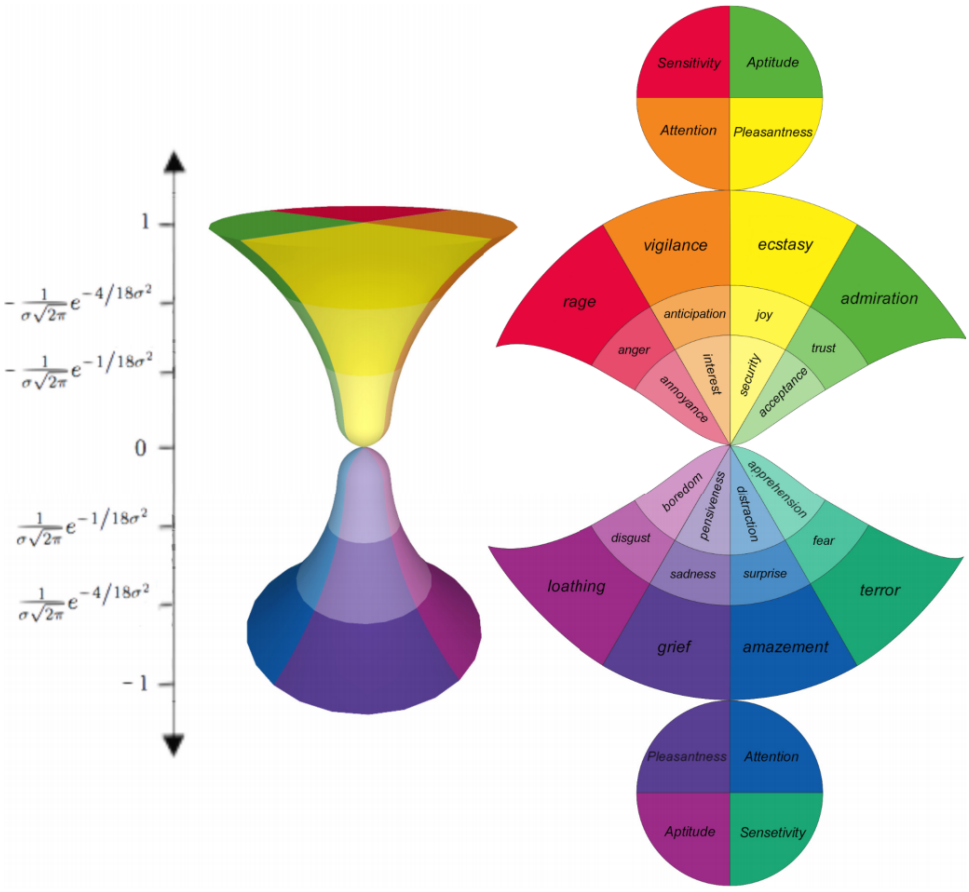


Figure 2.3: Hourglass Model of Emotions [9]

of Emotions. Each emotion can be represented as a $'sentic\ vector'$, based on the intensity on different dimension, each emotion can be characterized by unique four words in the table.

Table 2.1: The sentic levels of the Hourglass Model of Emotions

Interval	**Pleasantness**	**Attention**	**Sensitivity**	**Aptitude**
$[G(1), G(2/3))$	ecstasy	vigilance	rage	admiration
$[G(2/3), G(1/3))$	joy	anticipation	anger	trust
$[G(1/3), G(0))$	serenity	interest	annoyance	acceptance
$(G(0), -G(1/3)]$	pensiveness	distraction	apprehension	boredom
$(-G(1/3), -G(2/3)]$	sadness	surprise	fear	disgust
$(-G(2/3), -G(1)]$	grief	amazement	terror	loathing

The Hourglass of Emotion model is not only for emotion detection, but also can be used for polarity detection tasks. The polarity is defined in the following equation:

$$p = \sum_{i=1}^{N} \frac{(Pleasantness(c_i) + |Attention(c_i)| - |Sensitivity(c_i)| + Aptitude(c_i))}{3N} \tag{2.1}$$

where c_i is the input concept word and N is the total number of concept words and 3 is the isolation factor.

Overall, the Hourglass of Emotion model is designed to potentially describe the emotional experience which is suitable for every single person.

2.3 Aggregation Methods

2.3.1 Group Making and OWA operator

Group decision[11] is a task where several agents get involved in a decision process to generate a value that represents their individual decisions in the group process. In a classification group decision task, we would like to obtain a classification value which

can reflect the majority of all the classification agents and summarizes the collective value from those agents.

Such task of aggregating diverse methods/agents can be achieved by the OWA operator. Ordered Weighted Averaging (OWA) operator was first introduced by Yager [12]. OWA is an aggregation mechanism which can prioritize the classification of some methods based on the features of values those methods produced.

OWA operator Definition [12]: An OWA operator of dimension n is a mapping function F: $R^N \rightarrow R$, where for each dimension, there is a weight w_i associate with it. $w_i \in [0,1]$ and $\sum_{i=1}^{n}(w_i) = 1$ and is defined to aggregate a list of real values $a_1, a_2, ..., a_n$ according to the following expression:

$$F(a_1, a_2, ..., a_n) = \sum_{i=1}^{n} w_i \times a_{\sigma(i)} \tag{2.2}$$

Where $\sigma\{1, ..., n\} \longrightarrow \{1, ..., n\}$ is a permutation such that $a_{\sigma(i)} \geq a_{\sigma(i+1)}$

If B is the vector whose value is the ordered arguments values, e.g $b_i = a_{\sigma(i)}$, then

$$F(a_1, a_2, ..., a_n) = W^T B \tag{2.3}$$

OWA operator can achieve the concept of fuzzy majority, hence, it is one of the most commonly used operators in multi-criteria decision making. However, it is restricted to some portion of the criteria must be satisfied [13].

There is a more generic type of OWA operator introduced by Mitchell and Estrakh[14], where the inputs of OWA are not rearranged by the values but using a mapping function. Then, Yager introduced the more general type of the OWA operator, called the IOWA [15].

IOWA operator Definition [15]: An IOWA operator of dimension n is a mapping function F: $R^N \rightarrow R$, where for each dimension, there is a weight w_i associate with it, $w_i \in [0,1]$ and $\sum_{i=1}^{n}(w_i) = 1$. Thus, the equation of IOWA is defined as follows:

$$F(< u_1, a_1 >, ..., < u_n, a_n >) = \sum_{i=1}^{n} w_i \times a_{\sigma(i)} \tag{2.4}$$

where $\sigma\{1, ..., n\} \longrightarrow \{1, ..., n\}$ is a permutation function such that $u_{\sigma(i)} \geq u_{\sigma(i+1)}$, $\forall i = 1, ..., n-1$.

By this definition, the reordering of the set of values $a_1, ..., a_n$ are induced by the reordering of the set of values $u_1, ..., u_n$

The difference between OWA and IOWA operator is that they have a different way of reordering values. The reordering step of OWA operator is based on the magnitude of the values whereas the IOWA operator introduced a reordering function which mapping the value into weights and ordered by the weights.

There are two ways described in Yager's paper [16] for obtaining the associated weights. The first way is to learn the weights by sampling data and the second approach and the second one is to assign some semantics and weights to the weights.

According to Pasi and Yager [17], let $Q : [0, 1] \rightarrow [0, 1]$ be a function such that $Q(0) = 0$, $Q(1) = 1$, and $Q(x) \geq Q(y)$ for $x > y$ corresponding to a fuzzy set representation of a proportional monotone quantifier. Q(x) is the degree to which x satisfies the fuzzy concept being represented by the quantifier. The weighting vector is represented as follows:

$$w_i = Q(\frac{i}{n}) - Q(\frac{i}{n-1}) \tag{2.5}$$

There are some linguistic quantifiers introduced by Kacprzyk[18], that express the "at least half," "most of," and "as many as possible," using the formula in Equation 2.6 :

$$Q(r) = \begin{cases} 0 & if\ 0 \leq r < a \\ \frac{r-a}{b-a} & if\ a \leq r \leq b \\ 1 & if\ b < r \leq 1 \end{cases} \tag{2.6}$$

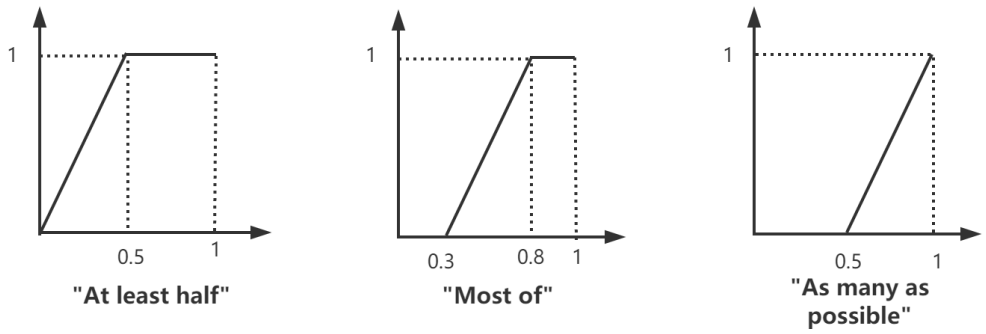


Figure 2.4: Linguistic quantifiers *at least half*", "*most of*" and "*as many aspossible*" [18].

In the Figure 2.4, they use $(0, 0.5)$, $(0.3, 0.8)$, $(0.5, 1)$ for $(a,\ b)$ respectively, which represents the semantic "*at least half*", "*most of*" and "*as many aspossible*".

2.4 Power Systems and Knowledge Graph

In the realm of the power industry, few studies have focused on domain-specific knowledge graphs application in power grid operation. Industrial application of knowledge graph at Siemens was an important step toward intelligent engineering and manufacturing which could highly improve workflow efficiency and data accessibility [19]. Tang [20] proposed an Enterprise-level information integration framework, enhanced power equipment management and improved efficiency in querying and classifying relevant information and updating product contents in realtime. Fan [21] proposed a method to construct the dispatch Knowledge graph for power systems which describes the behaviours of dispatchers semantically. They proposed the semi-auto labelling method to build a dispatcher-oriented corpus and a BiLSTM-CRF model was built and trained to extract the entities and create relationships for dispatcher behaviours. Yang [22] proposed a generic way of building enterprise-level knowledge graph in power field by fusing power transmission data and transformation assets with multi-source heterogeneous information, which helps to reduce information redundancy and

improve the accuracy of fusion. Su [23] proposed an automatic framework to extract ontological information which facilitates the sharing of multi-source heterogeneous power grid equipment data.

In the paper [24], single- and multi-threading methods for the analysis of grid energizing processes are designed. It was demonstrated that the graph database had better efficiency compared to a relational database. Xu [25] extracted a knowledge graph into topology and analyzed the relationship between the grid topology and power system reliability using the graph theory and statistical analysis.

In summary, all these efforts and studies confirm the increasing interests and needs for knowledge graph in the power grid domain.

2.5 Aspect Based Sentiment Analysis

Sentiment analysis (SA) [26], also known as opinion mining, has drawn increasing interests and needs in recent years. Based on granularity, sentiment analysis can be categorized into different tasks, including document-level sentiment analysis, sentence-level sentiment analysis, and aspect-level sentiment analysis [27]. Since document-level or sentence-level sentiment analysis cannot provide more detailed information sometimes, thus, the aspect-based sentiment analysis (ABSA) is proposed to identify the opinions of a specific target or aspects in reviews [28]. ABSA contains subtasks including aspect-category detection, opinion target extraction, opinion word extraction and aspect-level polarity classification. Opinion target, also known as aspect term, refers to the words or phrases in the sentence represents the features or entities towards which users show attitude. Opinion word, on the other hand, refers to the terms used to express an attitude towards the Opinion target. For example, in the sentence "the screen of the laptop is damaged, and it has a terrible battery life", the words "screen" and "battery life" are two opinion targets, and the words "damaged" and "terrible" are opinion words. Opinion target extraction (OTE) and Opinion word extraction (OWE) are two fundamental sub-tasks for aspect-level sen-

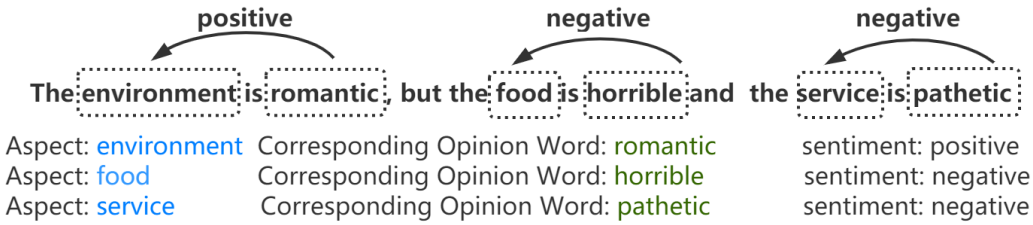


Figure 2.5: An example for the AOWE task and polarity detection for aspect analysis. The words in blue are two given aspects while the words in green are their corresponding opinion words. The arrows indicate the correspondence between aspects and opinion words.

timent analysis. In the recent year of research studies, there is many research work which has been done towards those two tasks. Xu [29] proposed a Dual Embedding CNN (DE-CNN) model with general and domain-knowledge embedding for extracting targets for product reviews. Yu [30] proposed a multi-task learning framework using Bi-LSTM to implicitly capture the relationships between the two tasks and results in an optimal prediction for both tasks.

Fan [31] defines a novel sub-task called aspect-oriented opinion words extraction (AOWE), where the objective is to extract the corresponding opinion word towards a specific aspect, Figure 2.5. They proposed a model called IOG, which refers to Inward-Outward LSTM plus Global context to solve AOWE task. Ying [32] proposed a model called Opinion Transmission Network that which exploit the connections between aspect-term polarity detection and AOWE and solve those two tasks simultaneously.

Chapter 3

Knowledge Graph Representation of Distribution Grid

3.1 Introduction

Distribution grid systems are complex networks containing multiple pieces of equipment. All of them interconnected, and all of them described a variety of pieces of information. A knowledge graph provides an interesting data format that allows us to represent information in a form of graphs, i.e., nodes and edges – relations between them. In this paper, we describe an application of a knowledge graph to represent information about a power grid. We show the main components of such a graph – called GridKG, a simple process of identifying electrical paths, and a few examples of grid analysis related to primary switches. The knowledge graph that integrates information about the topology of the power system with meta data about equipment and customers. We enhanced the knowledge graph with the data generated by algorithms we developed to identify upstream and downstream devices, as well as the number of customers connected to them.This additional data leads to a holistic view of the system. All this would allow us to gain further insight into the system's characteristics while analyzing the grid.

