Spring 2014

Incorporating WordNet in an Information Retrieval System

Shailesh Padave
San Jose State University

Incorporating WordNet in an Information Retrieval System

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Shailesh Padave

May 2014

The Designated Project Committee Approves the Project Titled

Incorporating WordNet in an Information Retrieval System

by

Shailesh Padave

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2014

Dr. Chris Pollett	Department of Computer Science
Dr. Sami Khuri	Department of Computer Science
Dr. Ronald Mak	Department of Computer Science

ABSTRACT

Incorporating WordNet in an Information Retrieval System

by Shailesh Padave

Query expansion is a method of modifying an initial query to enhance retrieval performance in information retrieval operations [11]. There are alternate ways to expand a user input query such as finding synonyms of words, re-weighting the query, fixing spelling mistakes, etc. [11]. In this project, we created a query rewriting algorithm, which uses synonyms for a given word for query expansion. These synonyms were chosen using WordNet, a lexical database for English [16] [15]. Similarity ranking functions and a part-of-speech tagger were written to extract the essential data from WordNet output. Various experiments were carried out after integrating WordNet in Yioop to evaluate the improvement in search results and its throughput.

ACKNOWLEDGEMENT

I would like to express my deepest appreciation and gratitude to my advisor Dr. Chris Pollett, who mentored me in this project. His advice, useful comments, remarks and engagement through the learning process of the master's project have been priceless. I would also like to thank Dr. Sami Khuri and Dr. Ronald Mak for being committee members and for their precious time and suggestions. I would like to thank my family and friends, who have supported me throughout the entire process.

Table of Contents

List of Figures

List of Tables

1. Introduction

The idea behind this project is to incorporate WordNet in an Information Retrieval System. WordNet was developed at the Princeton University by George A. Miller in the mid-1980s. WordNet is a large lexical database of English which has nouns, verbs, adjectives, and adverbs grouped together into sets of cognitive synonyms. WordNet is a useful tool for computational linguistics and natural language processing due to its structure [16]. In this project, WordNet's output is used in a query re-writing algorithm for query expansion in the Yioop search engine [11].

To retrieve relevant search results, a search engine uses query re-writing algorithms for query expansion. In a query expansion, we reformulate the user's input query using its synonyms. This technique improves the Yioop's search results by ordering web pages according to their relevance. This project will add a new feature in Yioop to make use of WordNet. Various experiments were performed on the WordNet to understand its internal working using command line instructions.

Chapter 2 introduces you to the Yioop search engine and some preliminary work that was done before commencing the project. Chapter 3 includes a brief introduction about the WordNet and its features. It also includes the database information used for WordNet and the different kinds of queries that WordNet supports. Chapter 4 describes how to use part-of-speech tagging for query time and crawl time. It also illustrates the importance of part-of-speech tagging in fetching results from WordNet output. In Chapter 5, various similarity-ranking algorithms are discussed

which are used in this project, e.g., cosine similarity ranking, intersection ranking and Okapi BM25 [9] [10]. The cosine-similarity ranking and intersection ranking methods are used to extract data from the WordNet output. This chapter also briefly describes the importance and effectiveness of ranking methods for an implementation of a WordNet feature in Yioop.

Chapter 6 gives a detailed explanation about the design and implementation of a WordNet feature in the Yioop search engine. Chapter 7 contains a list of experiments that were conducted after adding WordNet feature in Yioop along with the part-of-speech tagging. This chapter also documents the steps to integrate WordNet in Yioop. Chapter 8 contains conclusions about this idea and possible scopes for future work.

2. The Yioop Search Engine

Yioop is an open source, distributed crawler and search engine written in PHP, developed by Dr. Chris Pollett. It is designed in such a way that it allows users to produce indexes of web sites [15]. Until now, query expansion was not present in the Yioop. This project will add a query expansion in the Yioop using similar words from WordNet [11]. This feature will improve Yioop's search results. Yioop can be configured as either a general purpose search engine for the whole web or to provide search results for a set of URLs or domains [15].

WordNet is available on the Princeton University website [16]. To study WordNet's internal structure, we performed many experiments particularly on the command line instructions to understand the use of WordNet in Yioop for a query expansion. Currently, Yioop uses the reciprocal rank fusion score[1] to order web pages in its search results [15].

[1] Reciprocal rank fusion score is an addition of the document rank score, relevance score and proximity score after normalization [15].

