A Study of Text Similarity Methods

School of Computer Science and
Informatics
Cardiff University, 2022

CMT403 – MSc Computing
Dissertation

Author: Peishan Li
C21104607
Supervisor: Alia I Abdelmoty

Abstract:

The aim of this project is using the Natural Language Processing Tool (NLTK) to newly implement the path-based and information content based similarity packed in WordNet::Similarity Perl Module into the pre-existing system to perform semantic similarity analysis and comparison to find features and differences between the graph WordNet in Neo4j and Wordnet in NLTK and applicability between similarity methods in finding similarities between words at different levels. The Flask front-end is used to interact with graph data and NLTK data. The similarity was extended by using the NLTK but not with Neo4j even though the knowledge graph can support a wider variety of similarity methods but those methods are not suitable with WordNet.

Acknowledgements:

I would like to thank my supervisor Alia I Abdelmoty for her time and continued support and throughout the project.

I would also like to thank my friends and my family for their supporting and encouragement throughout the duration of this project.

Table of contents

List of Figures

1. Introduction

Semantic similarity has been a popular topic of research in the fields of data processing, linguistics and artificial intelligence. By measuring semantic similarity between two concepts in a computational ontology, it can be used for word elimination, system advancement, semantic annotation, bioinformatics, software domains and many more. It is therefore essential to improve the accuracy of similarity measures by measuring and comparing various aspects of similarity methods. The conventional way to indicate the similarity between two words senses is literature similarity, but with WordNet as a large lexical database that linked by semantic relations, we can now perform semantic similarity with WordNet. Therefore, the implementation of different similarity methods with WordNet as the graph database in Neo4j and WordNet in NLTK are performed together to find out the meaning behind semantic similarity. Path-based and information content-based measure, are worth exploring to focus on comparing the accuracy of different similarities and the differences in similarity values arising under different forms of wordnet. This paper therefore presents a systematic comparison of the two measures, path-based and information content-based, as implemented in the WordNet graph database and NLTK data respectively, through an intuitive and simple method, to show the strengths and weaknesses of different similarities in different data formats and different measurement methods, and to provide a direction for the selection of similarity methods in applications.

The remainder of the paper is as follows: in Section 2, the relevant concepts of WordNet, NLTK, Neo4j and the formulas and concepts of similarity measurement are introduced. In Section 3 the approach and design for implementing the NLTK similarity measure are presented, and in Section 4 the process of reducing the wordnet modelled into the neo4j graph database and performing the similarity algorithm for the system is documented in detail, respectively, as well as the process of the newly implemented NLTK wordnet similarity. In section 5 the similarity calculations are carried out for the wordnet of Neo4j and the wordnet of NLTK using three data sets with different similarities, and the relationships between the different algorithms and the different databases are analysed and compared. Section 6 and section 7 provide what improvements can be made in the future and conclusion of this project.

Aim and Objectives:

This project aims to determine semantic similarity between pairs of word senses with WordNet Similarity package using NLTK library in Python3.8 on top of an exist similarity query system that implemented by an undergraduate student Joe Hayes(2022). The existing system calculates the similarity with wordnet modeled into neo4j as graph database while the newly implemented similarity will use nltk wordnet data. With different data resource we are able to compare the advantages and disadvantages of similarity performances between wordnet with graph database and wordnet with nltk data.

Main objectives:

1.Study the existing database and system for storing and modelling Wordnet. a. study Neo4j b. study Wordnet structure, c. rebuild , install and run the existing system.
2.Investigate and identify similarity evaluation methods that can be applied to understand the content of Wordnet.
3.Evaluate and extend the similarity techniques that is developed in the existing software.
4.Build a user interface to query the database and extract the computed similarities.
5.Evaluate the developed system and interface with an appropriate set of questions (guided by the initial requirements).

Figure 1 Main Objectives

2. Background

This section is about related work that has already been investigated in other research papers, which also covers the area of Wordnet similarity measures. Additionally, it covers the essential background knowledge of NLTK library as well as Neo4j to fully understand the methodology and implementation that is undertaken in this report.

2.1 WordNet

What is Wordnet

WordNet is a semantically oriented large database of English words. Words are grouped into synsets according to their meaning under different parts of speech, including nouns, verbs, adverbs and adjectives. Each synset contains a word sense with a unique meaning, so the basis of how to group the words is depend on their meanings. The resulting network of meaningfully related words and concepts can be navigated with the browser, and words that are found in close proximity to each other in the network are semantically disambiguated, while the semantic similarity is shown by how close a sense or synset is.

Structure

The main relations between words in Wordnet is synonymy, which is a set of meanings that share the same concept and interchangeable in many cases. Synsets are connected to each other by conceptual-semantic relationships. Each synset has a short definition ('gloss'), which is mainly a few short sentences describing the use of each sense in the synset. Some words may have several different meanings, and these meanings are featured in each of the different senses.

Relations

The most common relation between synonyms in wordnet is the IS-A relations, also known as hyponymy and hyponymy. Wordnet classifies verb and noun as separate hierarchies. The root node in the hierarchy's terminal, which links increasingly specific synonyms such as "money" and "cash" and "coin" are associated with each other. Thus, in wordnet, the increasingly specific hyponymy relation from the root node downwards is that "exchange" includes "money" and " credit", and "money" includes "cash". In reverse, the increasingly

abstract hypernymy relationship from bottom to top indicates that "money" and "credit" form the category "exchange". In this IS-A taxonomy, hyponym relations allow for multiple inheritance: if “nickel” is a type of “coin”, and "coin" is a type of "cash", then “nickel” is a type of “cash”. WordNet classifies not only common nouns but also instances, including specific persons, countries and geographical entities. For example, King Charles III is an instance of King, and instances are always root nodes in its hierarchy.

In addition to Nouns, Verb phrases are also represented in hierarchies. In the tree of troponyms, increasingly specific forms of activity are expressed from top to bottom, and specific meanings are expressed depending on the semantic field in which the different verbs are located. However, verb meanings are difficult to organize in one hierarchical structure and many verbs has meanings from different conceptual domains. For example, in the domain of expression, 'communicate' - 'talk' - 'whisper' expresses increasingly specific ways of speaking, and in the domain of speed, 'move' - 'run' - "jog" expresses a gradual increase in speed, and in the field of emotion, "like"-"love"-"idolize" expresses an increasing degree of affection. At the same time there is another kind of verb which is interlinked in a necessary relationship, such as 'buy' and 'pay', 'proceed' and 'try', 'show' and 'see', etc.

In addition to the is-a hierarchy network of relations inside WordNet, there is also the non-hierarchy network of relations. Meronymy relations that representing the part-whole between noun synsets, antonymy, semantically similar and pertainyms relations that connect adjectives, and last the adverbs, are all grouped in the non-hierarchy taxonomy. Moreover, each word sense is given a short-written description or gloss.

In nouns in which parts and wholes are related, parts are inherited from their whole, but whole are not inherited from their parts. If a "chair" has different parts of "leg" and "seat", then "armchair" has " If a "chair" has different parts of "leg" and "seat", then "armchair" has " armchair" has "leg" and "seat". But if an "armchair" has "arm" does not mean "chair" and every but if an "armchair" has "arm" does not mean "chair" and every "furniture" has "arm".

There are three kinds of relationships contained in the adjective. The first is direct antonyms such as wet-dry and tall-short, which reflect strong semantic connection to each other. These semantically similar adjectives are the 'indirect antonyms' of the opposite adjective. The third

type of adjective is 'pertainyms', which refer to adjectives that are morphed from nouns, such as 'crime' - 'criminal'.

Adverbs in English have synonym and antonymy relations since they are morphed from adjectives such as surprisingly, the wordnet non-hierarchy contains only a few adverbs, such as hardly, mostly, really, etc.

Although most of the concepts are grouped under the same part of speech, there are still some cross part of speech relationships that exist.

For each part of speech, “a” represents “Adjectives”, “s” represents “Adjective Satellite”, “r” represents “Adverb”, “v” represents “Verb.”

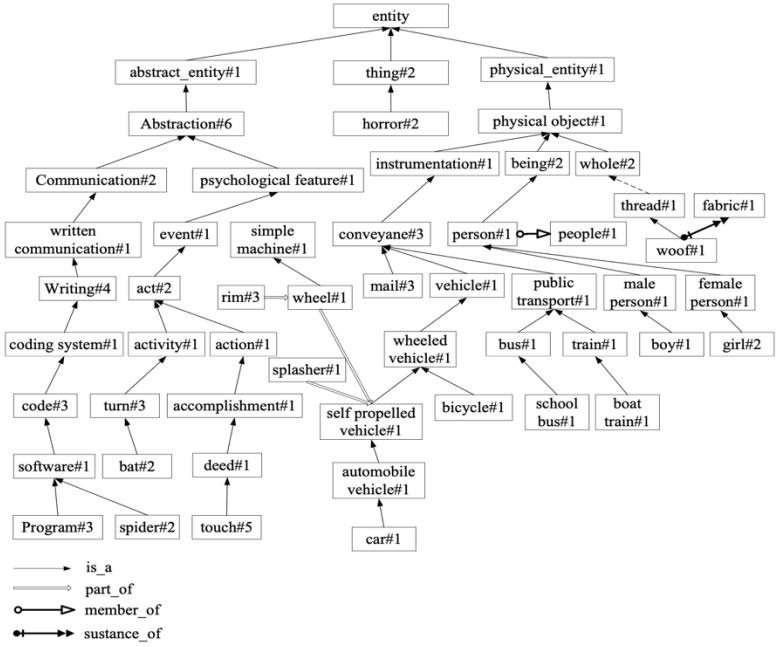


Figure 2 Fragment of WordNet concept in taxonomy（Meng et al.2014）

Wordnet as database

WordNet is generally provided as a WordNet RDF resource, which can be exported in a variety of formats such as HTML+RDFa, RDF/XML, Turtle and JSON. the RDF resource follows the OntoLex-Lemon model, which is a way of representing lexical data in a knowledge graph.

Semantic Similarity

WordNet::Similarity is a software package that consists of Perl modules that developed by Ted Pederson(2004) to measure similarity between senses based on the WordNet lexical database. It access the WordNet database with help of the WordNet::QueryData package which is a direct Perl interface to WordNet database developed by (Rennie 2000).The package contained Perl modules created as object classes with different methods that take two word senses as input. Semantic similarity measures are proposed by Leacock Chodorow (1998), Wu Palmer (1993), Resnik (1995), Lin (1998), Jiang Conrath (1997),path and three semantic relatedness introduced by Hirst St-Onge (1998) and the Extended Gloss Overlaps measure by Banerjee and Pedersen (2003) and a Gloss Vector measure produced by Pedersen and Lord (2003)are implemented in the package. With two given words, the command line interface to these modules present in the package can simply return the similarity measures. Measures described by Resnik, Lin, Jiang Conrath are based on information content files. Therefor the package also has support programs provided for generating information content files from various corpora. Tool to find depths of the taxonomies in Wordnet is also provided. Despite measures above that are already incorporated in the package, any other existing measures can be included into a Perl module by create an object using "new()"method. Then the similarity or relatedness value of a pair of word senses called from WordNet can be returned by using "getRelatedness()" method, even though only the similarity are implemented in this project.

Path Similarity

The path measure based on path lengths by counting the number of edges between two words or concepts in the is-a hierarchy by returning a floating value to represent how similar two words senses are. It is equal to the inverse of the shortest path length between two concepts. Once you have two given word getting from WordNet and the number of edges to represent the distance, a path similarity score will be returned with this equation 1.0 / (distance + 1). The score range are normally between 0 and 1 while 1 being the maximum value. It is a basic measure that delivers the shortest distance idea for other Path based measures to allow multiple extensions.

Lch Similarity

Leacock & Chodorow (1998) extend the path-based similarity as it will incorporate the depth of the queried taxonomy. It depends on the shortest path between two concepts and requires these two words in the same part of speech, and the maximum path length in the is–a hierarchy in which they occur. By comparing which length from the two given concepts to the root in the hypernym tree, the longest path is the Depth in the equation below. Two factors and one logarithmic equation are also involved with it to get the final results which range normally greater than 0 and return nothing when no path found to avoid log(0) .

$$LCH\ similarity = -\log \frac{shortestPath(Synset\ 1, Synset\ 2)}{2 * Depth}$$

Wup Similarity

The Wu & Palmer (1994) similarity measure is originally proposed to allow correct lexical choice of verb meanings to be performed based on different conceptual domains, even with the inexact matches of verbs between source language to the target language. As verbs are organized in multiple different conceptual domains, not in one hierarchical structure as nouns do, this limits two concepts need to be within same conceptual domain of hierarchical structure when using this measure to define how similar they are. It based on path lengths between concepts and finds the path length to the root node from the least common subsumer (LCS) of the two concepts in the hypernym-hyponym tree, which is the most specific concept they share as an ancestor. This value is scaled by the sum of the path lengths from the individual concepts to the root and the length form lcs to the root using equation below. C1 and C2 stands for concepts being compared, C3 is the least common subsume, N3 means the depth of lcs to the root, N1 and N2 are lengths form C1and C2 to C3.

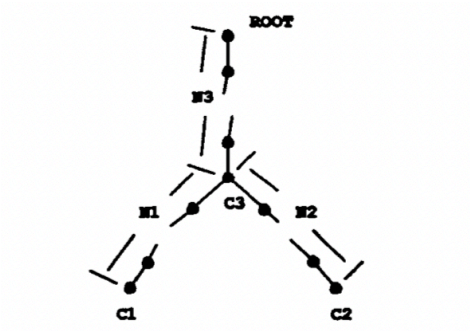


Figure 3 The concept of wup similarity measures

$$ConSim(C1, C2) = \frac{2*N3}{N1+N2+2*N3}$$

$$sim_{W\&P}(c_1, c_2) = \frac{2 * depth(lso(c_1, c_2))}{len(c_1, c_2) + 2 * depth(lso((c_1, c_2))}$$

However, Guessoum (2016) mentions that even though the Wu and Palmer semantic similarity metric has simplicity and high performance, the inherent drawback is obvious. Two concepts belonging to different hierarchies can deliver higher similarity level than two words belonging to same hierarchy, this inaccuracy shown is the weakness of wup similarity.

Resnik Similarity

Resnik (1995) proposed a measure to find the semantic similarity between any two concepts with their concepts defined in a hierarchical structure within ontology with information content based on WordNet. A large number of English nouns from WordNet and noun frequencies from the American English Brown Corpus are used in the construction process.

Probability of occurrence of concept c is computed by its frequency of instance occurring in the taxonomy as follow:

$$p(c) = \frac{freq(c)}{N}$$

With the probability of a concept ready, Resnik calculate the information content use the negative logarithmic of it.

$$IC = -\log p(c)$$

The final equation is as below, c1 and c2 stands for two concepts of given words and LCS stands for the Least Common Subsumer which is also called as Closest Common Ancestor.

$$sim_{\text{Re}snik}(c_1, c_2) = -\log p(lso(c_1, c_2)) = IC(lso(c_1, c_2)$$

This method depends on only the most specific common ancestor of two concepts in the taxonomical relations which results in score will be 0 if the LCS of two words is root node. However, it allows for disambiguation of the words to be compared through the intersection of shared information between words. For example, the word 'doctor' has the meaning of both doctor or Doctor of medicine, and the word 'nurse' has the meaning of both "The word 'doctor' means both 'health professional' and 'childcare worker'. When the words 'Doctor' and 'Nurse' are taken together and compared for similarity, the intersection of the information they share indicates that they both belong to the hospital workforce. The words 'Doctor', meaning a doctor, and 'Nurse', meaning a person who takes care of children, do not share a single piece of information, and even though their shared information could be 'job title ", but this node is much less specific than "hospital worker" in the overall is-a taxonomy. Therefore when "Doctor" and "Nurse" appear at the same time, their similarity is automatically calculated based on "Medical Doctor" and "Health Specialist". The similarity result is indeed more accurate when the ambiguity is removed. When the results of this information-based similarity measure are compared with human similarity judgements, and also with the results of the traditional edge-counting path-based similarity, the results show that this information-based similarity algorithm is very close to human judgements and outperforms edge-counting similarity.