3.2 Definitions of Concepts and Relations

One of the most critical activities required for constructing any knowledge graph is 'building' so called vocabulary. It is a set of concepts and relations that are used to name nodes and edges of a graph. Due to a space limitation, we present the most important categories of concepts (for nodes) and relations (for edges).

3.2.1 Concepts Definition

Classes of concepts should reflect items and elements that constitutes main pieces of data that a graph suppose to represent. In the case of a distribution grid, it has been decided to use a concept of *Element* as a basic component representing any asset of the system. An *Element* has four attributes: *id* that is the same as the asset ID assigned to it by the utility, *no_of_customers* that are connected to a downstream path, and two coordinates x and y as identifiers of its geographical location.

Additionally, there are a number of concepts that provide information about *Elements*. Any piece of information about an *Element* is provided as a node connected/linked to it. They are:

cNode – a fictitious connection point, *Element* is connected to two of them: one upstream, and one downstream ('decided' after running Algorithm, see the Section 3.3); there are two unique nodes – *ENode* representing the last/end node, and an *SNode* that is the staring node;

Component Type – a type of *Element*, for example, *Primary Transformer*, *Primary fuse*, *Capacitor*, *Elbow*;

Feeder – an identification of a feeder to which *Element* is connected;

Service Area – a name of a service area where *Element* is located;

Connected Voltage – a voltage value of connected *Element*;

Phase Type – a phase to which *Element* is connected;

Location – a type of location of *Element*;

Customer – a no of customers (downstream) connected to *Element*;

A few illustrations how nodes of these categories are interconnected are provided in the next subsection.

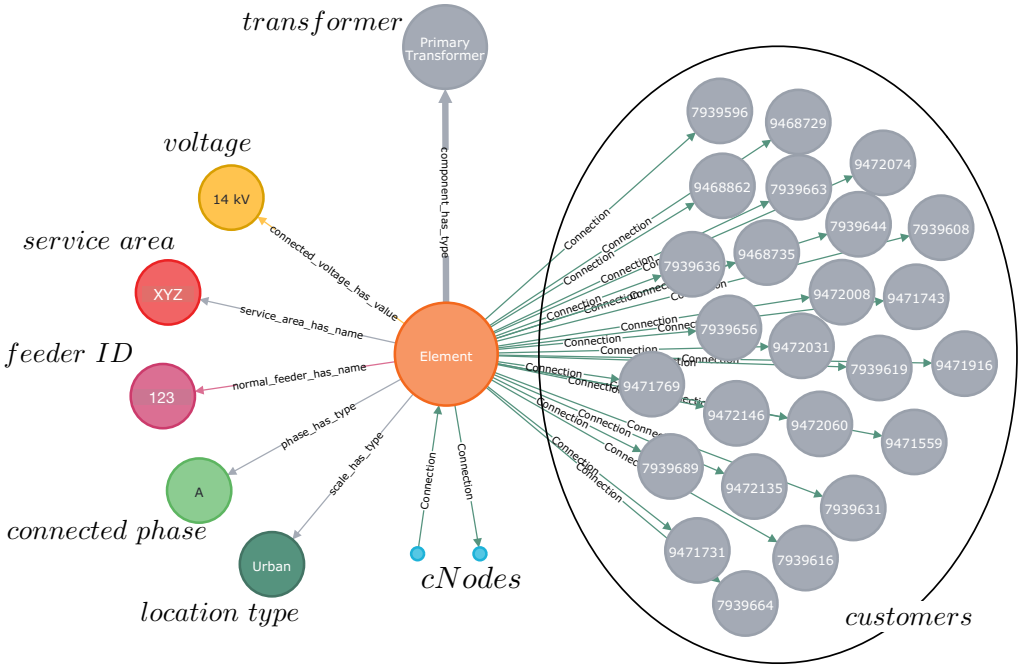


Figure 3.1: RDF triples with *Element* as their subject

3.2.2 Relations

Relations define a type of connection that exists between nodes. We are inclined to say that these connection constitute an essence of a graph-based representation. They allows to show how elements (nodes) are connected, and how they are linked to categories representing additional pieces of information. Here we have:

type of component – links *Element* to *Component Type*;

name of feeder – links *Element* to *Feeder*;

name of service area – links *Element* to *Service Area*;

connected voltage has value – links *Element* to *Connected Voltage*;

value of no of phases – links *Element* to *No of Phases*;

phase has type – links *Element* to *Phase Type*;;

connection – links *Element* to *cNode*;

Let us take a look at a few examples how elements are represented in the GridKG. The first example is already illustrated in Figure 3.1. Another one is shown in Figure 3.2. It displays *Elements* sorted by three different locations. An illustration of different types of *Elements* is included in Figure 3.3.

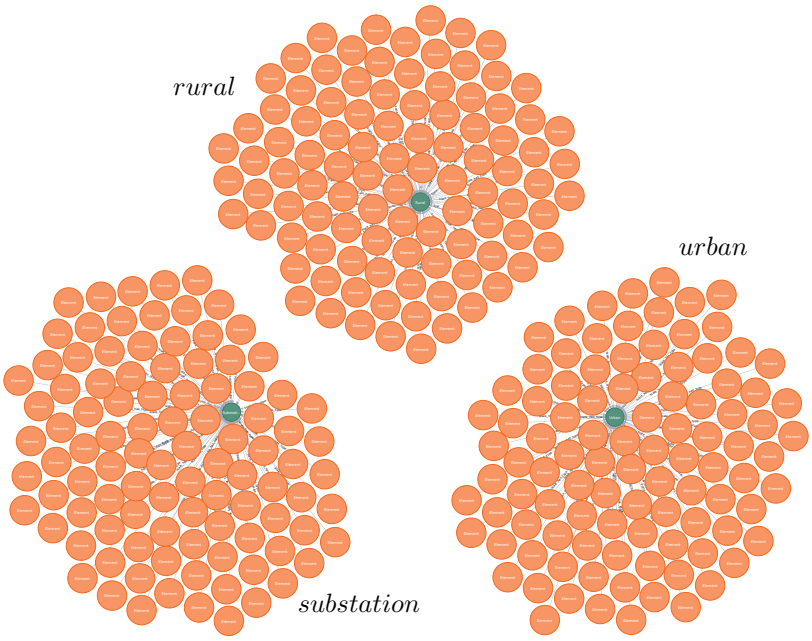


Figure 3.2: *Elements* by location type

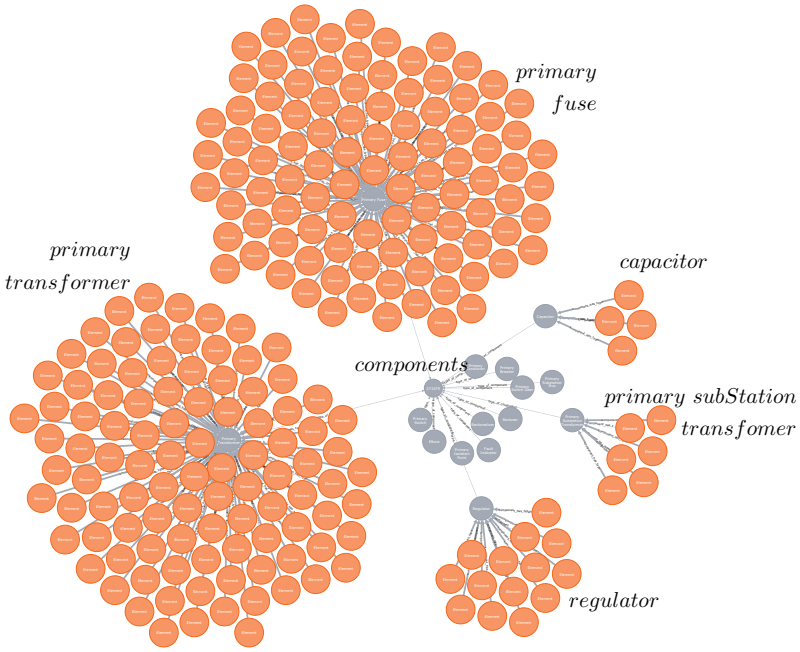


Figure 3.3: *Elements* by component type

The figures exhibit a characteristic feature of GridKG: nodes that represent electrical components and pieces of information describing them, as well as connections that link them together. Further, the connections – edges in a graph – clearly define relations that exist between nodes, i.e., components themselves, and components and information.

3.3 Algorithms for Processing Power Grid Knowledge Graph

An electrical path in a graph-based representation of grid is perceived as a sequence of tuples composed of two triples: $cNode_x$–$Connection \rightarrow Element_p$ and $Element_p$–$Connection \rightarrow cNode_y$. Therefore, a process of identifying paths means 'stitching' together multiple tuples in a way that $cNode_y$ of the predecessor tuple matches $cNode_x$ of the successor tuple.

The algorithm for identifying paths is shown below (Algorithm 1). At the beginning, all element-breakers and *cNodes* connected to them are put into a queue **Q** of elements to inspect for the purpose of finding other connected elements. Then a Breadth-First search is applied to find all adjacent elements and *cNodes* which satisfy the condition that the elements are not opened switches. A variable *level* is used as a property of elements to 'keep' track of elements' positions in the paths. For each element, the algorithm modifies the direction of edges from element of lower value of *level* to elements with higher value of *level*. The algorithm terminates when the queue **Q** is empty.

One of the additional benefits of identifying paths and their direction is generating new information about elements and adding it to the existing element nodes. For instance, by implementing a simple algorithm, we compute a number of customers for each element by a bottom-up approach. The number of customers for each element is determined by aggregating all the downstream customers. This highly can be beneficial to estimate the number of customers out of power after isolation of a grid

by a specific protection device.

Algorithm 1 Multi-Source Path Search

```
Initialization: Queue Q = [], level = 0, Sets: S ={}, nextElements = {}
Q ← list of pairs (cNode,breakers) i.e. Q ← { (n_1, b_1), (n_2, b_2), ... ,(n_n, b_n) }

while Q is not empty do
    size ← Q.size()
    for from 1 to size do
        (cNode, element) = Q.poll()
        if element.id ∈ S then                                  ▷ Element visited
            continue
        S.add(element.id)
        element.level ← level
        next_cN ← GetNextNode(element, cNode)
        nextElements ← GetNextEl(element, next_cN)
        for next_E in nextElements do
            SetConDirect(element, next_cN, next_E)
            if not (next_E = Switch & next_E.state = Opened) then
                Q.add(next_cN, next_E)
            end_if
        end_for
    end_for
    level ← level + 1
end_while
return
```

3.4 Utilization of Power Grid Knowledge Graph

3.4.1 Primary Switches on Path

Once electrical paths are identified, we can use GridKG to learn more about interconnection between its components. One of possible ways of learning more about the grid is finding out all downstream (and upstream) elements and connections from a given element of the system.

Listing 3.1: Code for Figure 3.4 code

```
MATCH
    path = (source:Element{id:1235})
    -[:Connection*]->(leaf:Element)
    -[c:Connection]->(eNode:ENode)
RETURN
    path, reduce(total = 0, e IN nodes(path) |
        CASE
        WHEN e.length IS NOT NULL
        THEN total + e.length
        ELSE total
        END ) AS totalLength
```

An example of a query in Neo4j query language Cypher is shown below. We provide a starting point/element and the query returns all downstream connected elements (shown in Figure 3.4). Additionally, it gives us a length of the electrical path – in our case it is equal to 911.34m. In Figure 3.4, we see information about types of elements, as well as their connected voltage and phase details.

3.4.2 Primary Switches and Conditions

Another type of analysis could lead to learning more about electrical path in the context of a number of protective devices present in the path. Such an information is very easy to retrieve from GridKG.

Below, we include a query that provides – as a result – a number of primary switches on the specified path, as well as all components of this path, Figure 3.5. We can identify the first and all subsequent protective devices on the upstream path from a specified element (top-left corner of the figure) to the primary breaker (down-right corner).

Listing 3.2: Code for Figure 3.5

```
MATCH
    path = (source:Element{id:1456})
    <-[:Connection*]-(root:Element)
    <-[c:Connection]-(sNode:SNode)
RETURN
    path,
    [ element IN nodes(path)
        WHERE
        (element)-[:component_has_type]
        ->(:Component_type
        {component_type: 'Primary Switch'})
    ] AS protectiveElements
```

Besides localization of the switches, we obtain more information about the path itself – its components, and information about them.

3.4.3 Elements and Conditions

As the last example of utilization of GridKG, we show a bit more involving query. This time, we want to know about the existence of primary switches that are in the

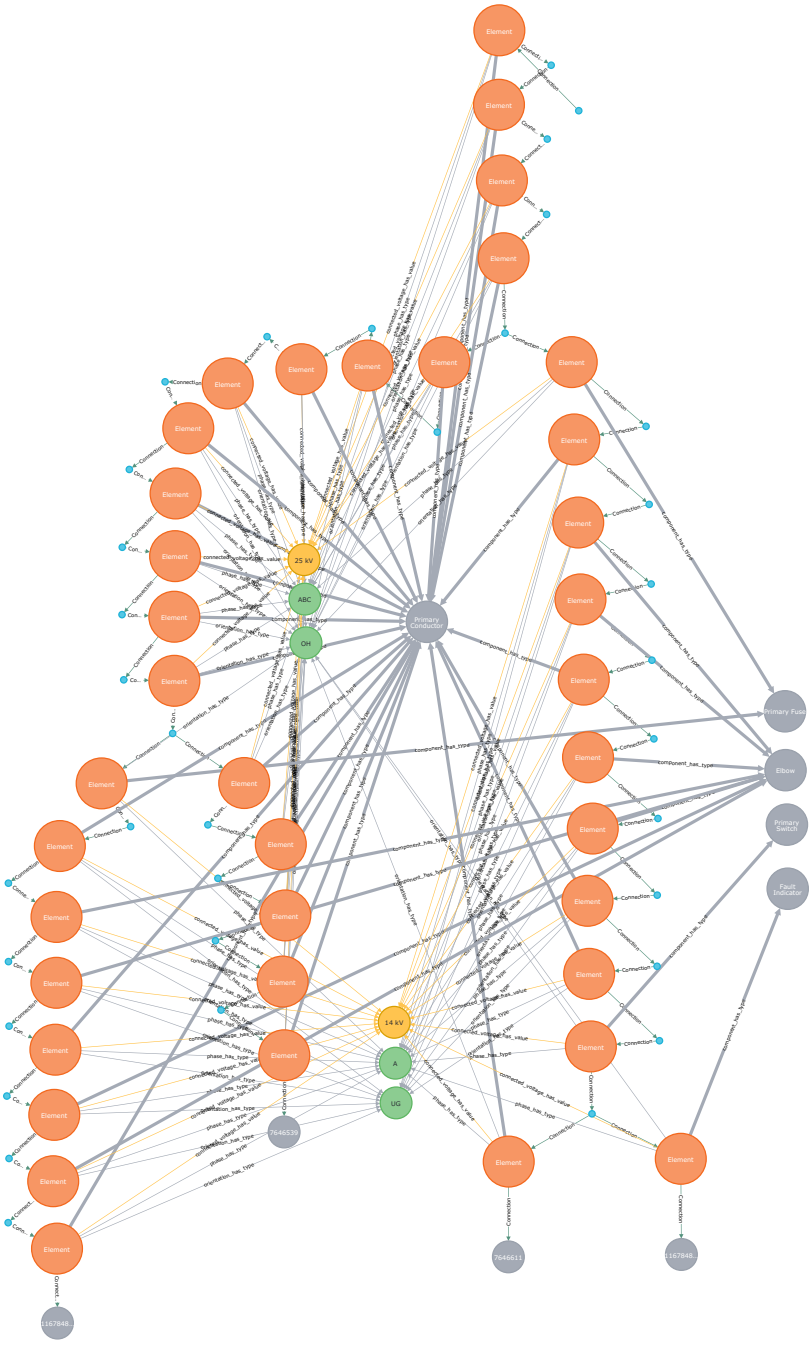


Figure 3.4: *Elements* of downstream paths: their types identified by links to gray circles where each circle represents a different type of electrical component; their voltages – yellow circles; and their phases and orientation – green circles.