3. WordNet

3.1.Introduction to WordNet

The WordNet project was initiated by the Psychology Professor, George Miller, in the 1980's and he was awarded the Antonio Zampolli Prize for his tremendous contribution to WordNet in 2006 [16]. Similar to WordNet, we found WordWeb, Artha, openthesaurus and Moby Thesaurus projects, but many of them worked only on Windows and the databases were not as extensive as the database of WordNet [18]. As WordNet is largely used in many information systems and it has a large lexical database for English, we decided to use WordNet for query expansion, leading to improved Yioop search engine results [19] [16] [11].

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are arranged into sets of conceptual synonyms (synsets), each expressing a distinct concept. WordNet groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets [19]. Synsets are associated by means of conceptual-semantics and lexical relations. For example, fly (noun) can refer to an insect and fly (verb) can refer to an act of moving through the air. The browser helps us to navigate the network of meaningfully related words and concepts. WordNet's structure makes it a useful tool for computational linguistics and natural language processing. WordNet superficially appears like a thesaurus, in which it groups words together based on their meanings [16].

However, there are some important distinctions. First, WordNet interlinks not just word-forms, i.e., strings of letters but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus do not follow any explicit pattern other than meaning similarity [16].

The purpose of WordNet is twofold [19]:

- To produce a combination of dictionary and thesaurus
- To support automatic text analysis and artificial intelligence applications

3.2.WordNet Database Structure

By integrating WordNet in Yioop, Yioop's search results are improved by using the query expansion technique. Similar words from WordNet results are used for query rewriting. For each word, WordNet provides similar words in different senses using its database information. The WordNet database contains information for verbs, nouns, adjectives and adverbs which are stored in a structured database.

Table 1: Types of Files in WordNet [16]

Types of word	Files
Adjective	data.adj , index.adj
Adverb	data.adv, index.adv
Noun	data.noun, index.noun
Verb	data.verb, index.verb

Each type of word is stored in data and index files. All database files follow UNIX naming convention. Index files are index.pos type and data files are data.pos type where "pos" can be [16]:

- Noun
- verb
- adjective
- adverb

For example, if the type of a word is a noun then, information for noun is stored in data.noun and index.noun. A data.noun file contains all the information about words which are noun and index.noun contains the position of each noun word in data file. Due to this, searching for any word became easier and more efficient for the WordNet [16].

An exception list is a list of irregular words which cannot be processed in an algorithmic manner [16]. The format of an entry in each file is

$< irregular\_word, its\_base\_form >$

For example, (corpora, corpus) and (wives, wife). Files in WordNet database are as follows [16]:

- noun.exc
- verb.exc
- adj.exc
- adv.exc

Every file contains *cntlist* which stores count of each word in glossary.

One entry in data.verb which will explain the importance of each field [16].

00048819 29 v 01 habit 0 002 @ 00047662 v 0000 + 03479089 n 0101 01 + 09 00 | put a habit on

- *synset_offset* - Current byte offset in the file (8 digit)
- *lex_filenum – (2 digit)* lexicographer file name containing the synset
- *ss_type – n,v,a,s,r*
- *w_cnt – number of words in synset (2 digit HEX)*
- *word – Actual search word*
- *gloss – represented as vertical bar followed by text string. May contain 1 or more examples*

Information about data.verb [16]

castilian n 1 1 @ 1 0 06979859

- *lemma – lower case word*
- *pos – n v a r*
- *synset_cnt – number of synset in that lemma*
- *p_cnt – number of pointers*

Here are some statistics for a WordNet database [16]:

Table 2: WordNet Database Statistics [16]

Part of speech	Unique String	Synset	Total Word-Sense pair
Noun	117798	82115	146312
Verb	11529	13767	25047
Adjective	21479	18156	30002
Adverb	4481	3621	5580
Total	155287	117659	206941

3.3.Search Result of WordNet

After installing WordNet, use wn.exe to run WordNet.