Lin Similarity

Lin (1998) builds on Resnik similarity by considering both the common information content between two words senses and also the respective information content that fully described each word. He concludes that the similarity between two words sense is the ratio of their common information content to the sum of individual information content of the two words. The returning score range is 0 to 1, will return 1 when a concept compared with itself.

$$Sim_{LIN}(c1,c2)=2*\{IC(LCS(c1,c2))\}/IC(c1)+IC(c2)$$

Jcn Similarity

Jiang & Conrath (1997) similarity is a variation of above information content theory, which computes the length in the taxonomy relation as the difference between the two compared senses of the word and their subsumer. In other words, it quantifies the semantic distance to gain similarity results. It is reflected in the calculation by subtracting the IC of the lcs from the sum of the ICs of the two concepts to obtain the similarity result. The similarity score returned by the inversed of the distance of information content between two concepts. The specific formula is as follows

$$\text{IC}distance_{JCN}(c1, c2) = (IC(c1) + IC(c2) - 2*\{ IC(LCS(c1,c2))\}$$

$$Sim_{JCN}(c1, c2) = 1/ (IC(c1) + IC(c2) - 2*\{ IC(LCS(c1,c2))\}$$

Related Work on Comparison of Different Measures

Hliaoutakis (2006) has used WordNet and MeSH to perform a semantic similarity comparison of the three methods Resnik, Jcn, Lin. In his experiments, he compared the similarity estimates of given nouns pairs by undergrads and medical experts with the three information content similarities and obtained that the Lin and Jcn algorithms show higher correlation results than Resnik, which shows that Lin and Jcn take more information content of their respective concepts into account than Resnik. This shows that the information content of the respective concepts considered by Lin and Jcn compared to Resnik's algorithm is able to improve the similarity accuracy. Resnik is limit on ontologies and inadequate consideration of context (Guessoum 2016).

2.2 **Nature Language Tool-Kit (NLTK)**

NLTK is a python library that provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet that allow you to work with human natural language data. NLTK provides a suite of text processing libraries for classification, tokenisation, word morphology reduction, word stemming, parsing, POS annotation and more. The library has tools for almost all NLP tasks. Most of the files in the NLTK corpus follow the same rules while a text corpus is a large body of text. These files are mainly plain text files, some of them are XML files and others are in other formats, but all can be accessed manually or through NLTK modules and Python.

NLTK Usage for WordNet

As mentioned above, WordNet is just another NLTK corpus reader and can be imported by code "from nltk.corpus import wordnet". NLTK have the interface to semantic similarity methods that provided in WordNet::Similarity package. Similarity methods in the Perl Wordnet module are implemented in the NLTK "nltk.corpus.reader.wordnet" module.
The method "lowest_common_hypernyms" stored in the source code of wordnet corpus reader module is an implementation of the "lowest Common Subsumer" method provided in the Perl WordNet module. This method get a list of lowest synsets in the hypernym tree. By setting a flag called "use_min_depth" to true or false to return the lowest minimum depth or lowest maximum depth. Maximum depth is the length of the longest hypernym path from a synset to the root while minimum depth is the shortest hypernym path to the root.

As various verb taxonomies do not share a single root like noun taxonomic links, and path-based similarity cannot function for synsets that are not connected. Simulate root is a flag that set to true by default to create a fake root that connects all the taxonomies to allow similarity working. Without this fake root node, the Wu-Palmer similarity implemented in wordnet corpus reader module cannot always compute a value that comply with those implementation given by Pederson's Perl of WordNet Similarity, especially for verbs.

WordNet and NLTK modules can be used together to find word meanings, synonyms, antonyms and semantic similarity, etc. Sample usage are as follow steps.

Import wordnet

```
>>>from nltk.corpus import wordnet as wn
```

Figure 4 import wordnet corpus

Look up all the Synsets of a word ”dog”

```
>>>wn.synsets('dog')
[Synset('dog.n.01'), Synset('frump.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'),
Synset('andiron.n.01'), Synset('chase.v.01')]
```

Figure 5 look up synsets in nltk

Query the definition of a synonym set using word ”apple”

```
>>> wn.synset('apple.n.01').definition()
'fruit with red or yellow or green skin and sweet to tart crisp whitish flesh'
```

Figure 6 query definition of a synonym in nltk

Query examples of word ‘dog’ for one of the meaning inside a synset

```
>>> wn.synset('dog.n.01').examples()
['the dog barked all night']
```

Figure 7 query examples of a synonym in nltk

Query a synset for a word ‘dog’ within part of speech ‘NOUN’ while pos can be NOUN, VERB, ADJ, ADV

```
>>> wn.synsets('dog',pos=wn.NOUN)
[Synset('dog.n.01'), Synset('frump.n.01'),
Synset('dog.n.03'), Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'),
Synset('andiron.n.01')]
```

Figure 8 query a syset within Pos of a synonym in nltk

Query all the lemmas inside a synset using synset ‘dog.n.01’

```
>>> wn.synset('dog.n.01').lemma_names( )
['dog', 'domestic_dog', 'Canis_familiaris']
```

Figure 9 query lemmas inside a synset in nltk

Query lemmas of a synset ‘dog.n.01’

```
>>> wn.synset('dog.n.01').lemmas( )
[Lemma('dog.n.01.dog'),
 Lemma('dog.n.01.domestic_dog'),
 Lemma('dog.n.01.Canis_familiaris')]
```

Figure 10 query lemmas of a synset in nltk

Query antonym of synset ‘good.a.01’

```
>>> good = wn.synset('good.a.01')
>>> good.lemmas()[0].antonyms()
[Lemma('bad.a.01.bad')]
```

Figure 11 query antonym of a synset in nltk

Query shortet path distance between two synsets ‘dog.n.01’ ‘cat.n.01’

```
wn.synset('dog.n.01').shortest_path_distance(wn.synset('cat.n.01'))
4
```

Figure 12 query shortest path distance of a synset in nltk

Query lowest common subsumer between two synsets ‘dog.n.01’ ‘cat.n.01’

```
wn.synset('dog.n.01').lowest_common_hypernyms(wn.synset('cat.n.01'))
[Synset('carnivore.n.01')]
```

Figure 13 query least common subsumer of a synset in nltk

Query semantic similarity between two synsets ‘dog.n.01’ ‘cat.n.01’

```
>>> dog = wn.synset('dog.n.01')
>>> cat = wn.synset('cat.n.01')
>>> dog.path_similarity(cat)
0.2
```

Figure 14 query semantic similarity between two noun synset in nltk

Since “path similarity’ can only be applied in part of speech NOUN and VERB, as a path cannot be found due to Adjectives and Adverbs are not formed into a hierarchy system. ‘Similar to’ is the most useful relationship between adjectives and adverbs.

```
>>> beau.similar_tos()
[Synset('beauteous.s.01'), Synset('bonny.s.01'), Synset('dishy.s.01'), Synset('exquisite.s.04'),
Synset('fine-looking.s.01'), Synset('glorious.s.03'), Synset('gorgeous.s.01'),
Synset('lovely.s.01'), Synset('picturesque.s.01'), Synset('pretty-pretty.s.01'),
Synset('pretty.s.01'), Synset('pulchritudinous.s.01'), Synset('ravishing.s.01'),
Synset('scenic.s.01'), Synset('stunning.s.04')]
```

Figure 15 query semantic similarity between two adj and adv synset in nltk

Brown Corpus

The Brown Corpus(1961) is a large body of text collection by NLTK that was created in 1961 at Brown University. It was the first million-word electronic corpus of American English. This untagged corpus contains plain text from 500 sources, and the sources have been categorized by genre, such as fact, news, editorial, fiction, etc. The information content used in Resnik and Jcn measures are both retrieved from Brown Corpus to perform similarity computation.

SemCor

A sense-tagged corpus named SemCor is a subset of Brown corpus, being used as information content source to obtain from for the Lin measures.

Information Content

The semantic measures described by Jcn, Resnik, and Lin, all of which are based on information content files to perform similarity measures. The result is dependent on the corpus used to generate the information content and the specifics of how the information content was created. There are two approached to retrieve IC(Meng et al. 2014)The traditional approach is copora-dependent IC metric, which measure the IC of word senses using a combination of words statistical data analysis on the actual usage in text a large corpus with also the knowledge of their lexical hierarchy from WordNet ontology (Seco et al. 2004) . The other approach is the corpora-independent IC metric, which is a highly concerned approach in recent years (Meng el al. 2014). Only the conventional copora-dependent IC approach is involved in this project. Brown corpus and SemCor corpus being the statical

analysis resources as Resnik, Jcn and Lin measures require using a corpus in addition to the WordNet to scale information content. The combination WordNet and additional corpus to perform these three measures are shown to achieve better correlations to human judgment than using just WordNet's information content alone (Pedersen 2010). With all the corpora can be accessed by using NLTK , information content are available to be scaled from untagged and sense-tagged corpora to allow performing information-based similarity measures.

2.3 Neo4j

Neo4j is an open-source, NoSQL, ACID-compliant, native graph database relational graph database that stores information as nodes, relationships and properties with the property graph model to structure all these data. (Neo4j Developer Guides,n.d.)

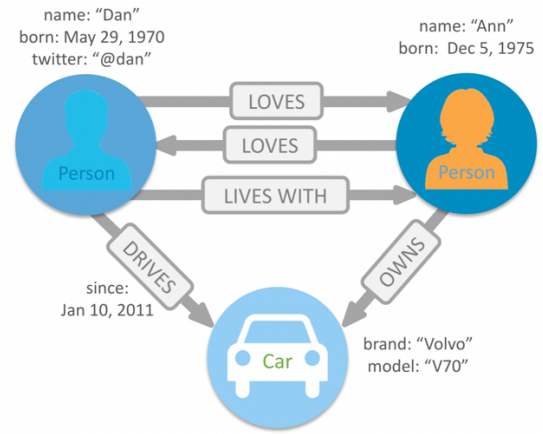


Figure 16 Example Property Graph from Neo4j website

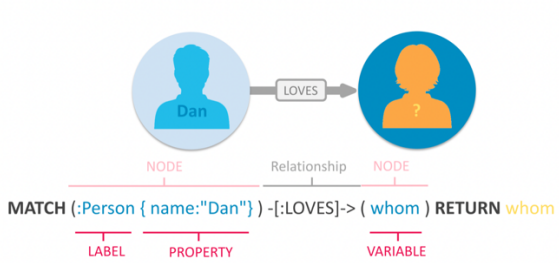


Figure 17 Example Blocks of Property Graph from Neo4j website

Nodes

The Nodes are the entities in the graph that can be tagged with labels, representing their different roles in specific domain. Any number of key-value pairs or properties can be held by nodes. Metadata are attached to Nodes with Label “Person” and Label “Car” to give nodes index or constraint information.

Relationships

Relationships is connections from a start node to an end node that can also have properties. For example, Person LOVES Person.

Properties

Properties is the key-value pairs that attached to nodes. For example, a Node “Person” has a Property “name” with a value of “Dan”.

Cypher

Cypher is a unique query language similar to SQL but optimized for graphs to execute on the database. It provides a visual way of matching nodes and relationships. The efficiency of the representation of nodes and relationships, and its constant time traversals ability in big graphs for not only depth and breadth make Neo4j very easy to work with(Neo4j Developer Guides, n.d.).

```
MATCH (tom:Person {name:'Tom Hanks'})-[rel:DIRECTED]-(movie:Movie)
```

Figure 18 Example Cypher Query from Neo4j website

Neo4j for Graph Data Science (GDS)

The Neo4j Graph Data Science (GDS) library provide some graph algorithms called Path finding algorithms. They find the path between two nodes or evaluate the availability and quality of paths. Dijkstra Source-Target Shortest Path, Dijkstra Single-Source Shortest Path, A* Shortest Path, Yen’s Shortest Path and other algorithms are grouped under Path finding algorithms. (311*)

Awesome Procedures on Cpyher（APOC）

The APOC plugin is an extensive library of Neo4j procedures that can help with data importing as well as path analysis.

Neo4j for Wordnet

As Neo4j can form a knowledge graph network based on interconnected concepts, entities, relationships and events, and then use the relationships and semantic data to put data into the graph. Thus a wordnet knowledge graph can be built in neo4j for wordnet data using synsets, concepts and semantic relationships in wordnet. Neo4j supports python as a driver and has py2neo as a community-driven driver to link to python and neo4j.

Limitation of neo4j semantic similarity

As that said, WordNet is applicable for three path similarity and three corpus-based similarity. The path finding algorithm and GDS library provided on the Neo4j website do not support path similarity for Lch and Wup except the shortest path which has already been implemented to the existing system. For information content-based measures that rely on corpus dataset cannot be computed using neo4j graph database. The limitation of algorithms offered in neo4j result in the existing system lack of rest of wordnet similarity measures that supposed to be completed. The focus on this project is to complete the rest of wordnet similarity on existing system to allow a further comparison study and understanding of each method with practical set of words.

2.4 Flask

Flask is a lightweight web application framework written in Python, also known as a 'microframework'. Flask has no default database, its WSGI toolkit uses Werkzeug and its templating engine uses Jinja2. Flask is licensed under BSD.

WTForms

WTForms is a Flask integration library. It is used to process browser form submissions and pass data from the front-end to the back-end. It extends Flask-WTF and adds a number of helper functions that make forms in Flask richer.

3. Project Approach and Specification

3.1 Approach

The previous section discussed wordnet and its structure and features in detail, as well as WordNet::Similarity Perl Module and its six similarity calculations for wordnet.Thus, implementing these six methods became the focus of this project. Currently, the system in place is to model wordnet into neo4j graph data and implement path finding similarity with the APOC and Graph Data Science Library plug-in provided in neo4j. The remaining five similarity methods covered in the Perl Module are not provided in the Neo4j Graph Algorithm, so it is necessary to find a way to implement the remaining five similarity methods.

Madnani (2007) has provided worked examples and references demonstrating the ease with which NLP tasks can be performed using NLTK. Farkiya et al. (2015) has also conducted research on Natural Language Processing using NLTK and WordNet, which use WordNet as the NLTK as the preferred corpus for discussing how to use NLTK for semantic analysis in NLP, again demonstrating the feasibility of NLTK using wordnet for NLP tasks. Thus, a deeper study of the book NLP with Python - Analyzing Text with the Natural Language Toolkit by the creators of NLTK Bird et al. (2009) reveals that NLTK provides a number of program modules for working with natural language tasks, including the nltk.corpus.reader.wordnet module for the wordnet corpus, which includes the six semantic similarity measures for wordnet contained in the WordNet::Similarity Package. In particular, the corpus-based information content value required for the information content measures can

be successfully obtained from the Brown corpus included in NLTK. The book also presents examples of how to return semantic similarity calculation for wordnet.