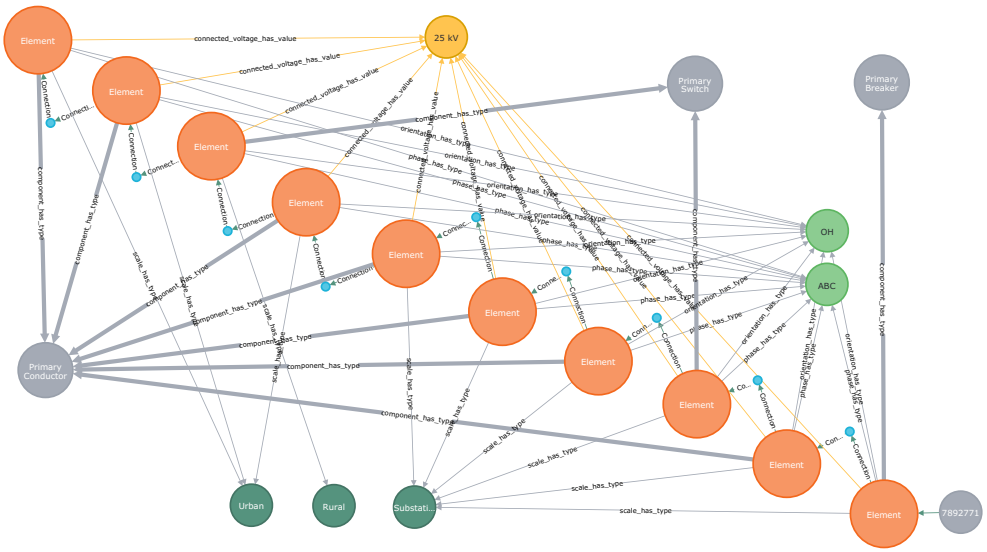


Figure 3.5: Location of Primary Switch/protective devices on the upstream path from top-left element to the primary breaker (down-right corner).

'XYZ' service area, connected to 14 kV, and located on paths that deliver power to more than 100 customers.

The query is below, and the obtained data is illustrated in Figure 3.6.

Listing 3.3: Code for Figure 3.6

```
MATCH
      (e:Element)
      -[:component_has_type]
      ->(c:Component_type
      {component_type: "Primary Switch"})
WHERE
      (e)-[:service_area_has_name]
      ->(:Service_area_name
      {service_area_name: "XYZ"})
AND
      (e)-[:connected_voltage_has_value]
      ->(:Connected_voltage_value
      {connected_voltage_value: "14 kV"})
AND e.num_customer > 100
RETURN e
```

3.5 Results and Conclusions

Today's distribution grids are complex networks constituted of multiple components. Power utilities collect and store, in relational databases, large amount of information about the grids' elements from transformers to individual poles. It is important for them to be able to have quickly access the data describing components, as well as

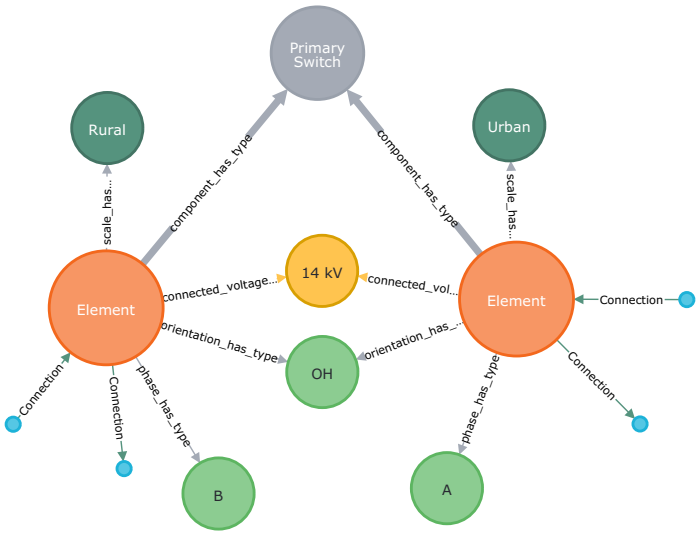


Figure 3.6: Primary switches in the 'XYZ' service area that connected to 14 kV and provide power to more than 100 customers downstream: they are of different scale, connected to different phases.

connections and relations between them.

We propose to use knowledge graphs as a suitable format for representing grid data. We describe some of the categories of nodes designed for representing different electrical elements and conceptual information describing those elements. We also define a number of relations between elements/concepts that are linked to edges connecting nodes of the graph.

Finally, we illustrate utilization of a distribution grid knowledge graph. We propose an algorithm for identifying electrical paths in the grid. Further, we include a few graph queries that take advantage of the identified paths and allow for: determining a length of downstream path from a specific element; determining a sequence of switches/protective devices on a given upstream path; as well as a set of switches that satisfy a condition related to downstream components.

Chapter 4

Emotion-based Analysis of Reviews using Knowledge Graph

4.1 Introduction

An continuous growing number of products and services means that a huge number of their reviews is generated. People express their opinions and quite often they direct some emotions towards the products and services they review. For both service owners and manufactures it is important to know what customers thinks, what they like and what they dislike – all this (could) have impact on what products and their features – aspects – to change, improve, or discontinue. It is even more interesting to find out what type of emotions the users have towards those services and products. Additionally, it would be quite appealing to have these reviews aggregated over multiple users and represented in a form of linguistic description of emotions the reviews have toward the reviewed items and products.

At the same time, advances in NLP techniques, development of the Hourglass Model of Emotions, deep learning methods for aspect sentiment analysis, and a semantically rich representation of data in a form of graphs create opportunities do develop and utilize new approaches for discovering new analysis of data.

Therefore, we propose application of NLP/DL based techniques to obtain linguistic description of aspects, and combine the identified aspects and their descriptions with a model of emotions. We use a knowledge graph as data representation format

and apply linguistic focused aggregation methods to provide multi-facet analysis and aggregation of reviews from the perspective of emotions of reviewers toward reviewed items and their aspects.

4.2 Model of Reviews, Aspects, and Emotions

A need to work with multiple reviews, aspects, and categories of items means that we need to represent them in a comprehensive and easily manageable way. This representation should be able to embody all that information in a relation-rich, flexible and expandable format. A model that addresses these needs and abilities is proposed here. It is a multi-dimensional detailed view of reviews that includes reviews themselves, aspects of items they refer to, description of those aspects, as well as emotions associated with them.

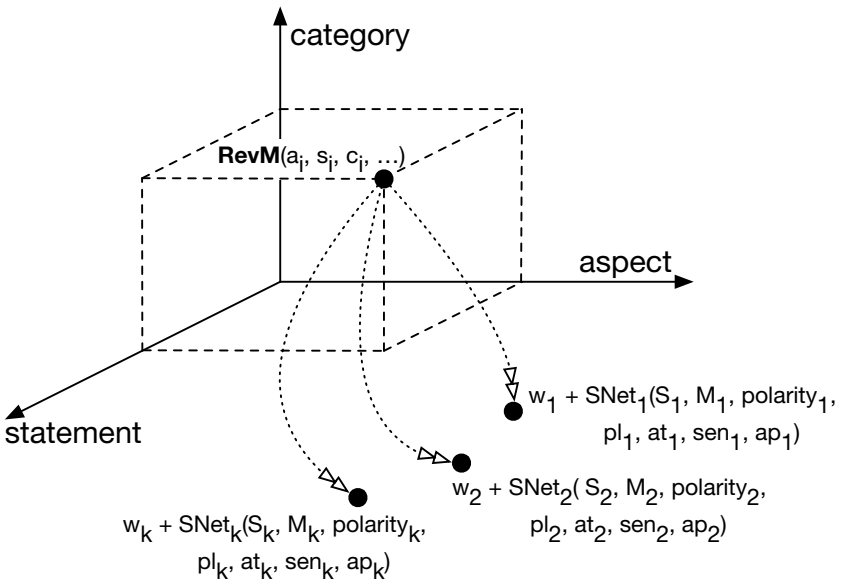
4.2.1 Model Overview

The advances in sentiment analysis techniques enable a through processing of sentences, see Section 2.5. As a result, we are able to obtain not only an aspect to which an identified sentiment refers to, but also a category to which it belongs to. Obtained data is the basis and input of our proposed model.

Our purpose is to enhance the results obtained from sentiment analysis methods with linguistic terms describing identified aspects. In other words, we want to find out justification behind the determined sentiment; we are interested in words that have been used to describe a given aspect. We apply a deep neural network to perform NLP and to find words directly associated/linked with aspects.

All this information allows us to construct quite a comprehensive model of reviews: it integrates information obtained from sentiment analysis, NLP processing and *SenticNet*. In general the model is a tuple:

$$\textbf{REmModel}(a_i, s_i, c_i, \{w_{n_i}, SN_{n_i}\}_{n_i=1,\ldots,N_i})$$



$$\textbf{RevM}(a_i, s_i, c_i, w_{n_i} + SNet_{n_i}(, S_{n_i}, M_{n_i}, polarity_{n_i}, pl_{n_i}, at_{n_i}, sen_{n_i}, ap_{n_i}), n_i{=}1, ..., k_i)$$

Figure 4.1: Set of triples describing **delicious** based on *SenticNet*: I) synonyms; II) Hourglass Model of Emotions; III) sentiment; and IV) mood-related words.

where: a_i is the aspect; c_i is its category while s_i is a sentence that contains the aspect. A set $\{w_{n_i}, SN_{n_i}\}$ contains pairs composed of a description word 'characterizing' the aspect and additional pieces of information obtained from the *SenticNet*:

$$SN_{n_i} = SNet_{n_i}(S_{n_i}, M_{n_i}, polarity_{n_i}, pl_{n_i}, at_{n_i}, sen_{n_i}, ap_{n_i})$$

In this case, we have a set of synonyms S_{n_i}, a set of mood tags M_{n_i}, as well as sentiment $polarity_{n_i}$ and its numerical value of intensity, and four values obtained from the Hourglass Model of Emotions: pleasantness – pl_{n_i}, attention – at_{n_i}, sensitivity – sen_{n_i}, and aptitude – ap_{n_i}. It should be mentioned that the sets S_{n_i}, and M_{n_i} could overlap, it means that all words and phrases are tightly interconnected.

4.2.2 Graph Schema

The presented above model of reviews is represented as a knowledge graph. To do it in a way that the graph is flexible, scale-able and easy to read also for a human

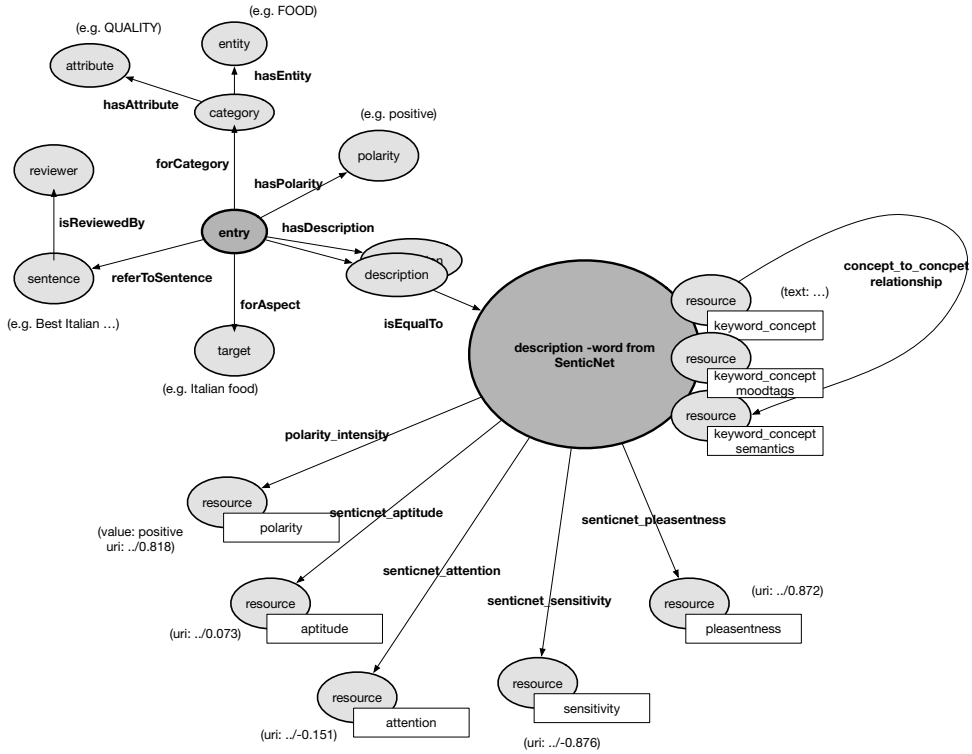


Figure 4.2: Schema for *Neo4j* implementation of **REmModel**.

we define a schema and vocabulary of nodes and connections. A visualization of it is presented in Figure 4.2.

In general, the schema is composed of two sets of nodes and relations: one of them to represent details of reviews, and the other one to represents the Hourglass Model of Emotions.

4.2.3 Building Graph: Loading Data

This subsection contains details about the used review data sets, a process of extracting description words related to aspects, and some information about *SenticNet*. At the end, we include a few snippets of the constructed graph representing our **REmModel**.

Review Data

In this section of the thesis, which is dedicated to the aspect based analysis processes, we use two data sets of reviews.