Thus NLTK is a sufficient approach for implementing path based similarity and information based similarity for wordnet, both in terms of algorithm and data. At this point the practicality and feasibility of using NLTK to implement wordnet similarity methods has been confirmed. The implementation of the new algorithm has therefore been shifted from Neo4j to the Nature Language Tool-Kit.
The NLTK algorithm complements the similarity algorithm in the existing system, allowing a good cross-sectional comparison between the NLTK and Neo4j . With this, the differences between Neo4j graph and original wordnet hierarchy structure for similarity analysis can be discussed. In order to facilitate an intuitive cross-sectional comparison, the results of the newly implemented algorithm are presented together with those of neo4j.

For the web interface, in order to maintain the uniformity of the existing system interface, Flask is the necessary tool to use to complete the web interface and also WTForm to pass data from front-end to back-end. The app.py receives the source and target words processed by forms.py, then passes the forms data to SimilaryMethods.py to run against wordnet.py which is the source code of the nltk wordnet corpus reader module to calculate the results. Then finally app.py passes these results back to the page for display.

3.2 Architecture of System

The solid blocks in the Figure below cover the improvements for new similarity methods run against NLTK data based on the existing system, while the dashed block shows the process of existing system's run against Neo4j. This involves downloading the NLTK data from its website and creating a new SimilarityMethods.py file to call the algorithm in the nltk corpus reader module and the nltk data package, to compute the results for the data received in forms.py, and then to supplement the app.py file with the new algorithms to handle the data received from forms and the results returned from similarityMethods.py, then finally the app.py file passes all the results to the new web page to display their results in a table.

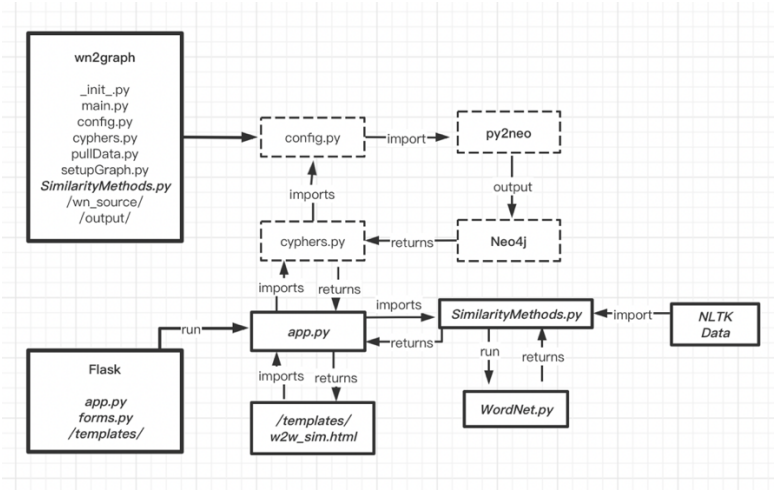


Figure 19 System Architecture

4.Implementation

4.1 Rebuild the existing system with Neo4j

Before the new similarity could be implemented using nltk, the pre-existing system based on the neo4j graph database need to be rebuilt first. All the implementation process for rebuilding the pre-existing system is follow the report written by Hayes(2022).

Load Data into Neo4j

The author of the existing system uses web scraper in python to receive the list of IDs and collect the data into its corresponding part of speech file, that is, the WordNet RDF is pulled down in the form of JSONdata, and finally the data is compiled into four packaged files according to the four parts of speech The data is compiled into four packaged files, "data.adj.json", "data.adv.json", "data.noun.json", "data.verb.json". Since this is a tedious and time-consuming process, and only needs to be done once.

Figure 20 Synset data files to be loaded into Neo4j

For importing ready data into graph database, Neo4j Desktop 1.4.15, running Neo4j 4.4.7 was used. The process is create a new project name as "WordNet" in Neo4j desktop and create a new local database "Local DBMS" with credentials for hosting graph and then load data into the database just created. The specific steps are as follow.

Creating a new project in Neo4j desktop.

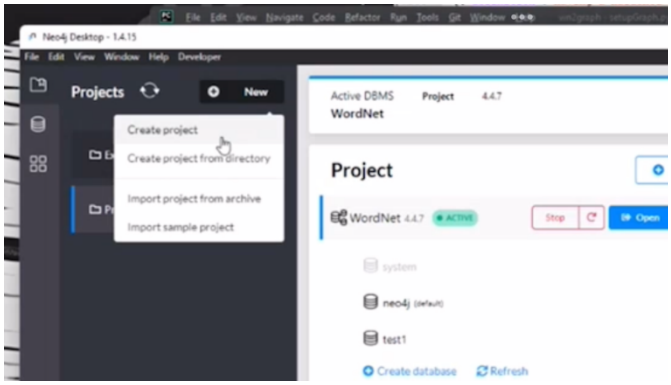


Figure 21 Create new project in Neo4j Desktop

Edit Project name as: WordNet

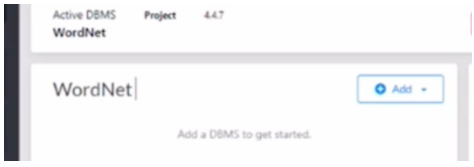


Figure 22 Edit new project name in Neo4j Desktop

Create a new "Local DBMS"

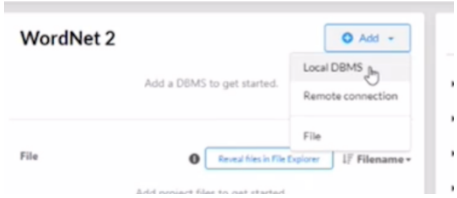


Figure 23 Sample of creating a Local DBMS

Edit username as "WordNet" and password as "123", choose version as neo4j 4.4.7 . These credentials are used in config.py for py2neo to connect Python and Neo4j

Figure 24 set up credentials for Local DBMS

Now the database is created, then APOC 4.4.0.5 and Graph Data Science Library (GDS) 2.0.4 are two essential plugins need to be downloaded to enable performance cypher queries and path finding procedures.

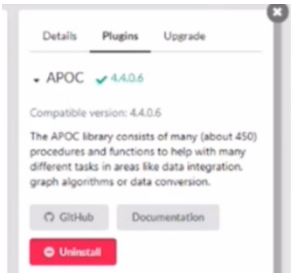


Figure 25 install APOC and GDS plugin

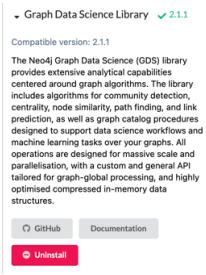


Figure 26 install APOC and GDS plugin

Edit setting to change security permission to allow APOC plugin to read and write as well as to enable the graph data science procedures. And then we also need to increase the heap size, which when use graph data science, it will create a temporary graph, which is called a projected graph. This graph is stored in RAM memory, so we need to increase the heap size to support it and allow handling more queries.

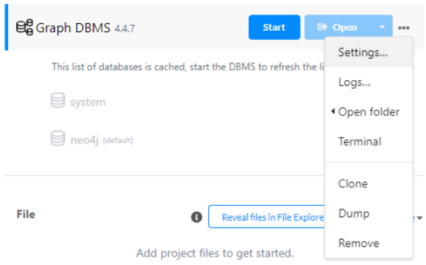


Figure 27 Access database configuration

Config Changes
dbms.memory.heap.max_size=4G

dbms.security.procedures.unrestricted=jwt.security.*,apoc.*,n10s.*,gds.*
New lines
apoc.export.file.enabled= true
apoc.import.file.enabled= true

Figure 28 Config changes of Neo4j

So far with the database config set up, we can then drag the ready JSON files into the import folder of Neo4j to run queries against them.

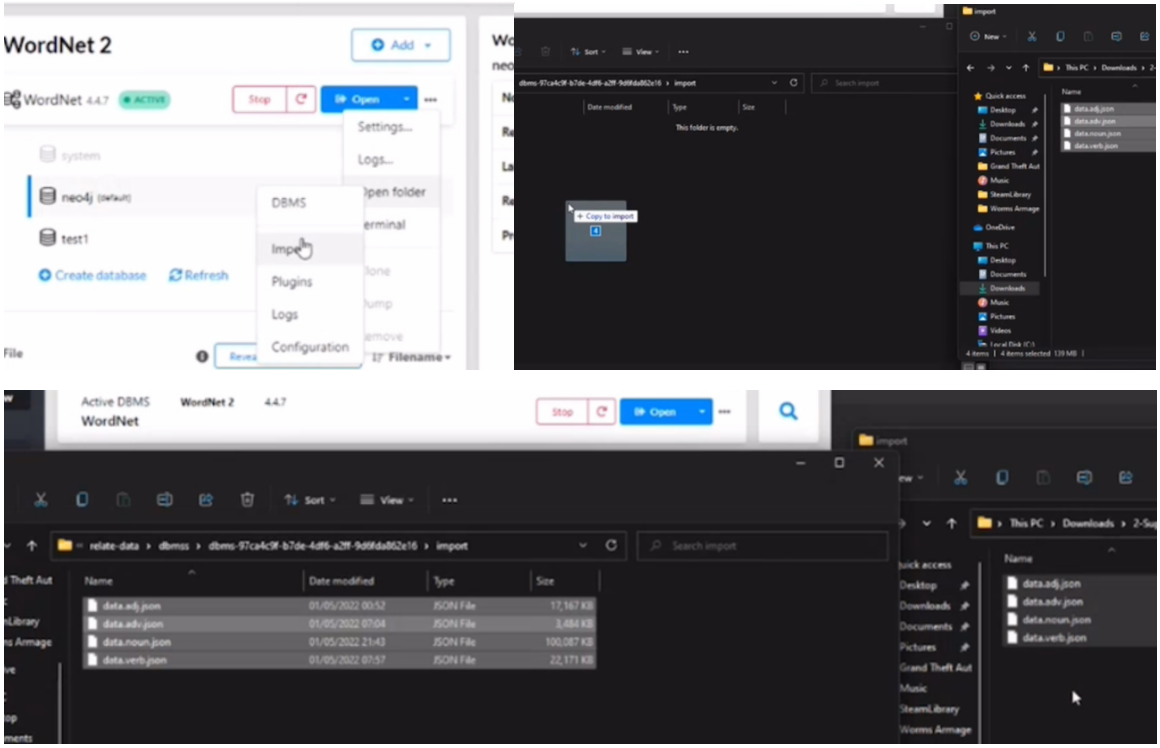


Figure 29 JSON files into import folder for Neo4j

With these four main files, we run setupgraph.py to build and model graph builds the database and inputs the data and puts all the relationships in there. Also we run config.py that stores configurations and common used variables and relationships required when graph setting. And then once we have running database, they're able to query. Then the flask which hosts the web page with the cyphers that will calculate the path similarity, Dijkstra weighted path based similarity and gds similarities which does not work in the end. With four main files and flask running, the cypher.py will use the config as well.

Python package called “py2neo” as a connection between the python and the neo4j database. With the graph local database credentials and Py2neo, a graph object is ready to set up.

```
from py2neo import Graph

# region Default values
# Graph database credentials
uri = "bolt://localhost:7687"
dbUser = "neo4j"
dbPass = "123"
dbName = "neo4j"
graph = Graph(uri, auth=(dbUser, dbPass), name=dbName)
```

Figure 30 Graph database credentials in config.py

Relations

A dictionary named wn_relations stores all the 25 semantic relations within WordNet RDF that allow similarity analysis. Every key in the dictionary is for searching inside JSON files when data inputting to build semantic relations. Each key follows a value to store a new relation name and a cost value that used for Dijkstra path weighted similarity.

```
# WordNet relations between senses and synsets stored in JSON files
# Key = JSON relation
# Index 0 = Changed relation name for neo4j
# Index 1 = Cost, used for weighted pathfinding
wn_relations = {
    "also": ["wn__also", 60],
    "antonym": ["wn__antonym", 90],
    "attribute": ["wn__attribute", 20],
    "causes": ["wn__causes", 40],
```

Figure 31 Sample of wn_relations in config.py

Lexical relation value “rel_lexSense” is stored for creating the initial model, “rel_sense” and “rel_canForm” is for model the remaining data. These relations also store a cost value within the relationship.

```
# Lexical Relations
# Index 0 = Relation name for neo4j
# Index 1 = Cost, used for weighted pathfinding
rel_canForm = ["canonicalForm", 0]
rel_sense = ["sense", 0]
rel_lexSense = ["lexicalisedSense", 100]
```

Figure 32 lexical relations stored in config.py

Nodes labels are stored as lexical_form, lexical_entry, lexical_sense and lexical_concept. Part of speech node labels are stored with their own ss_type.

```
# Lexical Node Lables
n_Form = "lex_Form"
n_Entry = "lex_Entry"
n_Sense = "lex_Sense"
n_Concept = "lex_Concept"
# PoS Nodes:
# Key = Node name in Neo4j
# Value = sysnet PoS
pos_nodes = {
    "Adjective": "a",
    "AdjectiveSatellite": "s",
    "Adverb": "r",
    "Verb": "v",
    "Noun": "n"
}
```

Figure 33 Node labels in config.py

A list is contained with part of speech file names for looping over the four main JSON files.

```
# PoS file names "data.pos.json"
file_pos = ["adj", "adv", "verb", "noun"]
```

Figure 34 POS file names in config.py

For setting up the graph, Py2neo graph object set up in config.py in Figure 35 was used to communicate with neo4j ,pass cypher queries as a strings and it will run against the graph database. And then “cypher.data()” was used to receive returned output from neo4j graph. “checkFiles()” is to check if all the “data.POS.json” files are ready before setting up the graph.

```
def checkFiles():
    for pos in config.file_pos:
        cypher = graph.run(f'''
            // Checking if files are imported by running query
            WITH "file://data.{pos}.json" AS path
            CALL apoc.load.json(path) yield value
            unwind keys(value) as key
            RETURN key, apoc.meta.type(value[key]) limit 12
            ''')

    return print("All files are imported and accesible\n-----<Setting up graph>-----")
```

Figure 35 checkFiles queriy in setupGraph.py

Now to begin the setting up graph, indexes and constraints are added to help maintain the graph structure. For instance, synsets can not share definition with each other as synonyms require a unique semantic meaning. Thus, “IndexConstraints” is created on lexical nodes including lexical_Form, lexical_Entry, lexical_Sense and lexical_Concept to constraint them have one unique key sense and lexical_Form have one unique written expression.

```
def IndexesConstraints():
    print(f"---\nStarting: {inspect.stack()[0][3]}")

    constraint_Concept = graph.run(f''' //Constraint for lexical Concepts to be unique
    CREATE CONSTRAINT con_{config.n_Concept}_id IF NOT EXISTS  FOR (s:{config.n_Concept}) REQUIRE (s.wn_id) IS UNIQUE''')

    index_Concept = graph.run(f''' //Index for Concepts on synset ID
    CREATE TEXT INDEX index_{config.n_Concept}_id IF NOT EXISTS FOR (s:{config.n_Concept}) on (s.wn_id)''')
```

Figure 36 Sample IndexConstraints() in setupGraph.py

Loading data into graph is start after indexes and constraints are set up. Each of the "data.POS.json" files are used with InputData() to create the synset nodes as lexical_Concept, and synonym nodes as lexical_Sense, and the relationship named "lexicalized Sense" is added with a direction starting from sense to concept. When running the cypher queries to input the data to read the JSON data, it first UNWIND the row to expand a list into a sequence of rows and allow accessing the list by using the variable name of the list and property key.

```
def InputData():
    print(f"---\nStarting: {inspect.stack()[0][3]}")
    for p in config.file_pos:
        cypher = graph.run(f'''
        // CREATE Synset + Lemmas
        // OMITTED old_keys, foreign, links, ili, gloss (null)

        WITH "file://data.{p}.json" AS path
            CALL apoc.load.json(path) yield value
            unwind value as data
            unwind data.lemmas as lemmas
            MERGE (s:{config.n_Concept} {{wn_id: data.id}})
            ON CREATE SET s += {{wn_definition: data.definition,
                                wn_example: data.examples,
                                wn_pos: data.pos,
                                wn_subject: data.subject}}
            WITH s, data, lemmas
            MERGE (l:{config.n_Sense} {{wn_sense_key:lemmas.sense_key}})
            ON CREATE SET l += {{wn_lemma: lemmas.lemma,
                                wn_pos: data.pos,
                                wn_language: lemmas.language,
                                wn_forms: lemmas.forms,
                                wn_subcats: lemmas.subcats,
                                wn_importance: lemmas.importance}}
            MERGE (s)<-[r:{config.rel_lexSense[0]}]-(l)
            ON CREATE SET r+= {{cost:{config.rel_lexSense[1]}}}
            RETURN COUNT(r)
        ''')
        print(f"POS File: {p} OUTPUT: {cypher.data()}")
```

Figure 37 InputData() in setupGraph.py

Now the basic lexical structure is completed for each POS without semantic relations. Then the BuildRelationshipsWN() function is used for implementing semantic relations between senses. It first MATCH on the nodes that share the synset ID within the current row of the JSON files. Then the query is run to build relations over four main POS files. To help manage the memory cost for the queries apoc.periodic.iterate is used. It splits the queriy into batches to reduce the risk of out of memory exception.

```
def BuildRelationshipsWN():
    print(f"---\nStarting: {inspect.stack()[0][3]}")

    for wn_rel, rel_item in config.wn_relations.items():
        print(f"Relation with word pointers: {wn_rel}")
        for p in config.file_pos:
            cypher = graph.run(f'''
            // Relationships ({wn_rel})

            CALL apoc.periodic.iterate('
                WITH "file://data.{p}.json" AS path
                CALL apoc.load.json(path) yield value
                unwind value as data
                unwind data.lemmas as lemmas
                unwind data.relations as relations
                WITH data.id as src_id,
                relations.rel_type as rel_type,
                relations.src_word as src_word,
                relations.trg_word as trg_word,
                relations.target as trg_id
                MATCH (s:{config.n_Concept}) WHERE s.wn_id = src_id AND rel_type = "{wn_rel}" AND src_word IS NULL
                MATCH (b:{config.n_Concept}) WHERE b.wn_id = trg_id AND trg_word IS NULL
                RETURN s, b, rel_type',
                'WITH s, b, rel_type
                MERGE (s)-[r:{rel_item[0]}]->(b)
                ON CREATE SET r += {{rel_type:"{rel_item[0]}",cost:{rel_item[1]}}}
                RETURN COUNT(r)', {{batchSize:100, parallel:false}})''')
            print(f"POS: {p} COUNT: {cypher.data()}")
```

Figure 38 BuildRelationshipsWN() between synset and synset in setupGraph.py

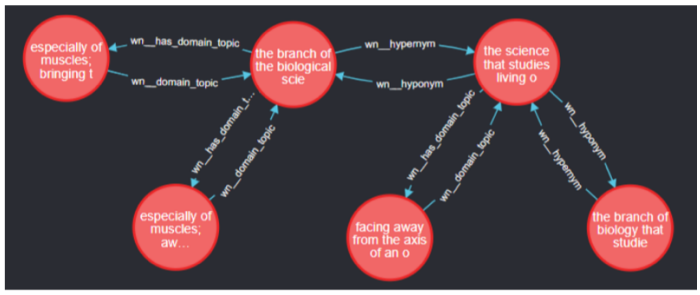


Figure 39 Example of synset to synset relations

Modelling the Graph

The lexical structure of the graph is only Senses to a Synset, part of the OntoLex-Lemon model still need to implement. The current model only hav senses and concepts and their relationships.

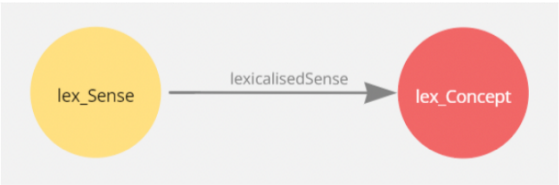


Figure 40 Current lexical structure model of Wordnet Graph

Then we add “lex_Form” as the written expression of the sense, “lex_Entry” as their Pos, and then a relation “canonicalForm” with direction start from “lex_Form” to “lex_Entry”.

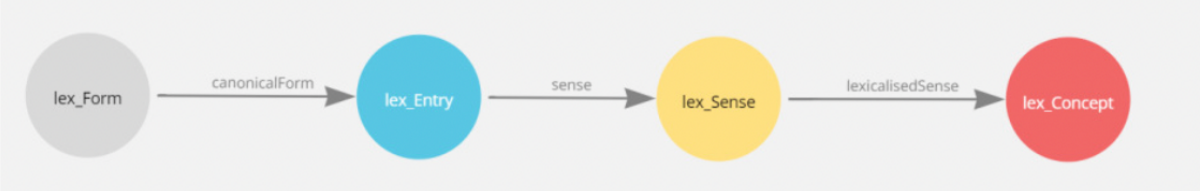


Figure 41 Completed lexical structure model of Wordnet Graph

A function BuildEntryToSense() is used to implement the lex_Entry into the model by matching each node's part of speech and returning a distinct lemma and merging to create a node for the each lemma in that part of speech.

```
def BuildEntryToSense():
    print(f"---\nStarting: {inspect.stack()[0][3]}")
    for pos_node, pos in config.pos_nodes.items():
        print(f"Creating Nodes :{config.n_Entry}:{pos_node}")

        cypher = graph.run(f'''
            // Create node that links canonical form of Senses to canonical node
            MATCH (l:{config.n_Sense} {{wn_pos:"{pos}"}})
            WITH DISTINCT l.wn_lemma as dist_lemma, l.wn_pos as pos
            MERGE (e:{config.n_Entry}:{pos_node} {{lex_canonicalForm: dist_lemma, wn_pos: pos}})
            RETURN COUNT(e);
            ''')
```

Figure 42 BuildEntryTOSense() in setupGraph creating lex_Entry nodes and adding POS label to node

```
    cypher = graph.run(f'''
        // Linking the nodes together
        MATCH (e:{config.n_Entry}:{pos_node})
        MATCH (l:{config.n_Sense} {{wn_pos:"{pos}"}})
        WHERE e.lex_canonicalForm = l.wn_lemma AND e.wn_pos = l.wn_pos
        WITH e, l
        MERGE (e)-[r:{config.rel_sense[0]}]->(l)
            ON CREATE SET r.cost = {config.rel_sense[1]}
        RETURN COUNT(r)
        ''')
```

Figure 43 BuildEntryTOSense() in setupGraph creating relationships between lex_Entry and lex_Sense

Then still using BuildFormToEntry() to add lex_Form to the graph model while the lex_Form is the written representation of this nodes, which means is a word form without any meaning on it.

```
def BuildFormToEntry():
    print(f"---\nStarting: {inspect.stack()[0][3]}")

    print(f"Creating distinct {config.n_Entry} Nodes")
    cypher = graph.run(f'''
        // Creates distinct nodes from Entry's canonicalForms
        CALL apoc.periodic.iterate('
        MATCH (e:{config.n_Entry})
        WITH DISTINCT e.lex_canonicalForm as writtenForm
        RETURN writtenForm','WITH writtenForm
        MERGE (f:{config.n_Form} {{lex_writtenForm: writtenForm}})',
        {{batchSize:100, parallel:True}})
        ''')
    print(f"OUTPUT:{cypher.data()}\nFinished creating")

    print(f"Linking {config.n_Form} to {config.n_Entry}")
    cypher = graph.run(f'''
        // Links the Form nodes to Entry nodes
        CALL apoc.periodic.iterate('
        MATCH (e:{config.n_Entry})
        MATCH (f:{config.n_Form})
        WHERE e.lex_canonicalForm = f.lex_writtenForm
        RETURN e, f',
        WITH e, f
        MERGE (f)-[r:{config.rel_canForm[0]}]->(e)
            ON CREATE SET r.cost = {config.rel_canForm[1]}',
        {{batchSize:100, parallel:False}})
        ''')
    print(f"OUTPUT:{cypher.data()}\nFinished linking")
    print(f"Finished: {inspect.stack()[0][3]}\n---")
```

Figure 44 BuildEntryTOSense() in setupGraph creating lex_Form nodes and links to lex_Entry

Now run the main() function in setupGraph.py to make sure the JSON files are imported and finish building the graph.

```
def main():
    try:
        checkFiles()
        try:
            start = time.time()
            IndexesConstraints()
            InputData()
            BuildRelationshipsWN()
            BuildEntryToSense()
            BuildFormToEntry()
            end = time.time()
            print(f"Finished in: {end - start}")
        except:
            print("Unabled to setup up graph.")
    except:
        print("Cannot find or missing imported files, check folder laction")
```

Figure 45 main.py in setupGraph.py

Now the graph is set up, we can check the structure in neo4j use cypher query:

MATCH p=(a:lex_Form) WHERE a.lex_writtenForm =“open” RETURN p limit 1

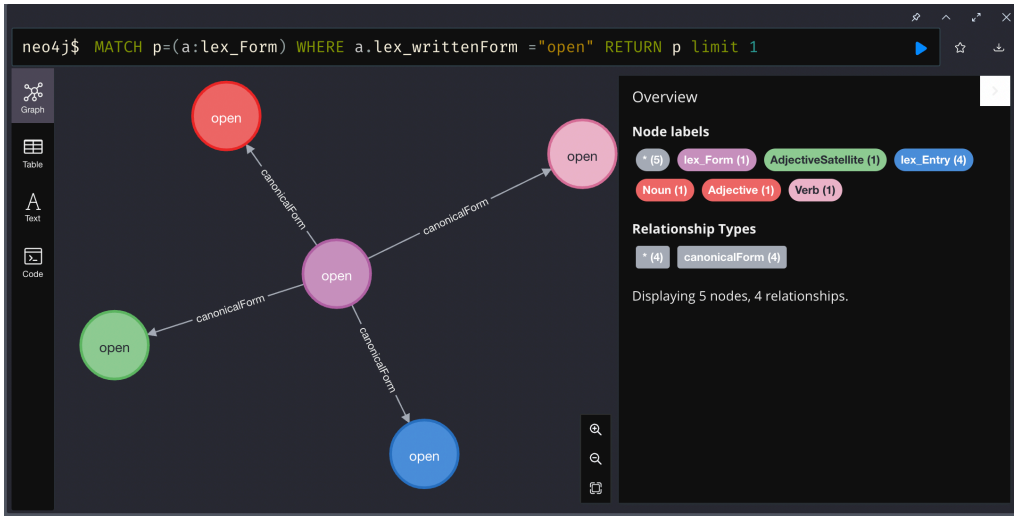


Figure 46 Example of graph model in Neo4j

Now you have written word form which is the simplest form without meaning, and it's each part of speech -v, n, a, s. And you click one of the part of speech like "v", and it get stands out, so that you got all the senses. So the "verb" version of "open" have all these different senses.

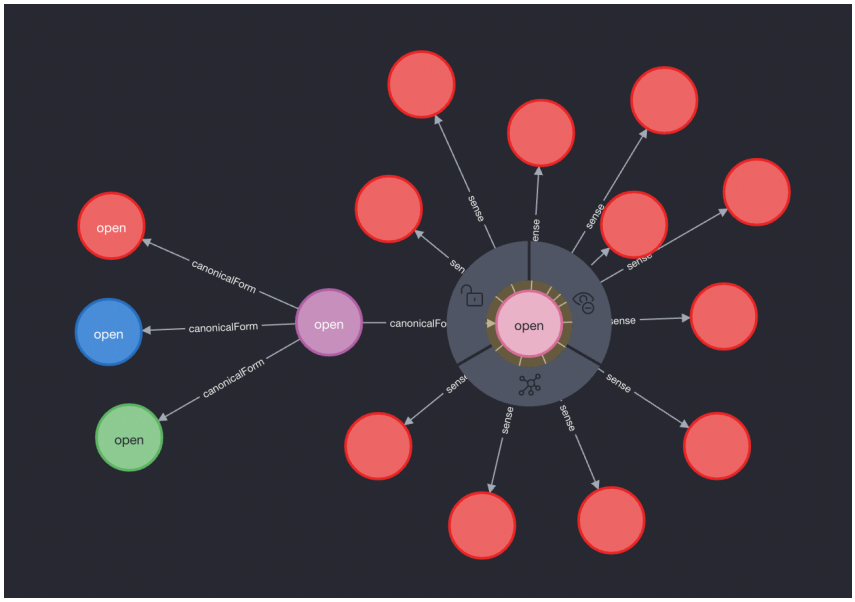


Figure 47Example of graph model in Neo4j

Simplified the graph a little bit and click one of these senses, it links to the concept which is the definition of this selected sense. Now here is the basic structure of graph model.

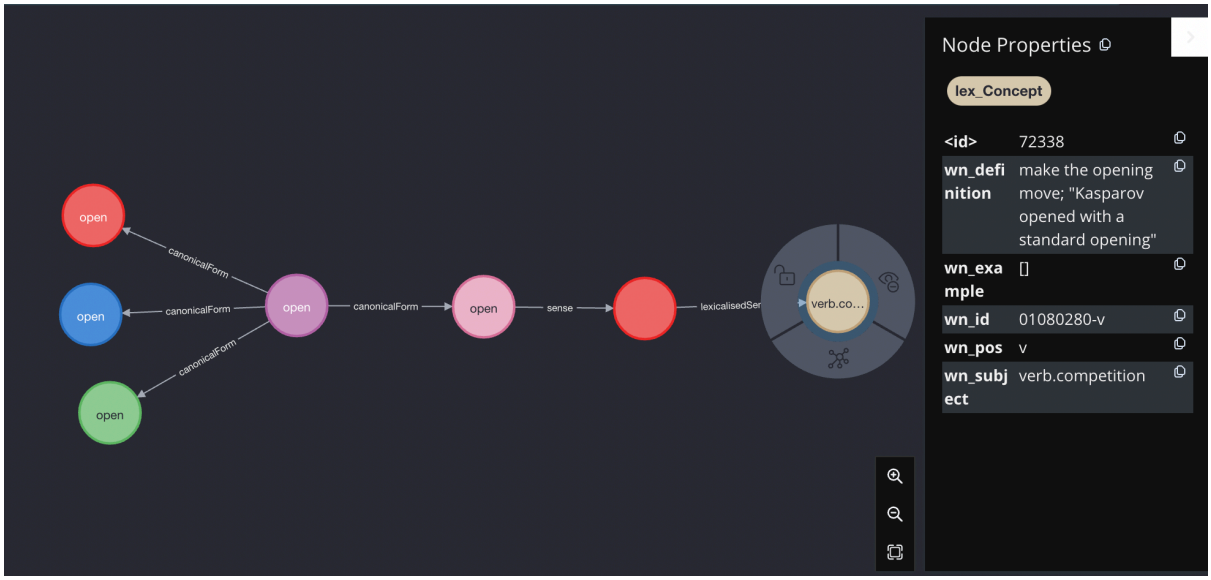


Figure 48 Example of graph model in Neo4j

Neo4j Similarity Analysis

The similarity analysis between word to word can be performed when Wordnet stored and modelled into a Neo4j graph. The basic shortest path similarity is used from neo4j built-in method and the Dijkstra weighted path analysis is used from the GDS plug-in methods.