The first one is Caroline [33]. It is a data set that contains all manually annotated sentences with aspects from Foursquare comments. It includes about 215K user reviews, in English, of restaurants all over the world. There are 585 samples with 1006 sentences. Samples are annotated following the SemEval2016 annotation guidelines.

The second data set of reviews is built by Fan [31] based on data sets from SemEval containing the laptop and restaurant reviews. The original data sets have been annotated by authors. They identified opinion words for different aspects/targets. The annotation was performed by two individuals. In the case of conflicts or difficulties in finding corresponding opinion words, the respected entries were removed. The SemEval data sets are very popular benchmarks for many sentiment related subtasks, including Aspect category detection, Opinion Target Extraction, Opinion Words Extraction and Target-Dependent Sentiment Analysis.

We utilize the Deep Neural Network architecture proposed by Fan [31], called IOG (Inward-Outward LSTM Model with Global Context) to extract description words. IOG can effectively encode target information into the left and right context. The inward-LSTM runs from the first word to the opinion target as a forward-LSTM and from the last word to the opinion target as a backward-LSTM. It is used to pass information from context to target while Outward-LSTM is to pass the target to its content, which runs from the opinion target to both ends of the sentence. To better capture the global meaning from the sentence, the output from inward and output LSTM takes as the input to the Bi-LiSTM. The model is trained on the SemEval reviews and inference on the Caroline [33] data set to predict the description words for each aspect term.

As a result of identification of description words, a number of pairs: aspect-

description, we call them Review Entries, have been loaded into the knowledge graph. A total of more than 100k nodes have been created with almost 1,200k connections between them. Table 4.1, contains details regarding names of all nodes and their corresponding quantities.

Table 4.1: **REmModel** database summarization

Node Label	**Count**
Categories	11
Reviewers	885
Sentences	2541
Review Entries (pairs: aspect-description)	5804
Targets (Aspects)	1121
Identified Description words	974
SenticNet Entries	100000

SenticNet

A section of knowledge graph 'dedicated' to emotions is built based on *SenticNet* model. It contains 100k phrases, and each phrase is 'linked' with the information such as synonyms, mood-tags, and elements of the Hourglass Model of Emotions. For example, the whole portrayal of the word **delicious** is shown below is a pseudo-RDF format:

```
<rdf:Description rdf:about="http://sentic.net/api/en/concept/delicious">
    <rdf:type rdf:resource="http://sentic.net/api/concept"/>
    <text xmlns="http://sentic.net">delicious</text>
    <semantics xmlns="http://sentic.net">
        <concept xmlns="http://sentic.net" rdf:resource="http://sentic.net/api/en/concept/yummy"/>
        <concept xmlns="http://sentic.net" rdf:resource="http://sentic.net/api/en/concept/tasty"/>
        <concept xmlns="http://sentic.net" rdf:resource="http://sentic.net/api/en/concept/good_eat"/>
        <concept xmlns="http://sentic.net" rdf:resource="http://sentic.net/api/en/concept/creamy"/>
    <concept xmlns="http://sentic.net" rdf:resource="http://sentic.net/api/en/concept/edible"/>
    </semantics>
    <sentics xmlns="http://sentic.net">
        <pleasantness xmlns="http://sentic.net" rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.075</pleasantness>
        <attention xmlns="http://sentic.net" rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.076</attention>
        <sensitivity xmlns="http://sentic.net" rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0</sensitivity>
        <aptitude xmlns="http://sentic.net" rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.166</aptitude>
    </sentics>
    <moodtags xmlns="http://sentic.net">
        <concept xmlns="http://sentic.net" rdf:resource="http://sentic.net/api/en/concept/interest"/>
        <concept xmlns="http://sentic.net" rdf:resource="http://sentic.net/api/en/concept/admiration"/>
    </moodtags>
    <polarity xmlns="http://sentic.net">
        <value xmlns="http://sentic.net">positive</value>
```

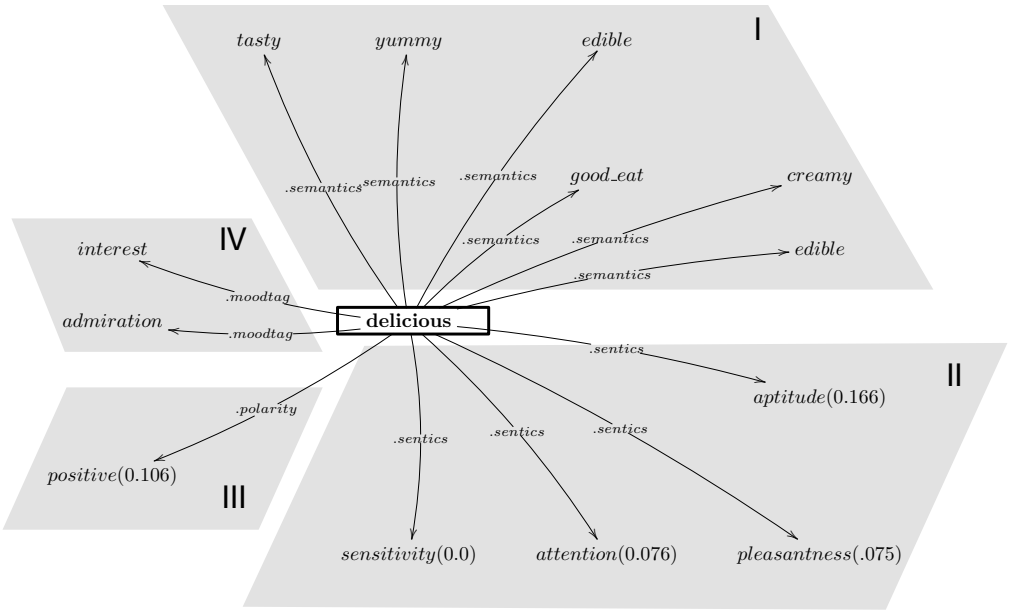


Figure 4.3: Set of triples describing **delicious** based on *SenticNet*: I) synonyms; II) Hourglass Model of Emotions; III) sentiment; and IV) mood-related words.

```
        <intensity xmlns="http://sentic.net" rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.106</intensity>
    </polarity>
</rdf:Description>
```

The presented above RDF description of the word **delicious** in the *SenticNet* is shown in a graphical form in Figure 4.3.

Visual Samples

Let us show a few snippets of the knowledge graph implementation of our **REm-Model**. Figure 4.4 is a fragment that shows Review Entries for the aspect **food**. We can observe that there are three categories of the Review Entries: ones that are positive, ones that are neural, and negative ones. There are decryption words (red bubbles) that are linked with aspects, and well as sentences from which the pairs <aspect-description word> have been taken.

A single description word **delicious** together with all Review Entries which use this word as a description are shown in Figure 4.5. The figure also contains 'a link' to *SenticNet* entry **delicious**, as well as a number of words that are synonyms to **delicious**. The synonyms are shown again in Figure 4.6.

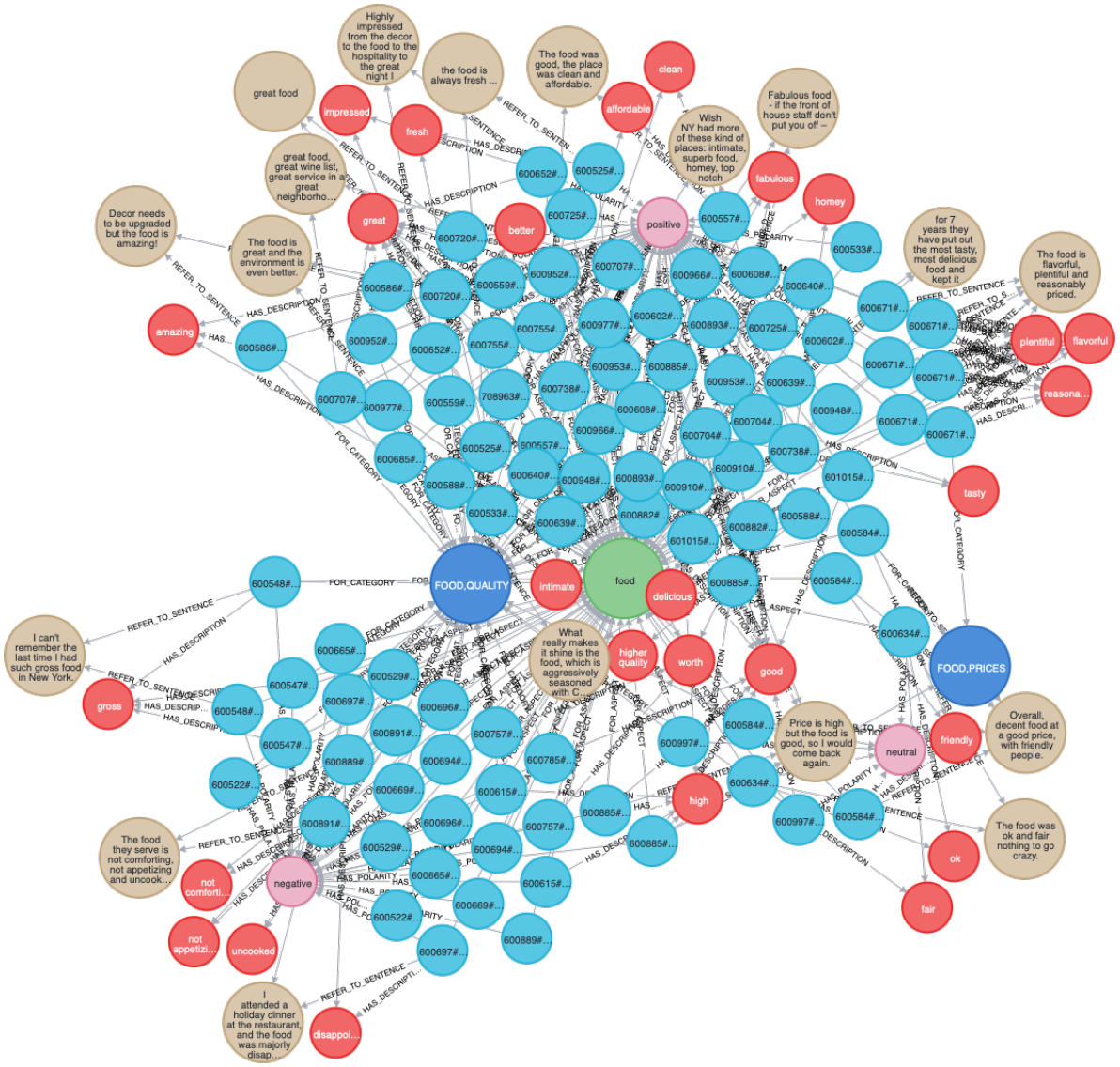


Figure 4.4: Snippet of **REmModel** for the aspect **food**: turquoise – Review Entries; beige – sentences; red – description words; pink – sentiment (positive, neutral, negative); dark blue – category entry.

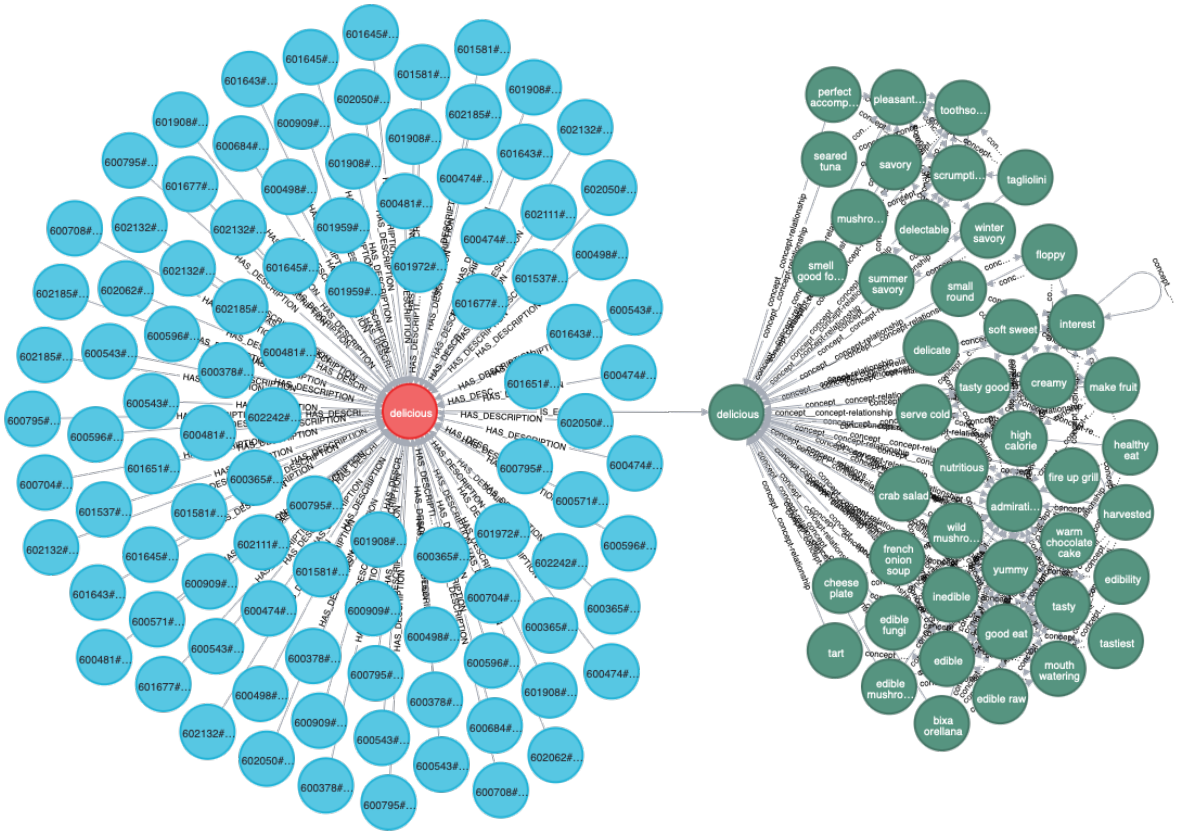


Figure 4.5: Fragment of **REmModel** illustrating integration of Review Entries (blue) with the words related to **delicious** from *S*enticNet (green) accomplish via the relation *is_equal* for the two nodes of **delicious** – one from the reviews and one from the *SenticNet*.

Figure 4.6: Fragment of **REmModel** for the word **delicious**: synonyms taken from the *SenticNet*.