Shortest Path Similarity

This method counts the minimum edges from the source node to target node to indicate semantic similarity using the equation below, s represents source word, t represents target word.

$$pathSim(s,t) = \frac{1}{1 + pathLength(s,t) - 4}$$

Minus four is because the graph model is start from Form to Entry to Sense to Concept, but the edge counting is start from Sense to Concept. Thus, for each source word and target word, the two edges from each Form to each Entry is not counted. Then we need to get the shortest path length between source node and target node by using cypher queries:

MATCH p1=shortestpath((n:lex_Form{lex_writtenForm:"source"})-[*]-(:lex_Form{lex_writtenForm:"target"})) RETURN p1

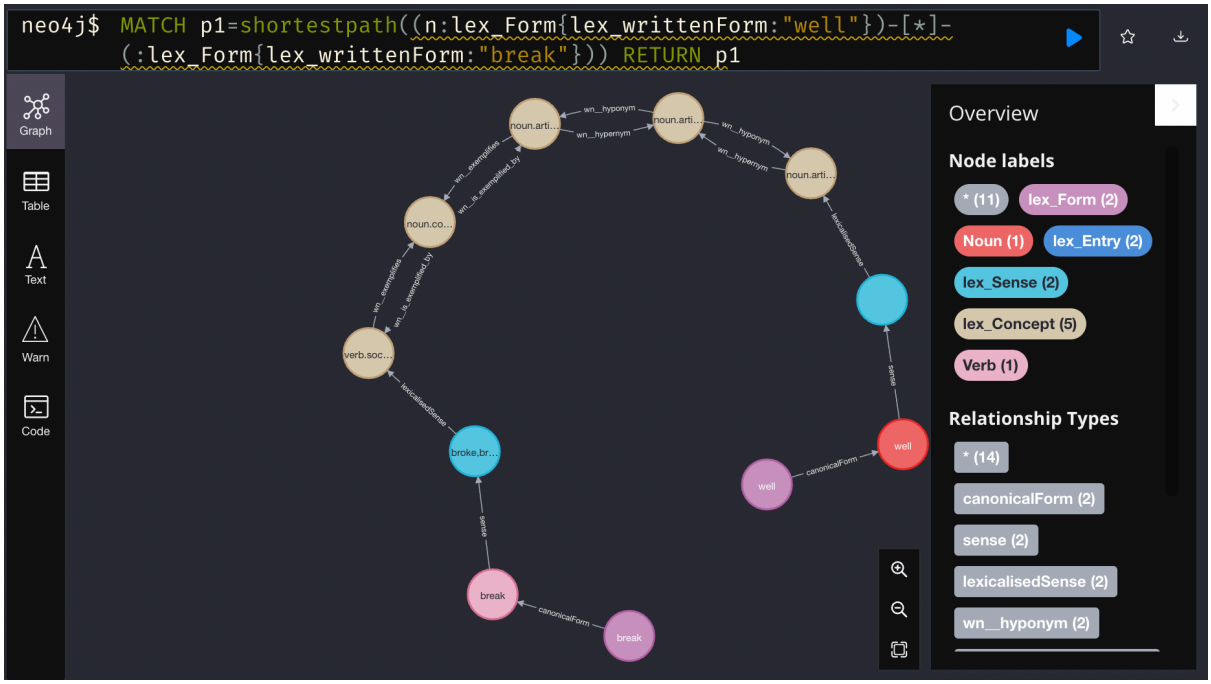


Figure 49 Example of shortest path from blue nodes "Sense" to "Sense"in Neo4j

```
// Path Similarity Word to Word
MATCH (n:lex_Form {lex_writtenForm:"well"})
MATCH (t:lex_Form {lex_writtenForm:"break"})

MATCH shortP=shortestpath((n)-[*0..15]-(t))

RETURN n.lex_writtenForm, t.lex_writtenForm,
    length(shortP)-4 as pathLength,
    (1.0/(1+length(shortP)-4)) as pathSim
```

Figure 50 Example of path similarity in Neo4j

	n.lex_writtenForm	t.lex_writtenForm	pathLength	pathSim
1	"well"	"break"	6	0.14285714285714285

Figure 51 Results of path similarity in Neo4j

Dijkstra Weighted Path Length Similarity

The concept of Dijkstra weighted path similarity is the same as for shortest path similarity, except that Dijkstra uses weighted path to calculate the path length. The weight is given from cost value that included in wordnet relations in config.py. With the cost value being higher,

the similarity is being lower and vice versa. The table of relations and their cost is in the appendix, Figure 101 to Figure 104.

The GDS library is needed to use Dijkstra weight for shortest path and GDS procedures always require a projected graph that stored in memory and need to be created for every GDS usage. For performing this similarity query, the whole graph is used.

```
// Create projected graph for GDS
CALL gds.graph.project.cypher(
    'all','
    MATCH (n)
    RETURN id(n) as id',
    'MATCH (n)-[r]-(m)
    RETURN id(n) as source, id(m) as target, type(r) as type, r.cost as cost')
    YIELD graphName as graph, nodeCount AS nodes, relationshipCount AS rels
```

Figure 52 Create a projected graph for using GDS

```
// Dijkstra Path Similarity Word to Word
MATCH (n:lex_Form {lex_writtenForm:"well"})
MATCH (t:lex_Form {lex_writtenForm:"break"})

CALL gds.shortestPath.dijkstra.stream("all", {
    sourceNode: n,
    targetNode: t,
    relationshipWeightProperty: "cost"
}) YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path as dijkstraPath
unwind costs as costsU
RETURN
    (length(dijkstraPath)-4) as dijkstraPathLen,
    (totalCost) as dijkstraTotalCost,
    avg(costsU)  as dijkstraAvgCost,
    (1.0/(1+length(dijkstraPath)-4)) as dijkstraPathSim,
    costs
```

Figure 53 Example Dijkstra query using GDS with "well" and "break"

dijkstraPathLen	dijkstraTotalCost	dijkstraAvgCost	dijkstraPathSim	costs
8	430.0	210.38461538461536	0.1111111111111111	[0.0, 0.0, 0.0, 15.0, 115.0,

Figure 54 Results of Dijkstra query

4.2 Install NLTK library

This section shows how python was used to download NLTK corpora data files in dat format by installing NLTK library using Jupyter Notebook.

To install NLTK, we create a new python3.8 notebook using the Jupiter Notebook web application that launched by Anaconda Navigator. Then we run code below to open a NLTK downloader.

```
In [1]: import nltk
        nltk.download()
Out[1]: True
```

Figure 55 Open NLTK downloader

At this point, an NLTK Downloader window opens, click on Download button finish downloading All packages

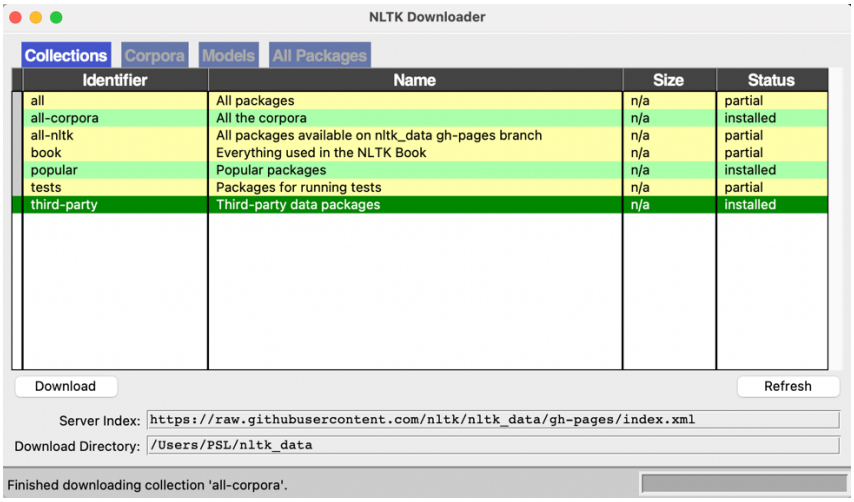


Figure 56 Finished downloading collections 'all-corpora'

We are now have all the corpora ready including wordnet and wordnet information content.

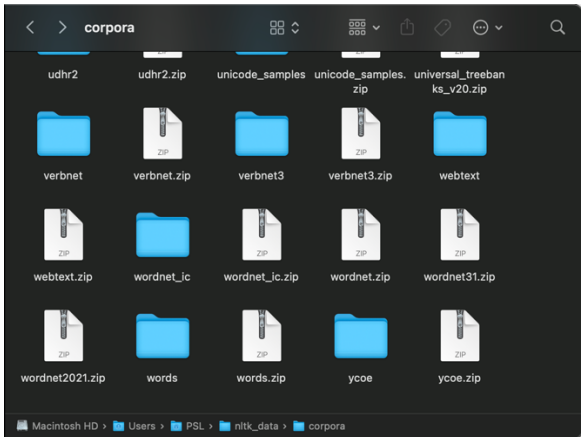


Figure 57 Location of WordNet and wordnet_ic corpus

After finishing downloading, we can also queries the synsets of word, e.g. motorcar, etc. to test whether the corpus has been downloaded and function successfully.

The representation of NLTK Wordnet corpus can seem like Wordnet RDF. The RDF is web based and is descriptive information, like the structure of NLTK corpus can also be processed into RDF format, which is meant to support the relationship between source data and

semantic relations. The figure below is an example of some of the wordnet data in the nltk library.

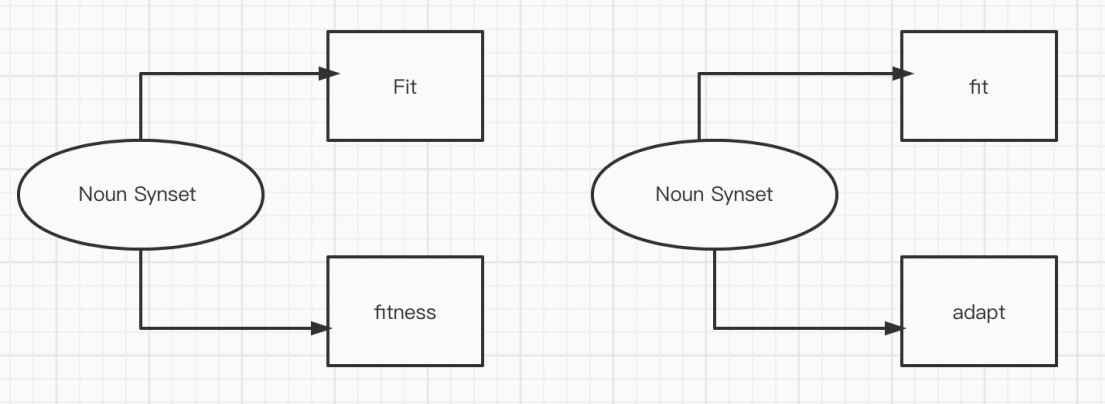


Figure 58 Example of schema

The words are considered as labels because it cannot be linked by different semantic relations. In Figure 58, the words “fit” occur as two labels with no semantic relationship between them.

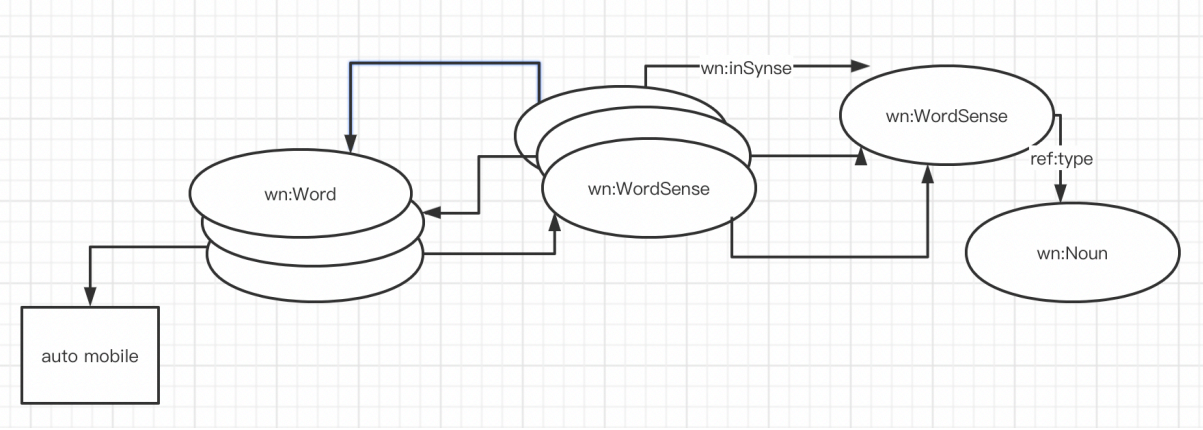


Figure 59 Diagram of schema by wordnet

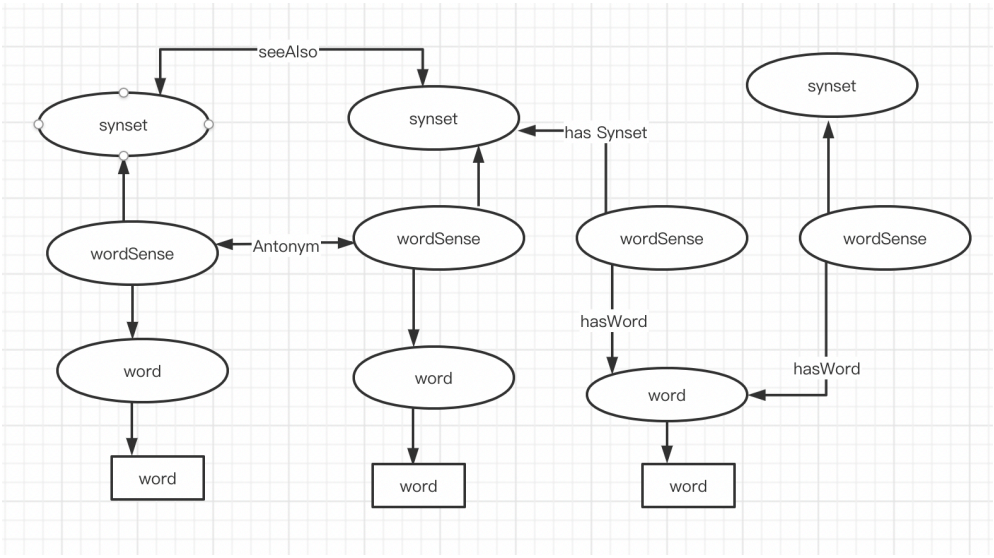


Figure 60 WordNet schema RDF representation

The model is composed by three layers. The first layer is "Word", second layer is "word sense", third layer is "synset" layer. A "word" can be related to various "synset" through several word sense.

With NLTK corpus download and import, queries can be run against the corpus data.

jupyter Untitled1 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
In [10]: from nltk.corpus import wordnet as wn
         wn.synsets("motorcar")

Out[10]: [Synset('car.n.01')]

In [13]: from nltk.corpus import wordnet as wn
         wn.synset("car.n.01").definition()

Out[13]: 'a motor vehicle with four wheels; usually propelled by an internal com
         bustion engine'

In [14]: from nltk.corpus import wordnet as wn
         wn.synset("car.n.01").examples()

Out[14]: ['he needs a car to get to work']

In [17]: wn.synsets('bank')

Out[17]: [Synset('bank.n.01'),
          Synset('depository_financial_institution.n.01'),
          Synset('bank.n.03'),
          Synset('bank.n.04'),
          Synset('bank.n.05'),
          Synset('bank.n.06'),
          Synset('bank.n.07'),
          Synset('savings_bank.n.02'),
          Synset('bank.n.09'),
          Synset('bank.n.10'),
          Synset('bank.v.01'),
          Synset('bank.v.02'),
          Synset('bank.v.03'),
          Synset('bank.v.04'),
          Synset('bank.v.05'),
          Synset('deposit.v.02'),
          Synset('bank.v.07'),
          Synset('trust.v.01')]
```

Figure 61 Queries of synsets, definition and example of words

4.3 Similarity analysis

With NLTK WordNet corpus data downloaded, we can perform WordNet::similarity analysis for our project. We implement both path-based measures and information content corpus-based measures using Natural Language Toolkit: WordNet which is another NLTK corpus reader.[9] A single root node that can be shared between various verb taxonomies do not exists, which will result in the measures working based on synsets cannot connect to each other properly. Thus WordNet::Similarity provides a fake root that connects all the taxonomies and can be turned on and off . This root can be enabled by simply set it to Ture. When enabled, one root node subsumes all the verb concepts and will allow similarity measures to be applied to any pair of verbs. Conversely, when the fake root is not enabled, then the concepts must be in the same is-a hierarchy to allow similarity measures working properly. For the noun taxonomy, there is usually a default root except for WordNet version 1.6.

Before similarity analysis start, we need to import wordnet into python first.