4.3 Aspect-based Analysis: Utilization of KG-based Model

4.3.1 Analysis using Single Description Words

As an example of utilization of the proposed **REmModel**, we analyze a single aspect **food** that has been mentioned in multiple sentences in the presented data set. A few examples of statements from different reviews for the aspect **food** are:

> *'I attended a holiday dinner at the restaurant, and the food was majorly disappointing'*;
>
> *'The food was great and tasty, but the sitting space was too small, I don't like being cramp in a corner.'*; and
>
> *'Price is high, but the food is good, so I would come back again'.*

We start the analysis with identification of description words that are used to characterize and provides the reviewer's perception of the aspect. It means, we query our model to obtain all description words of the term **food**. Table 4.2 contains a few examples of words that have been identified. A total of 541 description words is retrieved. As we can see, they are in a wide range from very 'positive' terms, like *great*, *excellent*, *yummy* to very 'negative' ones: *disappointed*, *awful* or even *horrible*.

Table 4.2: Description words for the aspect **food**

Data Set	Description Words
NY data	*'crave', 'horrible', 'dirty', 'excellent', 'authentic', 'healthy', 'beat', 'old', 'best', 'nice', 'yummy', 'great', 'good', 'homemade', 'not good', ...*
Test data	*'overrated', 'recommend', 'horrible', 'acceptable', 'inconsistent', 'professional', 'helpful', 'disappointed', 'arrogant', 'speaks for itself', 'well prepared', 'refilled', 'drenched', 'not brilliant', 'lousy', 'intimate', 'best', 'gross', 'mediocre', 'appropriate', 'awful', ...*

Further, we proceed to extract from the model the values representing four emotions associated with each description word. Again, we show a sample of such values in Table 4.3. To illustrate how different values are linked with different words, we have selected a number of words representing very emotionally dissimilar states.

Once the emotion values are available, we aggregate them emotion-wise, i.e., each type emotion individually, per column in Table 4.3. The process of aggregation is controlled by OWA (Section 2.3). As we have mentioned, OWA allows us to perform an aggregation in a number of ways where each of them is represented by a different degree of 'orness'. We can have a very optimistic way – a high degree of 'orness' – or very pessimistic one with a very low value of 'orness'. We also use linguist quantifiers to govern an aggregation process. In this work, we use three different aggregation approaches:

Table 4.3: Values of emotions for selected description words

Word	aptitude	pleasantness	sensitivity	attention
'great'	0.892	0.905	0.000	0.773
'excellent'	0.706	0.781	0.000	0.000
'yummy'	0.113	0.076	0.000	0.000
...				
'disappointed'	0.000	-0.840	0.898	0.000
'awful'	-0.620	-0.080	0.615	0.202
'horrible'	-0.940	-0.930	-0.930	0.000
...				

1) optimistic with a level of 'orness' equal to 0.67;

2) neutral characterized by the value of 'orness' equal to 0.50;

3) pessimistic ruled by a linguistic quantifier $MOST$ leading to the value of 'orness' of 0.33.

The values obtained via application of OWA are presented in Table 4.4. As intuition dictates, we see decrease in the value of each emotion when we start with optimistic and move towards a pessimistic view.

Table 4.4: Aggregated values of emotions of words describing the aspect **food**

Aggregation	aptitude	pleasantness	sensitivity	attention
optimistic	0.557	0.640	0.138	0.530
neutral	0.316	0.412	-0.033	0.295
pessimistic ($MOST$)	0.037	0.119	-0.128	0.045

As much as the numerical values are essential, it is more interesting and attractive to use linguistic descriptions of aspects. These descriptions should be determined based on the values of emotions obtained via aggregation of reviews. The approach

we applied here is quite simple – we query our model to retrieve the closest words to a vector built from each emotion value.

For our exemplary aspect **food**, we query the model **REmModel** using three vectors, Table 4.4, representing three levels of optimizing, i.e., 'orness' of the aggregation process. The obtained words are presented in Table 4.5. It can be stated that the found words reflect three different approaches to aggregate the reviews. It is especially visible for the optimistic one – the phrases *order lunch, spread idea* – reflect a positive option regarding the aspect **food**. The words *serve cold, hard find* that have been selected to represent a natural (simple averaging of emotion values) aggregation are reasonable, yet somehow less coherent. For the case of an pessimistic method of aggregation, we see a bit of mixture. The word *think clearly* could somehow reflect the emotional evaluation of the aspect from a negative point of view, yet the other word *culture* is more difficult to justify and explained.

Table 4.5: Linguistic 'view' of emotions linked to the reviews of **food**

Aggregation	most relevant phrases
optimistic	*order lunch, spread idea*
neutral	*serve cold, hard find*
pessimistic (*MOST*)	*culture, think clearly*

Additionally to the presented above approach, we provide another – linguistic-based – form of representing the obtained four emotions. For this purpose, we apply a procedure of converting numerical values into a 2-tuple fuzzy linguistic representation [34]. The process is based on identifying the most suitable linguistic terms based on a numerical value using a verbal description of six different states defined for each emotion in the Hourglass Model of Emotions, Table 2.1).

In order to do this, we propose a linguistic representation structure that uses the linguistic terms representing the states of emotions. The range of values of each emo-

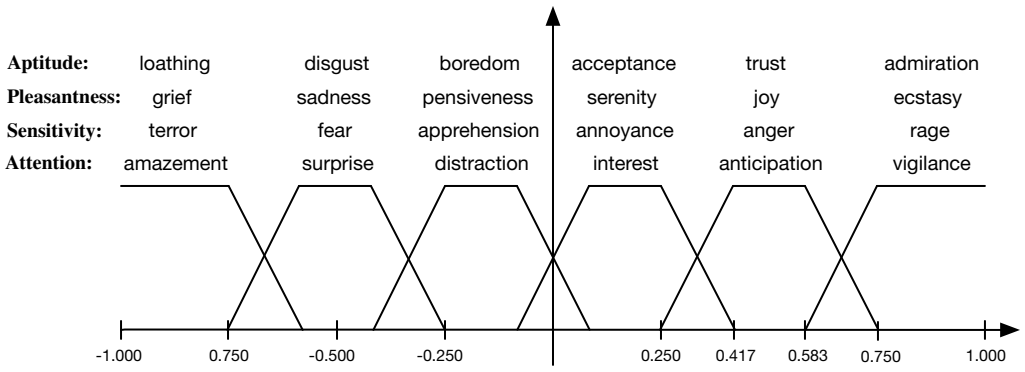


Figure 4.7: Membership functions and linguistic terms defined for emotions.

tion is treated as the universe of discourse $U = \langle -1, 1 \rangle$ on which the six linguistic terms are defined. As mentioned earlier, these terms indicate different states of emotion identified for each type of emotion. The terms are showed in Table 4.6. They are arranged in the vertical form reflecting the range of the universe of discourse, from -1 (top) to 1 (bottom). A more illustrative way is depicted in Figure 4.7. As we can see, the six linguistic terms are equally 'spread' in the universe of discourse.

Table 4.6: Linguistic terms defined for emotions (the left column indicates the universe of discourse, from -1 to 1)

$U = \langle -1, 1 \rangle$	aptitude	pleasantness	sensitivity	attention
$\langle -1, \ldots$	loathing	grief	terror	amazement
$\ldots$	disgust	sadness	fear	surprise
$\ldots$	boredom	pensiveness	apprehension	distraction
$\ldots$	acceptance	serenity	annoyance	interest
$\ldots$	trust	joy	anger	anticipation
$\ldots, 1 \rangle$	admiration	ecstasy	rage	vigilance

Using the defined structure of linguistic terms and their membership functions, we map the obtained emotion values to a more human-friendly format.

The approach presented in [34] is applied here for translating the values into linguistic terms. Each value of emotion is represented as a term and a degree to which

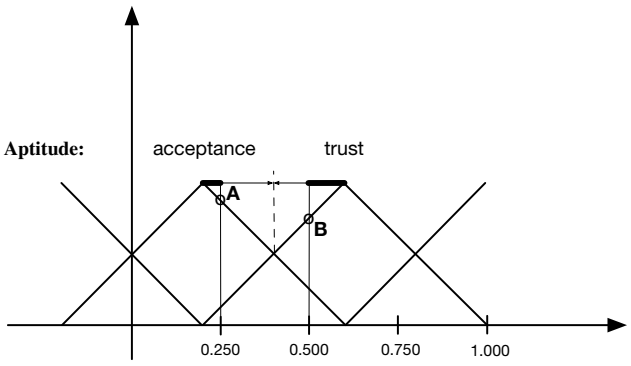


Figure 4.8: Translation of values into 2-tuple their linguistic representation.

this term is 'satisfied' by the value. Fig. 4.8 illustrate the approach for the emotion of aptitude and two terms: *acceptance* defined as a triangular fuzzy function (-0.20, 0.20, 0.60) and *trust* as a fuzzy function (0.20, 0.60, 1.00).

Let us take a value A=0.250. Following the approach proposed in [34], the value is mapped into the term *acceptance* as the first element of a tuple, and then the second element of this tuple is the value of A minus the centre of *acceptance*, i.e., 0.250-0.200. The value is then normalized into the interval (0.0, 0.5). The normalization factor is determined using the distance form the centre of the term *acceptances* to the 'middle point' between *acceptance* and *trust*. In our case, Fig. 4.8, the normalization factor is equal to 0.4 (0.2/0.5). Therefore, the obtained 2-tuple representation is (*acceptance*, +0.125). Following the same process, another point: B = 0.500 is translated into the 2-tuple (-0.250,*trust*he).

The linguistic descriptions, i.e., 2-tuples, of emotions obtained via the aggregation process, Table 4.4, are included in Tables 4.7, 4.8, 4.9. As we can see that each 2-tuple provides us with two pieces of information: the most suitable linguistic terms together with the value from the range -0.5 to 0.5 that indicates 'deviations' to the left or right from the centre of the term.

Table 4.7: Linguistic description of the aspect **food**: optimistic aggregation

	optimistic
Emotion:	
aptitude:	(-0.11, **trust**)
pleasantness:	(**joy**, +0.10)
sensitivity:	(-0.16, **annoyance**)
attention:	(-0.18, **anticipation**)

Table 4.8: Linguistic description of the aspect **food**: neutral aggregation

	neutral
Emotion:	
aptitude:	(**acceptance**, +0.29)
pleasantness:	(-0.47, **joy**)
sensitivity:	(**apprehension**, +0.42)
attention:	(**interest**, +0.24)

4.3.2 Analysis using Clusters of Description Words and their Synonyms

The presented analysis of emotions associated with the reviews of a given aspect is based on the description words that directly characterize a given aspect. In this section, we propose and describe a bit different approach that leads to the broader and more expressive characterization of aspects. The idea is to increase a set of description words, group them, and use the most representative ones from each group to characterized the investigated aspect.

In order to accomplish that we execute the following process. We extend the set of words that describe a considered aspect via extracting from *SenticNet* 'emotional' synonyms of the previously identified description words. The extracted words con-

Table 4.9: Linguistic description of the aspect **food**: pessimistic aggregation

	pessimistic
Emotion:	
aptitude:	(-0.41, **acceptance**)
pleasantness:	(-0.20, **serenity**)
sensitivity:	(**apprehension**, +0.18)
attention:	(-0.39, **interest**)

stitute a tree-like structure of synonyms. Each word/synonym is represented as a 4-dimensional vector with aptitude, pleasantness, sensitivity, and attention as individual dimensions.

Further, we use the trees of synonyms and vector representations of their nodes (words) to build a directed, weighted graph. This directed graph is clustered. The clusters are groups, formed by similarity between vectors, of words that are utilized to provide yet another description of the investigated aspect.

Extraction of synonyms from *SenticNet* is done in a very simple way due to the knowledge graph representation of the Hourglass Model of Emotions. A *Cypher* query on Neo4j executed for each description word provides as with a set of trees of synonyms – one tree, called hereafter a *synonym tree*. A single description word is associated with a single synonym tree. It is very easy to control the depth – number of levels – of synonym trees. The words/synonyms are nodes of the trees, while edges represent <word-its synonym> relations. It should be stated that the process of extracting synonyms for a word already in the tree is controlled by checking a distance of this word from the root word. In a case, the word is 'too far' (irrelevant) to the root it is not 'unfolded' anymore. An example of a synonym tree for the word **delicious** is shown in Figure 4.9.

The obtained trees of words and their synonyms together with 4-dimensional vec-

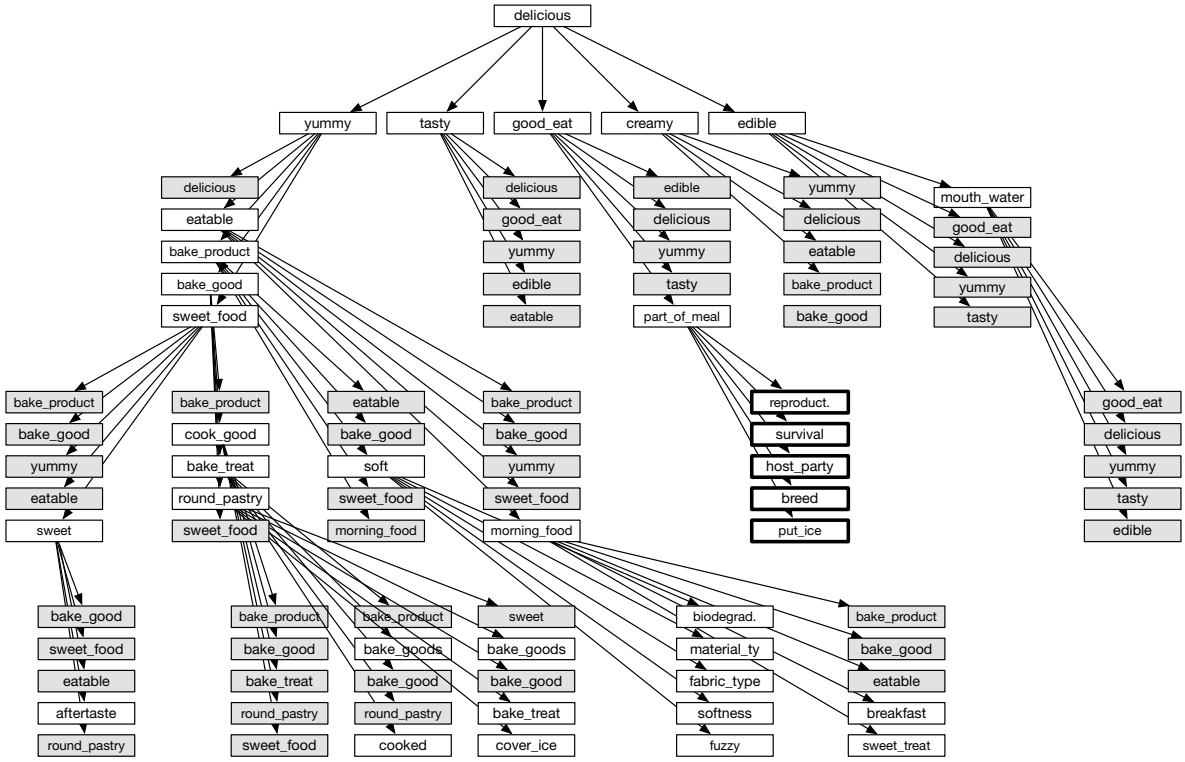


Figure 4.9: 3-level synonym tree for **delicious**: words in the thin white boxes are further 'unfolded' unless they are at the 3rd level, words in the thick white boxes are not unfolded due to their irrelevance (large distance to the root word), words in the gray boxes already exist in the tree.

tors representing them are used to construct a directed graph of words. This graph is built by fusing all synonym trees. In this graph, each word is uniquely represented by a single node what means that nodes from the synonym trees representing the same word are collapsed into one node. At the same time, edges from synonym tress are taken and inserted into the directed graph. In many cases, this results in multiple edges existing between two words going in both directions, as well as nodes (words) which are connected to multiple other nodes (a one to many scenario). The multiple edges between two different nodes (words) are fused into two – one in each direction – and weights reflecting a number of fused edges are assigned to each of them.