```
from nltk.corpus import wordnet
```

Figure 62 wordnet interface

Path-based Similarity Measures

This section looks at three path-based measures that counting the edges and then convert the edges from distance to similarity: path, lch(Leacock & Chordorow 1998)and wup(Wu-Palmer 1994) by applying NLTK WordNet Similarity.

Path Similarity

Path similarity calculates the similarity between two words using nltk wordnet corpus reader by returning a score range 0-1 based on the shortest path in the taxonomy. They start at one of the word sense in the is-a (hypernym/hypnoym) taxonomy and see how many moves at least it takes to reach the other word sense. For example, word1: deer to word2:elk takes 1 distance, so Path Sim= 1/(1+1) = 0.5, deer to giraffe takes 2 ,path sim= 1/3 = 0.33. Using formular: Path Similarity=1/(distance+1). A score denoting the similarity of the two objects normally between 0 and 1.

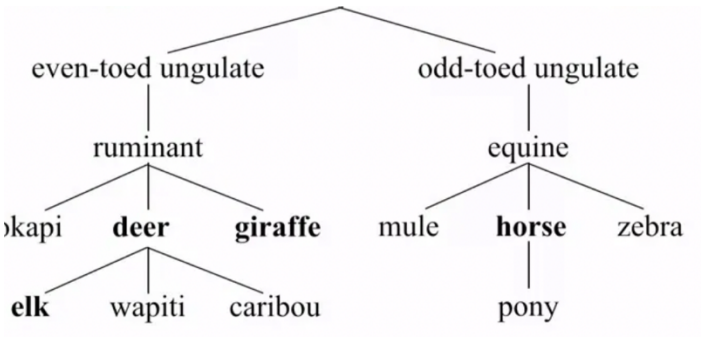


Figure 63 Sample of path that connects the senses in the is-a (hypernym/hypnoym) taxonomy

```
def path_similarity(self, other, verbose=False, simulate_root=True):
    """..."""

    distance = self.shortest_path_distance(
        other,
        simulate_root=simulate_root and (self._needs_root() or other._needs_root()),
    )
    if distance is None or distance < 0:
        return None
    return 1.0 / (distance + 1)
```

Figure 64 Source code of path similarity calculation written in nltk Corpus reader

Look up source word and target word using synsets()function so we can introduce synsets of given source word and given target word from wordnet. After we have the synset of two words we take the first sense element from their synsets as input to apply to the similarity method. Then we can return the path similarity value between two words using synset1.path_similarity(synset2) to call the path similarity equation stored in source code of nltk corpus reader. In case results cannot be returned as expect, we will return a default score 0.34.

```
##path_Similar
def pathSimilar(source, target):
    try:
        word1syn = wordnet.synsets(source)[0]
        word2syn = wordnet.synsets(target)[0]
        pathSimValue = word1syn.path_similarity(word2syn)
        return pathSimValue
    except Exception as e:
        return 0.34
```

Figure 65 Code to return path similarity value

Lch similarity

The lch(Leacock & Chordorow 1998) similarity based on the shortest path between two words but extends the similarity calculation by incorporating the maximum depth in the is-a

hierarchy of the queried taxonomy. The max depth is measured by the length of the longest hypernym path from a synset to the root. As a result, the lch formula is the negative logarithm of the shortest path between the two concepts (synset1 and synset2) divided by twice the total taxonomy depth. The returning results are normally greater than 0.

```
def lch_similarity(self, other, verbose=False, simulate_root=True):
    """..."""

    if self._pos != other._pos:
        raise WordNetError(
            "Computing the lch similarity requires "
            "%s and %s to have the same part of speech." % (self, other)
        )

    need_root = self._needs_root()

    if self._pos not in self._wordnet_corpus_reader._max_depth:
        self._wordnet_corpus_reader._compute_max_depth(self._pos, need_root)

    depth = self._wordnet_corpus_reader._max_depth[self._pos]

    distance = self.shortest_path_distance(
        other, simulate_root=simulate_root and need_root
    )

    if distance is None or distance < 0 or depth == 0:
        return None
    return -math.log((distance + 1) / (2.0 * depth))
```

Figure 66 Source code of lch similarity stored in nltk Corpus reader

```
    #                           for root in h.root_hypernyms()))
    def max_depth(self):
        """
        :return: The length of the longest hypernym path from this
            synset to the root.
        """

        if "_max_depth" not in self.__dict__:
            hypernyms = self.hypernyms() + self.instance_hypernyms()
            if not hypernyms:
                self._max_depth = 0
            else:
                self._max_depth = 1 + max(h.max_depth() for h in hypernyms)
        return self._max_depth
```

Figure 67 Source code of maximum taxonomy depth stored in nltk corpus reader

Same process as path similarity when return the lch similarity results. After introducing synsets of source word and target word in wordnet. Then we can return the lch similarity value between two words using synset1.lch_similarity(synset2) to call the calculation stored in source code of nltk corpus reader. In case results cannot be returned as expect, we will return a default score 0.34.

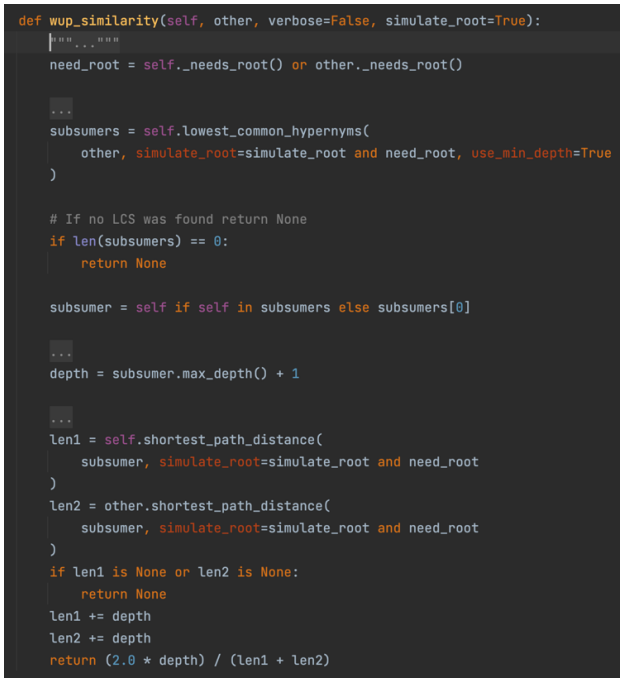
```
##lch
def lchSimilar(source, target):
    try:
        word1syn = wordnet.synsets(source)[0]
        word2syn = wordnet.synsets(target)[0]
        lch_value=word1syn.lch_similarity(word2syn)
        return lch_value
    except Exception as e:
        return 0.34
```

Figure 68 Code to return path similarity value

Wup similarity

As mentioned in background section, the wup similarity measure is originally proposed for verb meanings that organized in different conceptual domains, it limits two concepts need to be within same conceptual domain of hierarchical structure when define how similar they are. To connect words in wordnet from different taxonomies to avoid path not found situation, an imaginary root is created. Then a correction that add one value to the maximum depth of lcs arise for Nouns to account for the fake root node. Because a default root node named Entity always exist in Noun taxonomy, the one value added represents the 1 edge start from Root Node Entity Fake Root node. However, non-Noun words from other part of speech do not need to add one since the flag of simulate root is handling it automatically. With the correction and basic wup equation , it returns a score denoting how similar two word senses are, by considering the depths of the two words in the hypernym tree, along with the depth of their LCS(Least Common Subsumer) which is the number of nodes on the path from their most specific common ancestor node to the root. Due to the LCS is a necessity of wup similarity and the depth of the root of taxonomy is one, therefore the depth of the LCS will never be zero. The returning score range is 0<score<1.

$$\mathbf{Wu-Palmer} = 2 * \frac{\mathbf{depth}\ (\mathbf{lcs(s1, s2)})}{(\mathbf{depth}\ (\mathbf{s1}) + \mathbf{depth}\ (\mathbf{s2}))}$$



```
def wup_similarity(self, other, verbose=False, simulate_root=True):
    """..."""
    need_root = self._needs_root() or other._needs_root()

    ...
    subsumers = self.lowest_common_hypernyms(
        other, simulate_root=simulate_root and need_root, use_min_depth=True
    )

    # If no LCS was found return None
    if len(subsumers) == 0:
        return None

    subsumer = self if self in subsumers else subsumers[0]

    ...
    depth = subsumer.max_depth() + 1

    ...
    len1 = self.shortest_path_distance(
        subsumer, simulate_root=simulate_root and need_root
    )
    len2 = other.shortest_path_distance(
        subsumer, simulate_root=simulate_root and need_root
    )
    if len1 is None or len2 is None:
        return None
    len1 += depth
    len2 += depth
    return (2.0 * depth) / (len1 + len2)
```

Figure 69 Source code of wup similarity stored in nltk Corpus reader

Same process as path and lch similarity when return the wup similarity results. After looking up the synsets of source word and target word in wordnet. With the first word sense from their synsets taken,then we can return the wup similarity value between two words using method synset1.lch_similarity(synset2) to call the equation stored in source code of nltk corpus reader. In case results cannot be returned as expect, we will return a default score 0.34.

```
##wu_pa
def wupSimilar(source, target):
    try:
        word1syn = wordnet.synsets(source)[0]
        word2syn = wordnet.synsets(target)[0]
        wup_value=word1syn.wup_similarity(word2syn)
        return wup_value
    except Exception as e:
        return 0.34
```

Figure 70 Code to return wup similarity value

Information-based Measures

Three information content-based measures which is also a corpus-based measure of specificity a concept, including *res*(Resnik 1995),*lin*(Lin 1998),*jcn*(Jiang & Conrath 1997). A key to the informational similarity of two words is the extent to which they share common information in the IS-A taxonomy. Both *lin* and *jcn* measures apply the information content

of the LCS(least common subsume) of two objects with the sum of each information content of the individual concepts. The difference is the lin measure counts the information content of the LCS by the sum, but Jcn take away the information content of the LCS form this sum and then takes the inverse of it to convert the distance into a similarity measure.

Load an information content file from the wordnet_ic corpus for information-based similarity analysis.

```
from nltk.corpus import wordnet_ic

brown_ic = wordnet_ic.ic('ic-brown.dat')
semcor_ic = wordnet_ic.ic('ic-semcor.dat')
```

Figure 71 Code to load information content file

The *_ic notation is information content. Then we get the IC from two additional corpus, brown and semcor, which under file:nltk_data/corpora/wordnet_ic/ic-brown.dat & ic-semcor.dat. The *ic-brown.dat* and ic-semcor.dat file lists every word existing in the Brown corpus and their information content values which are associated with word frequencies.

Lin Similarity

Lin(1998) similarity use the ic of lcs and the sum of ic of source word and target word to get similarity results with given equation LinSim = 2 * IC(lcs) / (IC(s1) + IC(s2)).
SemCor corpus is set by default in wordnet corpus reader to allow the information content of words derived from.

```
##Lin's Measure
def linMeasure(source, target):
    try:
        word1syn = wordnet.synsets(source)[0]
        word2syn = wordnet.synsets(target)[0]
        #calculate lin similarity using secmor ic
        lin_measurement_value=word1syn.lin_similarity(word2syn, semcor_ic)
        return lin_measurement_value
    except Exception as e:
        return 0.34
```

Figure 72 Code to return Lin similarity value

Resnik similarity

Resnik (1992) measures semantic similarity based on object pairs collected from a corpus, along with their information content on the WordNet taxonomy. It can easily return the similarity of two words using wordnet corpus reader module combining the methods for semantic relevance. Brown corpus is set by default in wordnet corpus reader to allow Resnik method to get the information content value of least common subsume between source and target word.

```
#res Similarity
def resSimilar(source, target):
    try:
        word1syn = wordnet.synsets(source)[0]
        word2syn = wordnet.synsets(target)[0]
        res_sim_value = word1syn.res_similarity(word2syn, brown_ic)
        return res_sim_value
    except Exception as e:
        return 0.34
```

Figure 73 Code to return Resnik similarity value

Jcn Similarity

Jcn(1997) similarity subtracts the ic of the lcs from the sum of the ic of the two concepts to obtain the similarity result. The similarity score returned by the inversed of the distance of information content between two concepts with given equation 1 / (IC(s1) + IC(s2) - 2 * IC(lcs)). Brown corpus is set by default in wordnet corpus reader to allow the information content of words derived from.

```
##Jcn's Similarity
def jcnSimilar(source, target):
    try:
        word1syn = wordnet.synsets(source)[0]
        word2syn = wordnet.synsets(target)[0]
        jcn_sim_value=word1syn.jcn_similarity(word2syn, brown_ic)
        return jcn_sim_value
    except Exception as e:
        return 0.34
```

Figure 74 Code to return Resnik similarity value

	sourceNode	targetNode	path similarity	res similarity	lchScore	wupScore	jcnSimilar	Lin's measure
1	paper	material	0.5	4.692961638951252	2.9444389791664407	0.9230769230769231	0.8602140643861731	0.7837214347837824

Figure 75 Results of example of all methods implemented

4.4 Flask Interface

In the web interface, a new page “word to word sim function” has been created while maintaining the same appearance as the original system, where six newly implemented algorithms will run against the system to obtain results. As the user inputs the source word and target word and sends a run query request, the results of both neo4j and nltk will be displayed in a table together for a visual cross-sectional comparison.

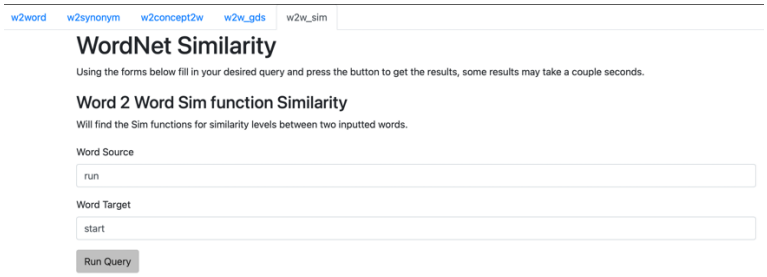


Figure 76 web page of word to word sim function

path similarity	res similarity	lchScore	wupScore	jcnSimilar	Lin's measure	shortestPathSim	dijkstraPathSim
0.09090909090909091	2.036327717785946	1.2396908869280152	0.4444444444444444	0.07636134564104474	0.2792134381256819	0.25	0.25

Figure 77 Table displaying results of nltk and neo4j methods

The queries for the new algorithm use SimilarityMethods.py, while neo4j queries for graphical runs use its original cyphers.py and config.py. Forms continue to use WTForms to pass data from the front-end to the back-end. While the algorithm for neo4j is available for single-input forms, the new implementation only uses dual-input forms for source node and target node.