A single weight is computed in the following way:

- for each pair <word$_A$, word$_{B_i}$ > (where i is a number of other words to which word$_A$ is connected) an Euclidean similarity is calculated;
- each similarity value is multiplied by a number of edges between <word$_A$, word$_{B_i}$ >, as a result a weight for each edge from word$_A$ to word$_{B_i}$ is determined;
- the weights are normalized using softmax, and these values are assigned as weights to single edges between <word$_A$, word$_{B_i}$ >.

We use the Markov Cluster algorithm (MCL) [35] as the clustering algorithm to detect clusters in the directed graph we have been created. MCL is an unsupervised clustering algorithm on the graph with the process of stochastic flow simulation. When running the MCL algorithm, there is a hyper-parameter called inflation. It controls the cluster granularity: increasing inflation increases granularity, thus results in more clusters. We use modularity [36] as the measurement to optimize the inflation parameter. The modularity can be considered to be the fraction of graph edges which belong to a cluster minus the fraction expected due to a random chance, where the value of it lies in the range from -1 to 1. Higher positive modularity values implies higher clustering quality. We tune the MCL algorithm by selecting the inflation with highest modularity value.

We use obtained clusters to describe an investigated aspect. We retrieve centers of the clusters – and in many cases the two most representative words of each cluster – and apply these words to construct a new description of the aspect. The fact that words in these clusters have been grouped based on the values of emotions (4-dimensional vectors crated based on values of aptitude, pleasantness, sensitivity, and attention) we can state that they represent an aggregated information about emotions. Further, we can say that these emotions reflect states of mind, moods of reviewers who provided their opinions about the considered aspect.

Of course, we should have in mind that clustering of description words and their synonyms could provide a bit too broad view. Another issue that should be consider it that these grouping of words is done based on the similarity of emotions. It means that the words themselves could be perceived as irrelevant, yet they represent 'emotional' similarity.

Let us take a look at one example of centers of clusters. Following our example of the aspect **food** we extract synonyms of description words from our model. The retrieving process has been performed up to the third depth-level in the case of synonym trees; please look at Figure 4.9 for an example of such a tree. After construction of the directed graph we have cluster it. As a result we obtained 15 clusters of words. Once again, we would like to emphasize the fact that the clustering is performed based on 'emotional' similarity. The obtained words for the aspect **food** are in presented in Table 4.10.

As it can be seen, we have obtained a very interesting set of new words that describe the aspect **food**. They provide a summary of all reviews using not only words directly related to our aspect. Among all presented words some of them seems to be very adequate, like *amazing, dumbstruck, creamy, lovely, healthy, yucky, gross*; while some seem to be quite distant, such as *demand, ancient, similar, glimpse, mass measure, same*. Again, these words should invoke emotions about the reviewed food and some look like descriptions while some look more like metaphors.

Table 4.10: Representative words of clusters as words describing the aspect **food**

Cluster	words
Cluster_00	astonishment, amazing
Cluster_01	dumbstruck, dumb-stricken
Cluster_02	demand, attentive
Cluster_03	sweet, creamy
Cluster_04	sheepishness, confusedly
Cluster_05	ancient artifact, ancient
Cluster_06	merit, great
Cluster_07	similarity, similar
Cluster_08	solid, glimpse
Cluster_09	mass measure, description
Cluster_10	good look, gorgeous
Cluster_11	lovely, familiar
Cluster_12	same, all right
Cluster_13	yucky, soapy
Cluster_14	gross
Cluster_15	healthy, live long life

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Knowledge graphs become more and more popular. We see their applications in many different domains from e-commerce, via many fields of engineering, governmental open data, to a variety of medical areas. These semantically rich data structures seem to provide a number of benefits: simplicity in collecting and storing streaming data, flexibility and easiness of modifications and changes in data structures/schema, and most in enabling to process and analyze data from the point of view of relations between individual pieces of data.

In this thesis we have applied knowledge graphs as data structures to two very different domains: power systems and analysis of reviews. It has been important to gain a first-hand experience in application of graphs and acquire practical knowledge of all processes related to their construction and practical and effective utilization.

The work presented in the thesis allows us to confirm a number of important aspects related to utilization of knowledge graphs.

- There is no doubt that building a vocabulary used for naming nodes and relations is a critical activity that should be done as the first step. It seems extremely important for multiple reasons: easiness of construing a graph, it means developing procedures and scripts for translating data into graphs; ability to query and retrieve information in a simple and intuitive way; and easiness

in interacting with graphs to develop data processing algorithms and procedures.

- Graphs as semantically rich structures offer a very easy and efficient access to relations between pieces of data. The ability to identify, extract, update, and create relations is unprecedented. There have been no problems with retrieving data points connected via a large number of relations, as well as creating new relations span across multiple data nodes.

- Graphs seem to provide a good framework for developing methods and algorithms for processing existing relations and information stored in graphs in order to augment them with additional data and information generated.

From the point of view of application of graphs to specific domains, we have been able to achieve the following contributions.

- In the area of power systems:
  - we have integrated information about topological structure of grid with data describing different pieces of equipment related to their operations and maintenance;
  - we have equipped a knowledge graph with new pieces of information via deploying algorithms for processing data about relations, this has led to a more comprehensive view of the system;
  - we have used the grid representing graph to retrieve, in a very easy way, information that otherwise requires an extensive manipulation and analysis of data extracted from multiple sources.

- In the are of analysis of reviews:
  - we have combined results of NLP-based analysis of reviews with the Hourglass Model of Emotions in a straightforward way building the **REm-**

Model that contains aspects, words describing these aspects, ad emotions associated with these words – all of them interconnected;

– we have used the graph-based model of reviews and emotions to extract options related to a specific aspect over a number of reviews, and aggregated that information to obtain a simple and more human friendly description of the aspect taking into account emotions associated with words and phrases used to review the aspect by multiple reviewers.

5.2 Future Work

The presented work on application of knowledge graphs to the domains of power system and review analysis can be treated as an onset for future work. We anticipate to continue working on building more comprehensive graphs and their more advanced utilization. In particular, we envision:

- integrating a graph-based model of the electrical grid with algorithms and process that allows us to simulate different events occurring the modeled system and in investigate their effects;
- adding more information of different nature, for example maintenance one, to the existing graph to create more comprehensive view of the system;
- developing methods and procedures that better utilize model of emotions and reviews in the context of analysis not only reviewed aspects of different items but also items and reviewers themselves;
- applying the graph-based model of emotions and reviews to variety of domains, for example, electronics and tourism.

Bibliography

[1] Y. Tang, T. Liu, G. Liu, J. Li, R. Dai, and C. Yuan, "Enhancement of power equipment management using knowledge graph," in *2019 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia)*, IEEE, 2019, pp. 905–910.

[2] R. Navigli and S. P. Ponzetto, "Babelnet: Building a very large multilingual semantic network," in *Proc. of the 48th annual meeting of the association for Comput. linguistics*, 2010, pp. 216–225.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*, Springer, 2007, pp. 722–735.

[4] *A lexical database for english*, https://wordnet.princeton.edu/, Accessed: 2020-08-11.

[5] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *Proc. of the 2012 ACM SIGMOD Int. Conf. on Management of Data*, 2012, pp. 481–492.

[6] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proc. of the 20th ACM SIGKDD Int. Conf. on Knowl. discovery and data mining*, 2014, pp. 601–610.

[7] J. Webber, "A programmatic introduction to neo4j," Oct. 2012, pp. 217–218. DOI: 10.1145/2384716.2384777.

[8] E. Cambria, S. Poria, D. Hazarika, and K. Kwok, "Senticnet 5: Discovering conceptual primitives for sentiment analysis by means of context embeddings," in *AAAI*, 2018.

[9] E. Cambria, A. Livingstone, and A. Hussain, "The hourglass of emotions," in *COST 2102 Training School*, 2011.

[10] Y. Susanto, A. G. Livingstone, B. C. Ng, and E. Cambria, "The hourglass model revisited," *IEEE Intelligent Systems*, vol. 35, no. 5, pp. 96–102, 2020. DOI: 10.1109/MIS.2020.2992799.

[11] F. Herrera and E. Herrera-Viedma, "Linguistic decision analysis: Steps for solving decision problems under linguistic information," *Fuzzy Sets and Systems*, vol. 115, pp. 67–82, Oct. 2000. DOI: 10.1016/S0165-0114(99)00024-X.

[12] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 183–190, 1988. DOI: 10.1109/21.87068.

[13] O. Appel, F. Chiclana, J. Carter, and H. Fujita, "A consensus approach to the sentiment analysis problem driven by support-based iowa majority," *International Journal of Intelligent Systems*, vol. 32, no. 9, pp. 947–965, 2017. DOI: https://doi.org/10.1002/int.21878. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/int.21878. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/int.21878.

[14] H. B. Mitchell and D. D. Estrakh, "A modified owa operator and its use in lossless dpcm image compression," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 5, no. 4, 429–436, Aug. 1997, ISSN: 0218-4885. DOI: 10.1142/S0218488597000324. [Online]. Available: https://doi.org/10.1142/S0218488597000324.

[15] R. R. Yager and D. P. Filev, "Induced ordered weighted averaging operators," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 2, pp. 141–150, 1999. DOI: 10.1109/3477.752789.

[16] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 183–190, 1988. DOI: 10.1109/21.87068.

[17] G. Pasi and R. Yager, "Modeling the concept of majority opinion in group decision making," *Inf. Sci.*, vol. 176, pp. 390–414, Feb. 2006. DOI: 10.1016/j.ins.2005.07.006.

[18] J. Kacprzyk, "Group decision making with a fuzzy linguistic majority," *Fuzzy Sets and Systems*, vol. 18, no. 2, pp. 105 –118, 1986, ISSN: 0165-0114. DOI: https://doi.org/10.1016/0165-0114(86)90014-X. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016501148690014X.

[19] T. Hubauer, S. Lamparter, P. Haase, and D. Herzig, "Use cases of the industrial knowledge graph at siemens," in *International Semantic Web Conference*, 2018.

[20] Y. Tang, T. Liu, G. Liu, J. Li, R. Dai, and C. Yuan, "Enhancement of power equipment management using knowledge graph," in *2019 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, 2019, pp. 905–910. DOI: 10.1109/ISGT-Asia.2019.8881348.

[21] S. Fan, X. Liu, Y. Chen, Z. Liao, Y. Zhao, H. Luo, and H. Fan, "How to construct a power knowledge graph with dispatching data?" *Scientific Programming*, vol. 2020, pp. 1–10, Jul. 2020. DOI: 10.1155/2020/8842463.

[22] Y. Yang, Z. Chen, J. Yan, Z. Xiong, J. Zhang, Y. Tu, and H. Yuan, "Multi-source heterogeneous information fusion of power assets based on knowledge graph," in *2019 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, 2019, pp. 213–218. DOI: 10.1109/SOLI48380.2019.8955005.

[23] Z. Su, M. Hao, Q. Zhang, B. Chai, and T. Zhao, "Automatic knowledge graph construction based on relational data of power terminal equipment," in *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, 2020, pp. 761–765. DOI: 10.1109/ICCCS49078.2020.9118512.

[24] B. Kan, W. Zhu, G. Liu, X. Chen, D. Shi, and W. Yu, "Topology modeling and analysis of a power grid network using a graph database," *International Journal of Computational Intelligence Systems*, vol. 10, p. 1355, Jan. 2017. DOI: 10.2991/ijcis.10.1.96.

[25] Y. Xu, T. Yu, and B. Yang, "Reliability assessment of distribution network through graph theory topology similarity and mathematical statistic analysis," *IET Generation, Transmission Distribution*, vol. 13, Oct. 2018. DOI: 10.1049/iet-gtd.2018.5520.

[26] B. Liu, *Sentiment Analysis and Opinion Mining*. Morgan Claypool Publishers, 2012, ISBN: 1608458849.

[27] Q. T. Ain, M. Ali, A. Riaz, A. Noureen, M. Kamran, B. Hayat, and A. Rehman, "Sentiment analysis using deep learning techniques: A review," *International Journal of Advanced Computer Science and Applications*, vol. 8, 2017.

[28] M. Pontiki, D. Galanis, H. Papageorgiou, I. Androutsopoulos, S. Manandhar, M. AL-Smadi, M. Al-Ayyoub, Y. Zhao, B. Qin, O. De Clercq, V. Hoste, M. Apidianaki, X. Tannier, N. Loukachevitch, E. Kotelnikov, N. Bel, S. M. Jiménez-Zafra, and G. Eryiğit, "SemEval-2016 task 5: Aspect based sentiment analysis," in *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 19–30. DOI: 10.18653/v1/S16-1002. [Online]. Available: https://www.aclweb.org/anthology/S16-1002.

[29] H. Xu, B. Liu, L. Shu, and P. S. Yu, "Double embeddings and CNN-based sequence labeling for aspect extraction," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 592–598. DOI: 10.18653/v1/P18-2094. [Online]. Available: https://www.aclweb.org/anthology/P18-2094.

[30] J. Yu, J. Jiang, and R. Xia, "Global inference for aspect and opinion terms co-extraction based on multi-task neural networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 1, pp. 168–177, 2019. DOI: 10.1109/TASLP.2018.2875170.

[31] Z. Fan, Z. Wu, X.-Y. Dai, S. Huang, and J. Chen, "Target-oriented opinion words extraction with target-fused neural sequence labeling," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 2509–2518. DOI: 10.18653/v1/N19-1259. [Online]. Available: https://www.aclweb.org/anthology/N19-1259.

[32] C. Ying, Z. Wu, X. Dai, S. Huang, and J. Chen, "Opinion transmission network for jointly improving aspect-oriented opinion words extraction and sentiment classification," in. Oct. 2020, pp. 629–640, ISBN: 978-3-030-60449-3. DOI: 10.1007/978-3-030-60450-9_50.

[33] C. Brun and V. Nikoulina, "Aspect based sentiment analysis into the wild," in *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 116–122. DOI: 10.18653/v1/W18-6217. [Online]. Available: https://www.aclweb.org/anthology/W18-6217.

[34] F. Herrera and L. Martinez, "A 2-tuple fuzzy linguistic representation model for computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 8, pp. 746–752, 2000.

[35] S. Dongen, "Graph clustering by flow simulation," *PhD thesis, Center for Math and Computer Science (CWI)*, May 2000.

[36] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *CoRR*, vol. abs/1308.0971, 2013. arXiv: 1308.0971. [Online]. Available: http://arxiv.org/abs/1308.0971.

Appendix A:
Power Grid Knowledge Graph

A.1 Graph Construction Process

A.1.1 Import from CSV files

The data describing details regarding connectivity and information about components of a power system, as well as customers is stored in CSV files. Therefore, the first step is importing all that data into a graph database. This involves creation of nodes that represent electrical components and relationships that represent electrical paths. The set of input CSV datafiles contains:

- a file with information about electrical components of the system;
- a file with basic data (all personal, sensitive information is removed) about customers being served by the power system.
- a file containing data with physical location of customers and which components they are connected to.

Below, we include a listing with procedures, written in Cypher – Neo4j graph query language, that read information from CSV files and create nodes and relations between them. The procedures use the vocabulary defined for representing power system information.

Listing A.1: Procedures for importing data from CSV files into Neo4j DB

```
// load connectivity data
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM 'file:///oms_connectivity.csv' AS network
WITH toInteger(network.node1) AS network_NODE1,
     toInteger(network.node2) AS network_NODE2,
     network
MERGE (n1:CNode {number:network_NODE1})
MERGE (n2:CNode {number:network_NODE2})
RETURN count(DISTINCT n1),count(DISTINCT n2),count(network);


:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM 'file:///oms_connectivity.csv' AS network
WITH
     toInteger(network.mslink) AS network_MSLIN,
     toFloat(network.length) AS network_LENGTH,
     network,
(CASE network.feature_id
WHEN '171' THEN 'Primary Isolation Point'
WHEN '254' THEN 'Fault Indicator'
WHEN '299' THEN 'Primary Substation Transformer'
WHEN '300' THEN 'Primary Breaker'
WHEN '301' THEN 'Primary Substation Bus'
WHEN '302' THEN 'Capacitor'
WHEN '304' THEN 'Primary Fuse'
WHEN '305' THEN 'Primary Generator'
WHEN '306' THEN 'Primary Conductor'
WHEN '307' THEN 'Primary Meter'
WHEN '308' THEN 'Recloser'
WHEN '309' THEN 'Regulator'
WHEN '310' THEN 'Sectionalizer'
WHEN '313' THEN 'Primary Switch'
WHEN '314' THEN 'Primary Transformer'
WHEN '322' THEN 'Elbow'
WHEN '327' THEN 'Primary Switch Gear'
ELSE 'OTHER' END) AS FEATURE

MERGE (e:Element {mslink:network_MSLINK})
     SET e.feature_id=FEATURE,
          e.length=network_LENGTH,
          e.no_phases=network.no_phases,
          e.normal_status=network.normal_status,
          e.orientation=network.orientation,
          e.phase=network.phase,
          e.scale=network.scale,
          e.service_area=network.service_area,
          e.connected_voltage=network.connected_voltage,
          e.system_voltage=network.system_voltage,
          e.normal_feeder_id=network.normal_feeder_id,
          e.x_coord=network.x_coord,
          e.y_coord=network.y_coord,
          e.name='Element'

:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM 'file:///oms_connectivity.csv' AS network
WITH network, toInteger(network.mslink) AS network_MSLINK,
     toInteger(network.node1) AS network_NODE1,
     toInteger(network.node2) AS network_NODE2
MATCH (n1:CNode {number:network_NODE1})
MATCH (n2:CNode {number:network_NODE2})
MATCH (e:Element {mslink:network_MSLINK})
MERGE (n1)-[c1:Connection{name:network.normal_status}]-(e)
```

```
MERGE (e)-[c2:Connection{name:network.normal_status}]-(n2)
RETURN count (c1), count(c2);


// import cispersl csv files to the database
CREATE CONSTRAINT ON (cp:CPremise) ASSERT cp.mslink IS UNIQUE;
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM 'file:///customer.csv' as line
WITH line.premise AS premise_mslink, line.xfmr AS xfmr, line
MERGE (cp: CPremise {mslink: premise_mslink})
SET cp += {critical_customer: line.critical_customer,
    account_type: line.account_type,
    customer_phases: line.customer_phases,
    xfmr: xfmr}

:auto USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS
FROM 'file:///customer.csv' as line
With
    line.mslink AS premise_mslink,
    toInteger(line.x_coord) AS x_coord,
    toInteger(line.y_coord) AS y_coord
MERGE (cp: CPremise {mslink: premise_mslink})
SET cp += { x_coord: x_coord,  y_coord: y_coord }



CREATE INDEX index_xfmr FOR (cp:CPremise) ON (cp.xfmr);
:auto USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS
FROM 'file:///customer_connect.csv' as line
MATCH (e:Element{mslink: toInteger(line.mslink)})
MATCH (cp: CPremise{xfmr:line.xfmr})
MERGE (e)-[c:Connection]-(cp)
```

A.2 Processing Graph Data

A.2.1 Converting Node Properties to Relationships

The code we use to process nodes of graphs is shown below. Its purpose is to convert node property information into nodes. As a result, we create an inter-operable ontology-based representation. The code performs a number of steps for each property:

- extracts values that the property takes;
- creates a single node for each property value;
- deletes the property from nodes, and connects these nodes to newly created nodes representing different values of the property.

Listing A.2: Java program for converting node properties into relationships

```
1    /**
2     * 1. create nodes for all property values;
3     *  CALL apoc.refactor.categorize(sourceKey, type, outgoing,
          label, targetKey, copiedKeys, batchSize)
4     * return each unique propertyKey into a category node and
          connect to it
5     */
6    private Result insertPropertyValueNode (Session session,
         String sourceProperty, String propertyRelName, String
         propertyLabel, String propertyNodePro, boolean outgoing,
         int batchSize) {
7        String sourceKey = sourceProperty;
8        String type = propertyRelName; //relationship name;
9        String label = propertyLabel ; // UpperCase + "_VALUE"; //
             node type
10       String targetKey = propertyNodePro; // node property
11       System.out.println("insertPropertyValueNode for property:
             " + sourceKey);
12       return session.run(
13               "CALL apoc.refactor.categorize($sourceKey, $type,
                    $outgoing, $label, $targetKey, $copiedKeys,
                    $batchSize)"
14               ,
15               parameters("sourceKey", sourceKey,
16                       "type", type, "outgoing", outgoing, "label
                            ", label,
17                       "targetKey", targetKey, "copiedKeys", new
                            ArrayList<>(),
18                       "batchSize", batchSize)
19       );
20
21   }
22   /**
23    * 1. Create Property Class Node (if not exists in the
          database)
24    * 2. Add the relationships between property value node and
          property-type class node
25    * @param tx
26    * @param clsLabel : the label for the PropertyClassNode,
27    * @param clsRel: the relationship to the clsNode
28    * @parm ingoing:
29    * @return
30    */
31   private Result insertPropertyClassNodeAndRel (final
         Transaction tx, String clsLabel, String proLabel, String
         clsRel, boolean ingoing) {
32       System.out.println("insertPropertyClassNodeAndRel for
             clsLabel: " + clsLabel + "...");
33       if (ingoing) {
34           return tx.run(  "Merge (clsNode:" + clsLabel + ") "  +
35                   "WITH clsNode " +
```

```
36                      "Match (valNode: " + proLabel + ") " +
37                      "MERGE (valNode)-[r: " + clsRel + "]-(clsNode)
                            "
38              );
39          }
40          return tx.run(  "Merge (clsNode:" + clsLabel + ") "  +
41                  "WITH clsNode " +
42                  "Match (valNode: " + proLabel + ") " +
43                  "MERGE (clsNode)-[r: " + clsRel + "]-(valNode)"
44          );
45
46      }
47      /**
48       *  View each categorizing properties to ontology
             relationships as a transcation work
49       *  consist of two steps :
50       *  1. Create a unique key constraint on the sourceProperty
51       *  2. create value nodes and relationships to the nodes which
              has the property.
52       *  3. creating a single node as the class node and  linking
             to  all value nodes.
53       * @param sourceProperty
54       * @param propertyRelName
55       * @param propertyLabel
56       * @param propertyNodePro
57       * @param outgoing
58       * @param batchSize
59       * @param clsLabel
60       * @param clsRel
61       * @param ingoing
62       * @param savedBookmarks
63       * @return
64       */
65      public List<Bookmark> convertPropertyToOntology (
66      final String sourceProperty, final String propertyRelName,
           final String propertyLabel,final String propertyNodePro,
           final boolean outgoing,final int batchSize,
67         final String clsLabel,final String clsRel,boolean ingoing,
              List<Bookmark> savedBookmarks) {
68
69          // To collect the session bookmarks
70
71          try ( Session session1 = driver.session( builder().
               withDefaultAccessMode(AccessMode.WRITE).withBookmarks(
               savedBookmarks).build()))
72          {
73              session1.writeTransaction(transaction -> {
74                  return this.createConstraint(transaction,
                       propertyLabel, propertyNodePro);
75              });
76              savedBookmarks.add(session1.lastBookmark());
77          }catch (ClientException e) {
78              System.out.println("index has already been created:
                   detailed message as follows:");
```

```
79              System.out.println(e.getMessage());
80          }
81
82
83          // apoc.refactor.categorize may occur deadlock because of
                simultaneously write transaction,
84          // retry 5 times if deadlock happens
85          try ( Session session2 = driver.session( builder().
                withDefaultAccessMode(AccessMode.WRITE)
86                  .withBookmarks(savedBookmarks).build())) {
87              this.insertPropertyValueNode(session2,sourceProperty,
                    propertyRelName,
88                      propertyLabel,propertyNodePro,outgoing,
                            batchSize);
89              savedBookmarks.add(session2.lastBookmark());
90          } catch (Exception e) {
91              System.out.println("Refactoring relationship have some
                    issues with deadlocks, retried some times , " +
92                      "but still having some issues: ");
93              System.out.println(e.getMessage());
94          }
95
96
97          try ( Session session3 = driver.session( builder().
                withDefaultAccessMode(AccessMode.WRITE).withBookmarks(
                savedBookmarks).build()))
98          {
99              session3.writeTransaction(transaction -> {
100                 return this.insertPropertyClassNodeAndRel(
                        transaction, clsLabel, propertyLabel, clsRel,
                        ingoing);
101             });
102             savedBookmarks.add(session3.lastBookmark());
103         }
104
105         return savedBookmarks;
106
107     }
```

A.2.2 Determining Number of Downstream Customers

The graph representation of components of the system allows us to easily compute a number of customers who are connected to the system downstream from a given node. This is achieved by performing two steps:

- finding the max level in Elements
- iterating from max level to zero, and computing a number of customer at the current level (node) based on already determined number at the next level.

Listing A.3: Code snippet for computing number of customers

```
1   private int queryMaxLevel(final Transaction transction) {
2       Result result = transction.run("MATCH (e:Element) " +
3       " RETURN {max_level: MAX(e.level)}");
4       while (result.hasNext()) {
5           Record record = result.next();
6           return record.values().get(0).get("max_level").asInt()
                ;
7       }
8       return -1;
9   }
10
11  private Result updateNumberOfCustomer(final Transaction
        transaction, int level) {
12      String queryTemplate = "MATCH (parent:Element)-[:
            Connection]->(n:CNode)-[:Connection]->(cur: Element) "
            +
13              "WHERE parent.level = $level AND cur.level =
                    $level + 1 " +
14              "WITH parent, SUM(cur.num_customer) AS
                    totalNumCustomer " +
15              "SET parent.num_customer = totalNumCustomer";
16
17      return transaction.run(queryTemplate, parameters("level",
            level));
18
19  }
20
21  public void preComputeNumberOfCustomer() {
22      List<Bookmark>  bookmarks = new ArrayList<>();
23      try (Session session = driver.session( builder().
24              withDefaultAccessMode(AccessMode.READ)
25              .build())) {
26          this.maxLevel = session.readTransaction(transaction ->
                {
27              return this.queryMaxLevel(transaction);
28          });
29          bookmarks.add(session.lastBookmark());
30      }
31
32      if (this.maxLevel == -1) {
33          System.out.println("The max level is not retrieved
                correctly");
34          return;
35      }
36
37      for (int level = this.maxLevel -1; level >= 0; level--) {
38          try (Session session = driver.session(builder()
39                  .withDefaultAccessMode(AccessMode.WRITE)
40                  .withBookmarks(bookmarks)
```

```
                    .build())) {
                final int currentLevel = level;
                System.out.print("computing number of customers
                    for level :" + currentLevel);
                session.writeTransaction(transaction -> {
                    return this.updateNumberOfCustomer(transaction
                        , currentLevel);
                });
                bookmarks.add(session.lastBookmark());
            }
        }
        return;
    }
```

A.2.3 Multi-Source Path Search

A detailed description of Multi-Source Path Search algorithm is included in Section 3.3. Below, we list a Java program implementing the algorithm.