```python
from wtforms import Form, StringField, SubmitField
from wtforms.validators import DataRequired, Length

class word2input(Form):

    word1 = StringField("Word Source",validators=[DataRequired(), Length(min=1, max=71)])
    word2 = StringField("Word Target",validators=[DataRequired(), Length(min=1, max=71)])

    submit1 = SubmitField('Run Query')


class word1inputs(Form):

    word1 = StringField("Word Source",validators=[DataRequired(), Length(min=1, max=71)])

    submit2 = SubmitField('Run Query')
```

Figure 78 Code of WTForm

```
<div class="form-group">
    {{ w2word_form.word1.label(class="form-label") }}
    {{ w2word_form.word1(class="form-control") }}
</div>
<div class="form-group">
    {{ w2word_form.word2.label(class="form-label") }}
    {{ w2word_form.word2(class="form-control") }}
</div>
{{ w2word_form.submit1(class="btn btn-default") }}
```

Figure 79 Code of form inputs in Flask HTML

For displaying data, the form data is received via app.py, then pass to neo4j to check if the GDS graph exists, and then pass to cyphers.py to return a list of dictionaries containing the shortest path and weighted path outputs, while app.py also pass the form data to SimilarityMethods.py then returns a list of dictionaries containing the nltk results. When no query is entered, the similarity of the words run and start is displayed as the default.

```python
@app.route("/w2w_sim/", methods=['GET', 'POST'])
def w2w_sim():
    print("Similar method")
    # get the form requested
    form = word2input(request.form)

    # request validate
    if request.method == 'POST' and form.validate():
        wordN = form.word1.data
        wordT = form.word2.data
        try:
            print(f"Sim projected graph created: {cyphers.checkExsistsGDSgraph()}")
            # get the results returned from similarityMethods.py
            pathSimilarityScore = pathSimilar(wordN, wordT)
            resSimilarityScore = resSimilar(wordN, wordT)

            lchScore = lchSimilar(wordN, wordT)
            wupScore = wupSimilar(wordN, wordT)
            jcnScore = jcnSimilar(wordN, wordT)

            # print all the similarity results
            print(f"lch score: {lchScore:.2f}, wup score: {wupScore:.2f},path similar score: {pathSimilarityScore:.2f}, res similar score: {resSimilarityScore:.2f},")

            ##information content measure method
            lin_measurement_score = linMeasure(wordN, wordT)
            print("Lin's measure: {:.2f}".format(lin_measurement_score))
```

Figure 80 Code of app.py for w2w_sim page

```python
    # Create a dictionary to gather the results printed
    cypherDict = {}
    cypherDict["sourceNode"] = wordN
    cypherDict["targetNode"] = wordT
    cypherDict["path similarity"] = pathSimilarityScore
    cypherDict["res similarity"] = resSimilarityScore
    cypherDict["lchScore"] = lchScore
    cypherDict["wupScore"] = wupScore
    cypherDict["jcnSimilar"] = jcnScore
    cypherDict["Lin's measure"] = lin_measurement_score

    ##gsd similarity
    gsdCypherOut = cyphers.w2wjacard(wordN, wordT)
    gsdCypherOutDict = gsdCypherOut[0]
    # for key in gsdCypherOutDict:
    #     cypherDict[key] = gsdCypherOutDict[key]

    ##
    cypherRelatedOut = cyphers.wordsFromRelatedConcepts(wordN)
    cypherRelatedOutDict = cypherRelatedOut[0]
    cypherDict["shortestPathSim"] = cypherRelatedOutDict["shortestPathSim"]
    cypherDict["dijkstraPathSim"] = cypherRelatedOutDict["dijkstraPathSim"]

    cypherOut = [cypherDict]
    print(cypherOut)

    # putting all the results above into template
    return render_template("w2w_sim.html", w2word_form=form, cypherOut=cypherOut)
except Exception as e:
    print("error", e)
```

Figure 81 Code of app.py for w2w_sim page

```
# default results into tamplate
print(f"default projected graph created: {cyphers.checkExsistsGDSgraph()}")
cypherOut = [{'sourceNode': 'run', 'targetNode': 'start', 'path similarity': 0.09090909090909091, 'res similarity': 2.036327717785946, 'lchScore': 1.2396908869280152
print(cypherOut)
return render_template("w2w_sim.html", w2word_form=form, cypherOut=cypherOut)
```

Figure 82 Code of app.py for w2w_sim page

For the home page, a query button "word to word sim function" has been added to the original page w2w_sim.

Figure 83 Code of app.py for w2w_sim page

5. Evaluation

Similarity measures

This section will focus on evaluating the different similarity measures as this is the focus of this project rather than usability of the interface design. As all the measures features, drawbacks and how methods differ to each other are already discussed in the background session. In this section, we mainly evaluate methods implemented with neo4j and NLTK from Noun part of speech, as Noun POS is formed in hierarchical structure. A direct approach chosen to evaluate semantic similarity measures is compare the relative between measures and human judged estimation to determine whether the similarity results are reasonable (Pederson 2006). Three pairs of words from Noun are used to compare the results, with each pair scaled from low, intermediate and high level of similarity by human judgement. The more relative the results shown close to human estimation, the more practical the measures are. With these three levels of similarity of data set calculated by eight measures within two kinds of database, it's obvious to indicate which measures is better or worse than any other methods with any purpose.

With Noun POS part, first pair of words chosen (set A) is an obvious unsimilar pair from human judge aspect, "girl" and "program". The second pair of words(set B) is "vehicle" and "bus" with intermediate similar level estimated by human. The third pair of words (set C)

selected is “paper” and “material”, which reflect high is-a semantic connection as paper is a kind of material.

POS: N		Neo4j				NLTK						
						Path-Based Sim				Information-Based Sim		
Human Judgement	SET	ShortestPath Length	Shortest Path Sim	D path length	Dijkstra weighted path Sim	Shortest Path Length	Path[0-1]	Lch[>0]	Wup [0-1]	Res [no range]	Lin[0-1]	Jcn [no range]
low	A: girl& program	7	0.125	7	0.125	12	0.076923	1.072636	0.142857	-0.0	-0.0	0.072910
mid	B: vehicle& bus	5	0.16667	5	0.16667	3	0.25	2.251292	0.823529	5.768923	0.657173	0.215293
high	C: paper& material	3	0.25	3	0.25	1	0.5	2.944439	0.923077	4.692962	0.783721	0.8602141

Figure 84 Summary results of each measures for Noun

Set A

The comparison of the words in pair A yielded the same results for the shorteset path similarity and the Dijkstra shortest path similarity from Neo4j, both with a similarity value of 0.125, reflecting a low similarity, in line with the low similarity expected as human judgement. Also, the three path-based similarities implemented in NLTK yielded pathSim=0.076923, lchSim=1.072636 and lchSim=0.142857, respectively, all of which are at the lower end of their respective interval levels of similarity, which is also in agreement with human judgement's estimate. As can be seen in Figure 87, the latest common subsumer between 'girl' and 'program' is the root node 'Entity'. According to the features of the equations for Resnik and Lin discussed in the Background section, when the root node, which does not cover any information content, is the lcs of the concepts "girl" and "program", Resnik and Lin return 0 as the resnikSim value and linSim value undoubtly. Since the

similarity value of jcn is obtained by subtracting the sum of the respective ic values of "program" and "girl" from their lcs "Entity" root node, and then converted to the reciprocal, the result of jcnSim=0.072910 is obtained, which still reflects a very low level of similarity.

As can be seen through the data set A, all measures conformed to human judgement within their respective intervals and obtained reasonable similarity results, except for the Resnik and Lin methods which could not successfully calculate similarity values due to the limitation of the root node as lcs. Therefore, the Resnik and Lin methods should be avoided when computing two concepts that are far apart. (While neo4j's path similarity both yielded shorter distances and greater similarities than NLTK's path similarity, it is clear that nltk's path similarity is closer to "low" human judgement than neo4j's path similarity.

```
In [31]: from nltk.corpus import wordnet as wn
         wn.synsets("girl")
Out[31]: [Synset('girl.n.01'),
          Synset('female_child.n.01'),
          Synset('daughter.n.01'),
          Synset('girlfriend.n.02'),
          Synset('girl.n.05')]

In [30]: wn.synsets("program")
Out[30]: [Synset('plan.n.01'),
          Synset('program.n.02'),
          Synset('broadcast.n.02'),
          Synset('platform.n.02'),
          Synset('program.n.05'),
          Synset('course_of_study.n.01'),
          Synset('program.n.07'),
          Synset('program.n.08'),
          Synset('program.v.01'),
          Synset('program.v.02')]
```

Figure 85 Sample synsets of "girl" and "program"

```
In [36]: wn.synset('girl.n.01').shortest_path_distance(wn.synset('program.n.01'))
Out[36]: 12
```

Figure 86 NLTK shortest path distance between first element of synset "girl" and "program"

```
In [33]: wn.synset('girl.n.01').lowest_common_hypernyms(wn.synset('program.n.01'))
Out[33]: [Synset('entity.n.01')]
```

Figure 87 NLTK lowest common subsumer between first element of synset "girl" and "program"

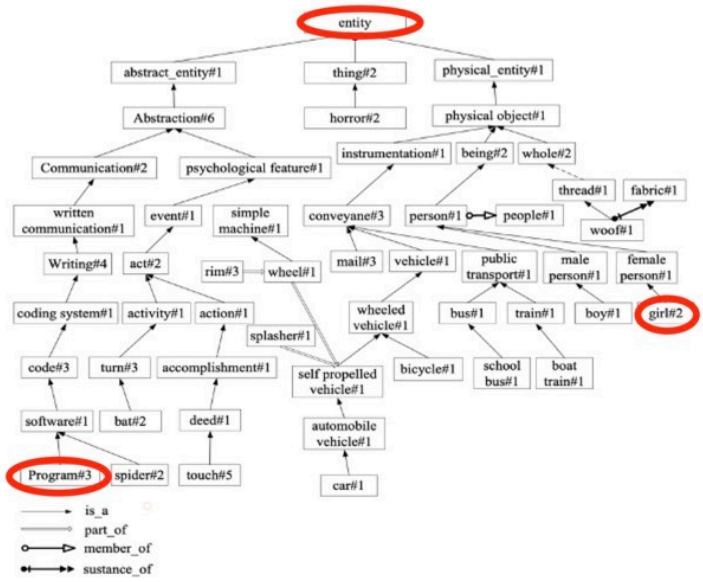


Figure 88 Example of lcs and edges between"girl" and "program"in NLTK

Set B

In the set B vehicle and bus computation, neo4j's shortest Path and Dijkstra weighted path both yielded a shortest path length of 5 and similarity value of 0.16667, demonstrating a low similarity value below the human judgement. The similarity value of 0.16667 is lower than the artificially judged "moderately similar" level. The corresponding path similarity in NLTK gives a shortest distance of 3 and a similarity value of 0.25. According to the formula pathSim = 1.0 / (distance + 1), the range of results is from 0 to 1, with 1 representing the maximum value that can be obtained by comparing the word with itself, and 0.5 representing the maximum similarity between two different concepts, so it is clear that 0.25 here is in line with the 'moderate similarity' human judgement.

Thus nltk's path computes a shorter distance than the path similarity in neo4j, and a higher similarity that is closer to human judgement. This difference can be seen in Figure 89 below. According to the graph model, setB needs to pass through 4 concepts and 5 edges in neo4j's graph, whereas in comparison to Figure 91, only 2 concepts and 3 edges are needed in wordnet's is-a hierarchy, resulting in a higher similarity compared to neo4j's path Sim has a higher similarity. Currently for both setA and setB, the NLTK path similarity values are closer to the similarity judged by humans than neo4j's path Sim.

```
In [48]: wn.synset('vehicle.n.01').shortest_path_distance(wn.synset('bus.n.01'))
Out[48]: 3
```

Figure 89 shortest path distance of vehicle and bus in NLTK

```
In [49]: wn.synset('vehicle.n.01').lowest_common_hypernyms(wn.synset('bus.n.01'))
Out[49]: [Synset('conveyance.n.03')]
```

Figure 90 least common subsumer of vehicle and bus in NLTK

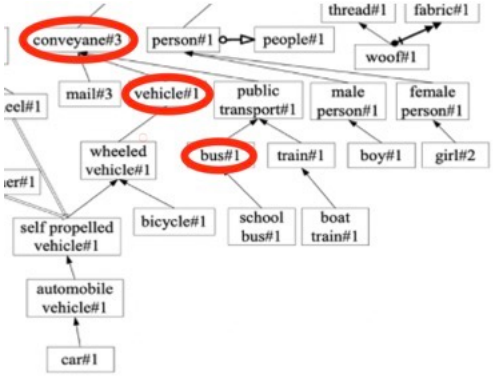


Figure 91 A fragment of is-a hierarchy with vehicle and bus

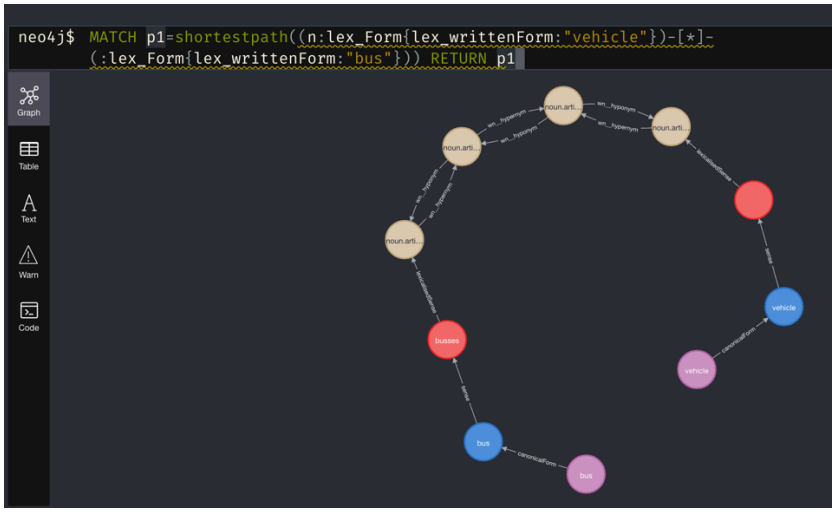


Figure 92 shortest path distance of vehicle and bus in Neo4j

Figure 91 shows that "bus" is the closest hypernyms to the most informative leaf node in the is-a hierarchy, implying that "bus" also has a considerable amount of information. "vehicle" is not as close to a leaf node, but it is a specific concept under the lcs "conveyance" as "bus". The concept of "vehicle" is not as close to a leaf node, but like "bus" it is a specific concept under lcs "conveyance", and also has a rich information content. Since Lin's method takes into account the information content of both lcs " conveyance " and " vehicle " and " bus ", it is obvious that Lin is able to obtain a high similarity result of 0.657173 when all three ic values are at a high level, though it is lower than wup with the same [0-1] range, but this is still in line with the "medium" similarity of human judgement.

The wup similarity value of SetB comes from the depths of lcs "conveyance.n.03 "vehicle.n.01" and ""bus.n.01", which are 6,7,8 respectively. Since nltk adds a fake root node to wordnet in order to connect the verb hierarchical taxonomies of different domains, and the noun hierarchy already has an "entity" root node, this results in an extra edge from the "entity" root to the fake root when we calculate the depth of the noun, so all depth values must be added by one to ensure the accuracy of the result. At this point the depths of lcs, "vehicle" and "bus" are, respectively, 7, 8 and 9, and the result wup similarity value 0.823529 is calculated by applying the formula (2.0 * depth(lcs)) / (depth1 + depth2) , very close to the maximum value of 1.

In the interval 0-1, wup shows a very high similarity compared to both path and Lin. As with the number of edges in the hierarchy, path only takes into account the shortest path of length 3, and is not as close to the human expectation of 'medium similarity' as wup, which takes into account depth. Lin, similar to wup, considers the lcs and the two individual concepts, except that wup counts the number of edges from each of the three concepts to the root node, whereas lin takes the information concept of each of the three concepts and measures the relationship between them. The result is higher for wup than for lin, indicating that the edge counting measure leads to higher results.