Listing A.4: Code snippet for search algorithm

```
private List<Element> queryRootElement(final Transaction tx ,
    long parentId) {
        Result result = tx.run(
        "MATCH (e: Element {feature_id: $featureType})-[c:
            Connection]-> (n:CNode {number: $parentId}) " +
                "RETURN {mslink: e.mslink, feature_id: e.
                    feature_id} UNION " +
                "MATCH (e: Element {feature_id: $featureType})<-[c
                    :Connection]-(n:CNode{number: $parentId}) " +
                "RETURN {mslink: e.mslink, feature_id: e.
                    feature_id};", parameters( "featureType",
                    START_STR, "parentId",parentId)
        );
        while (result.hasNext()) {
            Record record = result.next();
            Element breaker = Element.builder()
                    .mslink(record.values().get(0).get("mslink").
                        asInt())
                    .featureId(record.values().get(0).get("
                        feature_id").asString())
                    .build();
            breakers.add(breaker);
        }
        return breakers;
    }

    private Node queryNextNode(final Transaction tx , int mslink,
        long parentId) {
        Result result = tx.run(
```

```
21                  "MATCH (e: Element) <-[c:Connection]- (n:CNode)" +
22                  "WHERE e.mslink = $mslink AND  n.number <>
                        $parentId " +
23                  "RETURN {number: n.number} " +
24                  "UNION " +
25                  "MATCH (e: Element) -[c:Connection]-> (n:CNode) "
                        +
26                  "WHERE e.mslink = $mslink AND  n.number <>
                        $parentId " +
27                  "RETURN {number: n.number};",
28                  parameters( "mslink", mslink, "parentId",parentId)
29          );
30          while(result.hasNext()) {
31              Record record = result.next();
32              int nodeNum = record.values().get(0).get("number").
                    asInt();
33              return Node.builder()
34                      .number((long) nodeNum)
35                      .build();
36          }
37          return null;
38      }
39
40
41
42      private List<Element> queryNextElements(final Transaction tx ,
            int mslink, long parentId) {
43          List<Element>  resultList = new ArrayList<>();
44          Result result = tx.run(
45                          "MATCH (e: Element) <-[c:Connection]- (n:
                                CNode) " +
46                            "WHERE e.mslink <> $mslink AND n.number =
                                $parentId " +
47                            "RETURN {mslink: e.mslink, feature_id: e.
                                feature_id} " +
48                            "UNION " +
49                            "MATCH (e: Element) -[c:Connection]-> (n:
                                CNode)" +
50                            "WHERE e.mslink <> $mslink AND  n.number =
                                 $parentId " +
51                            "RETURN {mslink: e.mslink, feature_id: e.
                                feature_id};",
52                  parameters( "mslink", mslink, "parentId",parentId)
53          );
54          while(result.hasNext()) {
55              Record record = result.next();
56              resultList.add(Element.builder()
57                      .mslink(record.values().get(0).get("mslink").
                            asInt())
58                      .featureId(record.values().get(0).get("
                            feature_id").asString())
59                      .build());
60          }
61
```

```
62          return resultList;
63      }
64      private Result setLevelProperty(final Transaction tx, int
            mslink, int level) {
65          return tx.run("MERGE (e: Element {mslink: $mslink}) " +
66                  "SET e += {level: $level} ", parameters("mslink",
                        mslink, "level", level)
67          );
68      }
69      /**
70       * Reverse the connection to let it consistent with the energy
               flow
71       * @param tx
72       * @param preMslink
73       * @param number
74       * @param mslink
75       * @return
76       */
77      private Result modifyNodeRelationship(final Transaction tx,
            int preMslink, long number, int mslink) {
78
79          String queryTemplate = "MATCH (e:Element {mslink:
                $premslink}) <-[c:Connection]-(n:CNode{number:$number})
                 " +
80                  "WITH id(c) as id MATCH ()-[r]->() WHERE id(r) =
                        id "  +
81                  "CALL apoc.refactor.invert(r) yield input, output
                        RETURN * " +
82                  "UNION " +
83                  "MATCH (e:Element {mslink: $mslink}) -[c:
                        Connection]-> (n:CNode{number:$number}) " +
84                  "WITH id(c) as id MATCH ()-[r]->() WHERE id(r) =
                        id " +
85                  "CALL apoc.refactor.invert(r) yield input, output
                        RETURN * ";
86          Result result = tx.run(queryTemplate, parameters("
                premslink", preMslink, "number", number, "mslink",
                mslink));
87
88          return result;
89      }
90
91      /**
92       * [optional] set the parentNode and childNode information
93       * @param tx
94       * @param mslink
95       * @param parentNum
96       * @param childNum
97       * @return
98       */
99      private Result createDirectionProperty(final Transaction tx,
            int mslink,
100                                            long parentNum, long
                                                childNum) {
```

```
101             return tx.run(
102                 "MERGE (e:Element {mslink: $mslink})" +
103                   "SET e += {parentNum: $parentNum, childNum:
                        $childNum}",
104                 parameters("mslink", mslink, "parentNum", parentNum, "
                        childNum", childNum)
105             );
106         }
107 
108         public void buildHierarchical() {
109             buildHierarchicalIteratively(this.breakers, Node.builder()
                    .number(0L).build());
110         }
111 
112         /**
113          * build the power stream by setting the level property for
                each element,
114          * 1. set the property of the current Element
115          * 2. query the neighbourhood node.which is the childNode
116          * 3. find childElement and call the func recursively if it
                satisfiy filter requirement.
117          * @param rootElements
118          * @param rootNode
119          */
120         private void buildHierarchicalIteratively(final List<Element>
                rootElements, final Node rootNode) {
121             List<Element> nextElements = null;
122             Set<Integer> visited = new HashSet<>();
123             Deque<Holder> queue = new LinkedList<>();
124             ElementFilter switchFilter = SwitchFilter.getSwitchFilter
                    ();
125             for(Element breaker: rootElements) {
126                 queue.add(new Holder(breaker, rootNode));
127             }
128             int level = 0;
129             while (!queue.isEmpty()) {
130                 int size = queue.size();
131                 // iterate all children
132                 for (int idx = 0 ; idx < size; idx ++){
133                     final Holder holder = queue.poll();
134                     final Element curElement = holder.getElement();
135                     final Node parentNode = holder.getNode();
136                     final int curLevel = level;
137                     if (visited.contains(curElement.getMslink())) {
138                         continue;
139                     }
140                     try ( Session session = driver.session( builder().
                            withDefaultAccessMode(AccessMode.WRITE).build()
                            ))
141                     {
142                         session.writeTransaction(transaction -> {
143                             System.out.println("mslink: " + curElement
                                    .getMslink() + " set level value: " +
                                    curLevel);
```

```
144                         return this.setLevelProperty(transaction,
145                                 curElement.getMslink(), curLevel);
146                     });
147                     visited.add(curElement.getMslink());
148                     final Node childNode = session.readTransaction
                            (transaction -> {
149                         return this.queryNextNode(transaction,
                                curElement.getMslink(),parentNode.
                                getNumber());
150                     });
151                     if (childNode != null) {
152                         nextElements = session.readTransaction(
                                transaction -> {
153                             return this.queryNextElements(
                                    transaction, curElement.getMslink()
                                    ,childNode.getNumber());
154                         });
155                     }
156                     if (childNode != null && nextElements != null
                            && nextElements.size() > 0) {
157                         List<Element> elements = switchFilter.
                                doFilter(session, nextElements);
158                         elements.forEach(element -> {
159                             if (element.getMslink() != null) {
160                                 session.writeTransaction(
                                        transaction -> {
161                                     return this.
                                            modifyNodeRelationship(
                                            transaction,
162                                                 curElement.getMslink()
                                                    , childNode.
                                                    getNumber(),
                                                    element.getMslink()
                                                    );
163                                 });
164                                 queue.add(new Holder(element,
                                        childNode));
165                             }
166                         });
167                     }
168                 }
169             }
170             level += 1;
171         }
172     }
```

Appendix B:
Sentiment Knowledge Graph

B.1 Construction of Review Model (REmModel)

The graph-based Review EmotionModel (REmModel) contains information that has been obtained as a results of aspect-based sentiment analysis process. The file *result.csv* contain the results. The information is organized in the form of columns:

- *reviewer*: id of the reviewer;
- *text*: sentence of the review;
- *category*: aspect term category;
- *target*: aspect term;
- *polarity*: sentiment polarity (positive/negative);
- *description list*: list of opinion words described the aspect term.

The Cypher code implementing the construction process.

Listing B.1: Code snippet for REmModel

```
//1. import reviewer nodes with Reviewer Label and  id(NOT NULL) as its property.

LOAD CSV WITH HEADERS FROM "result.csv"
AS row
WITH row
WHERE row.review_id IS NOT NULL
MERGE (reviewer:Reviewer {reviewId: row.review_id})



// 2. import target with targetValue as properties,
//  change the default value to 'UNKNOWN_TARGET'
// only one node has property UNKNOWN_TARGET.
// TEST VIA COMMAND :
// MATCH (n:Target)
// WHERE n.targetValue = "UNKNOWN_TARGET" return COUNT(n)
```

```
// OUTPUT: n = 1
LOAD CSV WITH HEADERS FROM "result.csv"
AS row
MERGE (target:Target{targetValue: CASE trim(row.target)
WHEN 'NULL' THEN 'UNKNOWN_TARGET' ELSE trim(row.target) END})


// 3. import polarity from the file.
// polarity only has three type values.(no missing values)
// Test:
// MATCH (n:Polarity) RETURN n LIMIT 25
//
//     "n"
//
//     {"polarityValue":"negative"}
//
//     {"polarityValue":"positive"}
//
//     {"polarityValue":"neutral"}
//

LOAD CSV WITH HEADERS FROM "result.csv"
AS row
MERGE (polarity:Polarity{polarityValue: CASE trim(row.polarity)
WHEN 'NULL' THEN 'UNKNOWN_POLARITY'
ELSE trim(row.polarity) END})




// 4. import sentence with  text as its property,
// and find the reviewer who writes the sentence,
// add relationships to that setences.
// Example  find all the sentences written by reviewer 1004293

// MATCH  (reviewer:Reviewer {reviewId: "1004293"})
// <-[:IS_REVIEWED_BY]- (sentence:Sentence)
// RETURN sentence.text

USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "result.csv"
AS row
MERGE (sentence:Sentence{text: trim(row.text)})


USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "result.csv"
AS row
MATCH (reviewer:Reviewer {reviewId: row.review_id})
MATCH (sentence:Sentence {text: trim(row.text)})
MERGE (sentence)-[r:IS_REVIEWED_BY]->(reviewer)



// 5. import Entity and Attributes simutanouesly.
// Test:  MATCH (n:Entity) RETURN COUNT(n)  | Expected:  6
// Test:  MATCH (n:Attribute) RETURN COUNT(n)  | Expected:  5

LOAD CSV WITH HEADERS FROM "result.csv"
AS row
WITH SPLIT(row.category, '#')  AS categoryList
MERGE (entity:Entity {entityVal: trim(categoryList[0])})
MERGE (attribute:Attribute {attributeVal: trim(categoryList[1])})



// 6. Create Relationships for category.
```

```
LOAD CSV WITH HEADERS FROM "result.csv"
AS row
WITH SPLIT(row.category, '#') AS categoryList
MATCH (entity:Entity {entityVal: trim(categoryList[0])})
MATCH (attribute:Attribute {attributeVal: trim(categoryList[1])})
MERGE (category:Category{categoryVal: categoryList})
MERGE (category)-[:HAS_ATTRIBUTE]->(attribute)
MERGE (category)-[:HAS_ENTITY]->(entity)



// 7. manipulate the description lists, and create all descriptions nodes
// Make Sure to have apoc installed.

LOAD CSV WITH HEADERS FROM "result.csv"
AS row
WITH row, apoc.convert.fromJsonList(row.description_list) AS descriptionList
UNWIND descriptionList AS descriptionWord
MERGE (description:Description{description: trim(descriptionWord)})


// 8. Create Entry nodes with properties ID: "categoryId#SentenceId#TargetId"
// An Entry Node is unique determinated by sentence, target and category.
LOAD CSV WITH HEADERS FROM "result.csv"
AS row
WITH row, SPLIT(row.category, '#') AS categoryList
MATCH (reviewer:Reviewer {reviewId: row.review_id})
MATCH (entity:Entity {entityVal: trim(categoryList[0])})
MATCH (attribute:Attribute {attributeVal: trim(categoryList[1])})
MATCH (category:Category)-[:HAS_ATTRIBUTE]->(attribute)
MATCH (category:Category)-[:HAS_ENTITY]->(entity)
MATCH (target:Target{targetValue:
CASE trim(row.target)
WHEN 'NULL' THEN 'UNKNOWN_TARGET'
ELSE trim(row.target) END})
MATCH (sentence:Sentence)-[:IS_REVIEWED_BY]->(reviewer)
MERGE (entry:Entry{entryId: apoc.text.join([toString(id(sentence)),
toString(id(target)),toString(id(category))], '#')})
RETURN entry.entryId, sentence.text, target.targetValue,
id(category), reviewer.reviewId,
entity.entityVal,attribute.attributeVal
```

B.2 SenticNet RDF Format

The original XML file that contains SenticNet model has been processed according to the following procedure:

- based on the DOM4j library the XML file is parsed into org.dom4j.Elements;
- distinct objects bypassing the elements are created, and then combined in ordeer to create RDF triples;
- RDF objects triples are converted back to dom4j.Elements, and an output RDF file is created.

The interface that describes the steps, as well as an example of RDF description are shown below.

Listing B.2: Interface for converting XML to RDF format

```java
public interface InterfaceHandler<T> {

    /**
     * Process the whole documents for particular nodes
     * @param document
     * @return
     */
    Document processAll(Document document);

    /**
     * Convert org.dom4j.Element to particular pojo object
        instance
     * @param element
     * @return
     */
    T convertElement2Object(Element element);


    /**
     * Create triples based on the mapped pojo object
     * @param object
     */
    void createTriples( T object);


    /**
     * Modify the representation of the original triple
     * @param element
     * @param object
     */
    void modifyRepresentation(Element element, T object);

}
```

Listing B.3: A Valid RDF description

```
<rdf:Description rdf:about="http://sentic.net/api/en/concept/a_little">
  <rdf:type rdf:resource="http://sentic.net/api/concept"/>
  <text xmlns="http://sentic.net/">a little</text>
  <concept:concept-relationship
      rdf:resource="http://sentic.net/api/en/concept/least"/>
  <concept:concept-relationship
      rdf:resource="http://sentic.net/api/en/concept/little"/>
  <concept:concept-relationship
      rdf:resource="http://sentic.net/api/en/concept/small_amount"/>
  <concept:concept-relationship
      rdf:resource="http://sentic.net/api/en/concept/shortage"/>
  <concept:concept-relationship
      rdf:resource="http://sentic.net/api/en/concept/scarce"/>
```

```
    <concept:concept-relationship
        rdf:resource="http://sentic.net/api/en/concept/sadness"/>
    <concept:concept-relationship
        rdf:resource="http://sentic.net/api/en/concept/disgust"/>
    <sentics:pleasantness rdf:resource="http://sentic.net/pleasantness/-0.99"/>
    <sentics:attention rdf:resource="http://sentic.net/attention/0.0"/>
    <sentics:sensitivity rdf:resource="http://sentic.net/sensitivity/0.0"/>
    <sentics:aptitude rdf:resource="http://sentic.net/aptitude/-0.7"/>
    <polarity:intensity rdf:resource="http://sentic.net/intensity/-0.84"/>
</rdf:Description>
```