```
In [59]: wn.synset("vehicle.n.01").max_depth()
Out[59]: 7
```

Figure 93 max depth of set B vehicle

```
In [60]: wn.synset("bus.n.01").max_depth()
Out[60]: 8
```

Figure 94 max depth of set B bus

```
In [61]: wn.synset("girl.n.01").max_depth()
Out[61]: 9
```

Figure 95 max depth of set A girl

```
In [62]: wn.synset("program.n.01").max_depth()
Out[62]: 6
```

Figure 96 max depth of set A program

Set C

In set C, the similarity of almost all measures increases with the similarity of the human judgement of the concept, in particular nltk's path obtains the highest similarity value of 0.5 between two different concepts, which is fully consistent with the "high similarity" of the human judgement. "From Figure 99, we can see that although in nltk wordnet the distances between“paper” and “material” is only one edge, while in neo4j there are three edges, again verifying that nltk's path similarity is higher than that in neo4j. The similarity in neo4j is more consistent with human judgement.

```
In [140]: wn.synset("paper.n.01").hypernyms()
Out[140]: [Synset('material.n.01')]
```

Figure 97 hypernyms of set C“paper”

```
In [141]: wn.synset('paper.n.01').lowest_common_hypernyms(wn.synset('material.n.01'))
Out[141]: [Synset('material.n.01')]
```

Figure 98 least common subsumer of set C“paper”and “material”

```
In [142]: wn.synset('paper.n.01').shortest_path_distance(wn.synset('material.n.01'))
Out[142]: 1
```

Figure 99 shortest path distance of set C between“paper”and “material”in nltk

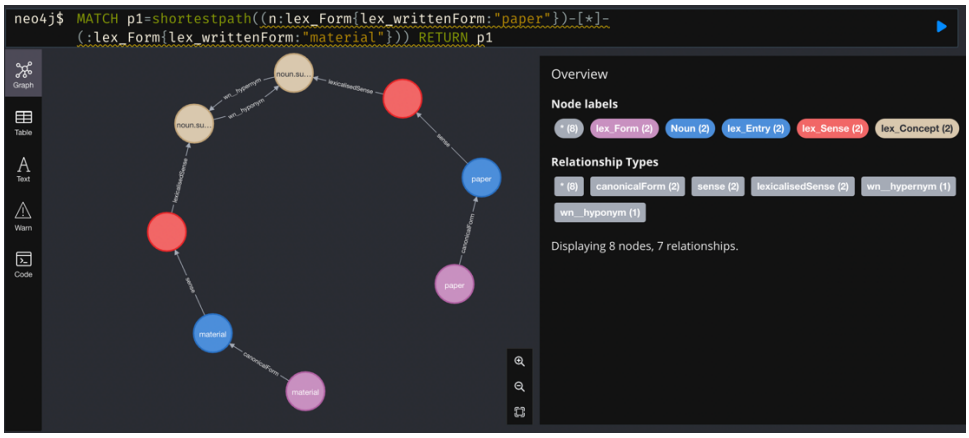


Figure 100 shortest path distance between“paper”and “material”in neo4j

However, when all other methods have reasonable similarity values, the Resnik of set B falls rather than rises. The Resnik sim value of set C is 4.692962, which is lower than the Resnik sim value of set B of 5.768923. In set C, the lcs of "paper" and "material" are "material" itself, while In set B, the lcs of "vehicle" and "bus" are "conveyance", when the ic of "conveyance" is higher than the ic of "material" ,Resnik directly considers the "vehicle" and "bus" of set B to be more similar than set A's "paper" and "material". This is an inaccurate

result according to human judgement and other measures results. Lin and Jcn, which also based on information content for calculation, get very reasonable results as they take into account two more individual concept's ic's in their formulae than Resnik does. Therefore Resnik's calculation based only on the ic of the least common subsumer can affect the accuracy of the similarity, again validating the findings of Hliaoutakis (2006) who using WordNet to compare the semantic similarity of the three methods Resnik,jcn,lin. The information content of the respective concepts considered by Jcn and Lin improves the accuracy of the information content-based similarity measures.

Similarity Evaluation Conclusion

In conclusion, with a longitudinal comparison of the three data sets, the NLTK values demonstrate higher accuracy than neo4j when calculating path-based similarity due to differences in neo4j's graph model and is-a hierarchy. By comparing the least similar set A, it can be concluded that the Resnik and Lin methods cannot effectively compare the similarity values of a pair of concepts that are very far away and whose lcs are root node because of a flaw in the formula and will return 0. Therefore, the Resnik and Lin methods should be avoided when the two words are very far away. A comparison of set B and set C shows that Resnik's calculation of similarity values based only on the lcs of the two concepts is much less accurate than Jcn and Lin's calculation of the ic values of each of the two concepts based on the lcs. Therefore, when using Information to measure similarity, the Jcn and Lin methods are more accurate.

Web interface

The web interface is simple to use and maintains the same minimalist design as the original system. The table presents the new measures results together with previous methods helps user to have an intuitive perception between two different types of data. However, the lack of visualisation of the results still absent. The difference between source word and target word with graph database and the original wordnet is-a hierarchy data is difficult for the user to understand. Last but not least, the web interface does not give the user more control as it only allowing quick query for all similarities at once.

6. Future Work

The goal of this project was to explore the capabilities and features of semantic similarity with WordNet and NLTK and Neo4j in order to create a web program for similarity analysis. Although the newly implemented similarity measures were effective, the focus on algorithms throughout the project resulted in an overly simplistic user interface, with the results of the calculations all being presented in simple tables. This lacks the visualisation of the interaction with the graph database in neo4j and the wordnet in nltk, which would affect the user's understanding of the differences between the results and is a waste of the interaction with the nodes and graphs in neo4j, which lacks interactivity.

I also considered giving the user the option to select a particular similarity algorithm to calculate the similarity results as they prefer. However, in order to avoid complications, I did not implement this feature as I thought it would be more convenient to present the results simultaneously at once for side-by-side comparison and better satisfy the purpose of this project. In future work, I think the focus could be shifted to a user experience-based similarity experience system, where the user is given more control over the selection.

7. Conclusion

This project was built to explore the differences in semantic similarity and the new techniques and capabilities of graph databases and natural language processing tools to handle complex queries. I attempted to implement new similarities on top of an established graph-based similarity query system in order to achieve a comparison between different algorithms and different databases and to provide more insight and understanding of the differences between similarities to scholars interested in natural language processing. Although Neo4j and NLTK were both technologies that required a lot of new learning time for me in the early stages of the project, leading me to abandon refining the user interface and can only focus on the algorithms instead. However, I was able to successfully complete the research, design and analysis of the project, successfully implementing the new algorithms and gaining a more in-depth study and view of the different kinds of similarity algorithms. I think the evaluation part of the project using human judgement and three data sets of varying degrees to compare the differences between similarity algorithms and natural language processing tools and wordnet in relational databases was a good starting point.

8. Reflection and Learning

In the beginning of this project, I was confused by my lack of confidence and fear of challenges when faced the new topics of natural language processing and graphical databases, which I had not been exposed to at all, especially as my project is an improvement on someone else's project. In the early stages of learning of graph databases and various similarity algorithms, I was not clearly aware of the focus on my work. At first I spent a lot of time thinking about how to devise new similarity algorithm formulas, then I worked on how to dynamically visualise the knowledge graphs in neo4j, and finally it took me a long time to decide that the focus of my work is to implement the new similarity algorithm and compare it with the previous one. Looking back at my progress, I think that instead of seeking advice from the supervisor in a timely manner when I encountered difficulties, I repeatedly procrastinated in fear of the project. I also did not plan my time wisely and when an idea did not work, I still spent a lot of time dwelling on it instead of being flexible and considering new ideas. I should try to be more proactive and think of different possibilities without being boxed in by the initial proposal and learn to communicate and ask for help and be brave enough to seek new solutions. Fortunately, I was able to identify my avoidance mentality in the middle stages of the project and changed my mindset in time to start finding a new direction for my work with the help of my supervisor and made substantial progress.
I'm grateful that I managed to have positive progress with this work I afraid of in the beginning. My ability of project management, time management and the way I think is changed. When facing with bigger challenges in the future, I hope to learn from this project and focus on a specific and feasible direction at the beginning, which will greatly improve my confidence and efficiency to complete the project.

9. References

Bird, S., Klein, E. and Loper, E., 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Brown Corpus. 1961. *List of Examples*. http://icame.uib.no/brown/bcm-los.html. [Accessed: 15 September 2022]

Farkiya, A., Saini, P., Sinha, S. and Desai, S., 2015. Natural language processing using NLTK and wordNet. *IJCSIT) Int J Comput Sci Inf Technol*, *6*(6), pp.5465-5454.

Guessoum, D., Miraoui, M. and Tadj, C., 2016, October. A modification of wu and palmer semantic similarity measure. In *The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies* (pp. 42-46).

Global Wordnet Association. *2019. Open Wordnet Documentation(en)* http://globalwordnet.githubio/gwadoc/. [Accessed 16 August 2022]

Hliaoutakis, A., Varelas, G., Voutsakis, E., Petrakis, E.G. and Milios, E., 2006. Information retrieval by semantic similarity. *International journal on semantic Web and information systems (IJSWIS)*, *2*(3), pp.55-73.

Hayes, J., 2022, May. Storing WordNet in a graph database using Neo4j.

Jiang, J.J. and Conrath, D.W., 1997. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*.

Lin, D., 1998, July. An information-theoretic definition of similarity. In *Icml* (Vol. 98, No. 1998, pp. 296-304).

Lord, P.W., Stevens, R.D., Brass, A. and Goble, C.A., 2003. Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. *Bioinformatics*, *19*(10), pp.1275-1283.

Meng, L., Huang, R. and Gu, J., 2014. Measuring semantic similarity of word pairs using path and information content. *International Journal of Future Generation Communication and Networking*, *7*(3), pp.183-194.

Madnani, N., 2007. Getting started on natural language processing with Python. *XRDS: Crossroads, the ACM Magazine for Students*, *13*(4), pp.5-5.

Neo4j Graph Data Science. *Path finding* https://neo4j.com/docs/graph-data-science/current/algorithms/pathfinding/ [Accessed: 21 September 2022]

Neo4j Developer Guides. *What Is a Graph Database*. [no date]. Available at: https://neo4j.com/developer/graph-database/ [Accessed: 2 September 2022].

NLTK Documentation. *Accessing Text Corpora and Lexical Resources*. https://www.nltk.org/book/ch02.html. [Accessed: 13 September 2022]

NLTK Documentation. *Natural Language Toolkit*. https://www.nltk.org/book/ch02.html. [Accessed: 13 September 2022]

NLTK Documentation. *Source code for nltk.corpus.reader.wordnet* https://www.nltk.org/_modules/nltk/corpus/reader/wordnet.html. [Accessed: 13 September 2022]

NLTK Documentation. *Sample usage for wordnet* https://www.nltk.org/_modules/nltk/corpus/reader/wordnet.html. [Accessed: 13 September 2022]

Pedersen, T., Patwardhan, S. and Michelizzi, J., 2004, July. WordNet:: Similarity-Measuring the Relatedness of Concepts. In *AAAI* (Vol. 4, pp. 25-29).

Pedersen, T., Pakhomov, S.V., Patwardhan, S. and Chute, C.G., 2007. Measures of semantic similarity and relatedness in the biomedical domain. *Journal of biomedical informatics*, *40*(3), pp.288-299.

Pedersen, T., 2010, June. Information content measures of semantic similarity perform better without sense-tagged text. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 329-332).

Princeton University.*Princeton Wordnet 3.1 Wordnet RDF* Available:http://wordnet-rdf.princeton.edu/about. [Accessed 23 September 2022].

Rennie, J. 2000. *WordNet::QueryData: module for accessing the WordNet database* http://search.cpan.org/dist/WordNet-QueryData. [Accessed: 21 September 2022]

Seco, N., Veale, T. and Hayes, J., 2004, August. An intrinsic information content metric for semantic similarity in WordNet. In *Ecai* (Vol. 16, p. 1089).

10 Appendix

Relationship	Cost	Definition	Example
also_see	60	a word having a loose semantic relation to another word	learn see also school
antonym	90	an opposite and inherently incompatible word	smart has antonym stupid
attribute	20	an abstraction belonging to or characteristic of an entity	fertile has attributes fecundity
causes	40	Concept A is an entity that produces an effect or is responsible for events or results of Concept B.	kill causes die

Figure 101 Table of WordNet RDF semantic relations, cost, with definitions and examples from GWA

derivation	15	a concept which is a derivationally related form of a given concept.	yearly is the derivation of year
domain_region	65	a concept which is a geographical / cultural domain pointer of a given concept.	England is a domain region of War of the Roses
domain_topic	65	a concept which is the scientific category pointer of a given concept.	computer science is a domain topic of CPU
entails	50	Impose, involve, or imply as a necessary accompaniment or result"	snore entails sleep
exemplifies	80	a concept which is the example of a given concept.	wings exemplifies plural form
has_domain_region	65	a concept which is the term in the geographical / cultural domain of a given concept.	sushi has domain region of Japan
has_domain_topic	65	a concept which is a term in the scientific category of a given concept.	CPU has domain topic of computer science
holo_member	65	Concept B is a member of Concept A	team has member-holonym player
holo_part	65	Y is an amount/piece/portion of X	liquid has holo-portion drop
holo_substance	65	Concept-B is a substance of Concept-A	wood has substance-holonym stick
hypernym	65	a word that is more general than a given word	*animal* is a hypernym of *dog*
hyponym	65	a word that is more specific than a given word	dog has hyponym animal
instance_hypernym	65	the type of an instance	manchester has instance hypernym city
instance_hyponym	65	an occurrence of something	city has instance hyponym manchester
is_exemplified_by	80	a concept which is the type of a given concept.	plural form is exemplified by wings

Figure 102 Table of WordNet RDF semantic relations, cost, with definitions and examples from GWA

mero_member	65	Concept A is a member of Concept B	fleet has member-meronym ship
mero_part	65	concept A is a component of concept B	glove has part-meronym finger
mero_substance	65	Concept A is made of concept B.	stick has substance-meronym wood
pertainym	25	a concept which is of or pertaining to a given concept.	slowly is the pertainym of slow
participle	25	a concept which is a participial adjective derived from a verb expressed by a given concept."	interesting is the participial of interest
similar_to	20	expressing closely related meanings	half-length has near synonym abridged

Figure 103 Table of WordNet RDF semantic relations, cost, with definitions and examples from GWA

Relationship	Cost	Definition	Example
canonicalForm	0	the written representation of a word to its canonical form in its relative part of speech	Open -> Open (v) Open (n) Open (s) Open (a)
sense	0	the unique sense of that canonical form.	Open (v) -> Open (sense (v))
lexicalisedSense	100	the sense of a word lexicalised in to a concept	(sense (v)) -> (concept (v))

Figure 104 Table of lexical relations for WordNet model, with cost, definition, example