Here is the output of WordNet:

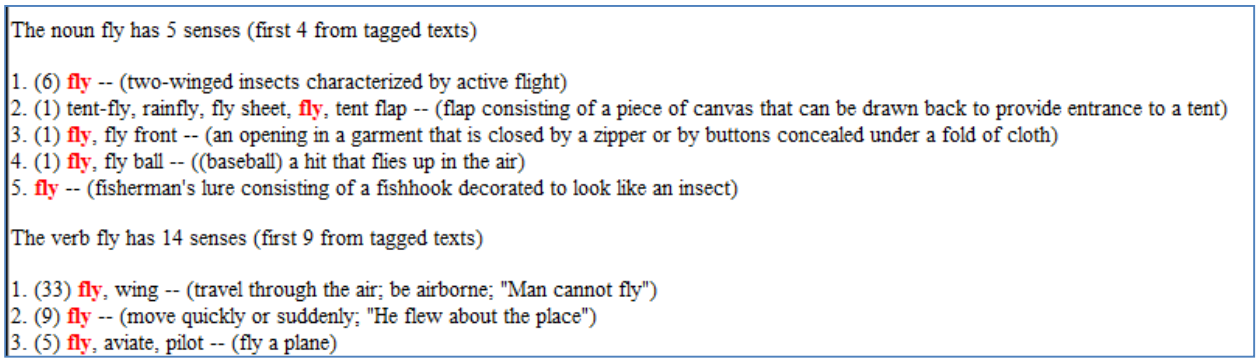

Figure 1: WordNet Output from GUI

As shown in Figure1, the WordNet represents a given word in four senses - noun, verb, adjective, and adverb. This WordNet output is used by a query rewriting algorithm in query expansion [11].

Each sense contains similar words followed by the use of that word in a sentence. We can obtain the similar output from command line.

Command line output for the input word - *fly* is shown in Figure 2:

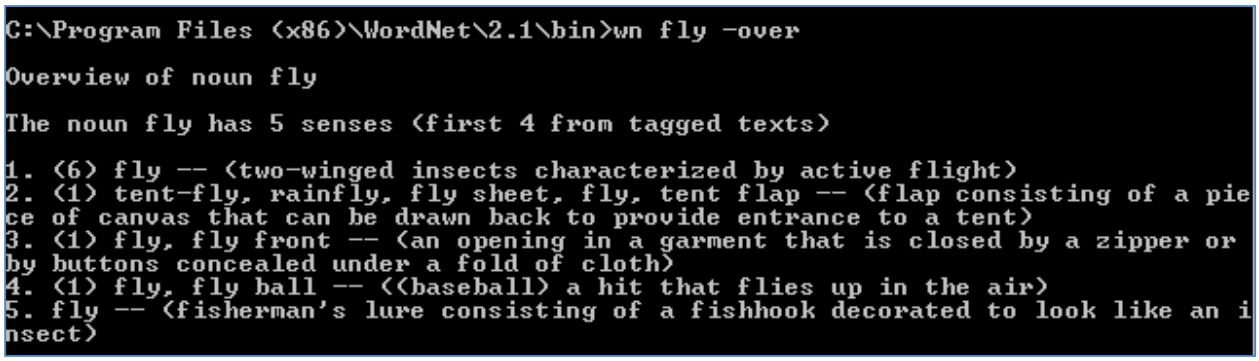

Figure 2: Search Result Output from WordNet using command line

The following command is used to get the WordNet output:

wn <input_word> -over

This command outputs all the similar words from four different senses. Different command line arguments provide the information about hypernyms, hyponyms etc. Hypernyms can be defined as Y is a hypernym of X if every X is a kind of Y, for example, canine is a hypernym of dog. Hyponyms are defined as Y is a hyponym of X if every Y is a kind of X, for example, dog is a hyponym of canine [19].

3.4.Applications of WordNet

WordNet has been used for a number of different purposes in information systems, such as word sense disambiguation, information retrieval, automatic text classification, automatic text summarization, machine translation and even automatic crossword puzzle generation [19].

4. Part-Of-Speech Tagging

In this project, a part-of-speech tagger is used in between the Yioop and the WordNet layer. Part-of-speech tagging is a process of marking up or tagging a word in a text or a sentence or a corpus corresponding to a particular part-of-speech based on its definition and context [14]. Part-of-speech tags for a word in a phrase, a paragraph or a sentence are decided by its relationship with the adjacent word. A part-of-speech tagging is also known as POS tagging or POST. A part-of-speech tagging is mainly used by corpus linguistic for grammatical tagging and word category disambiguation [14].

In this process, we identify the words as noun, verb, adjective and adverb. Part-of-speech tagging is divided into two distinguished groups, which are rule- based part-of-speech tagging and stochastic part-of-speech tagging [14].

Rule-based part-of-speech tagging is the oldest approach which uses hand written rules for tagging and it depends on a dictionary or a lexicon to retrieve suitable tags for the tagging process [12]. A bigger lexicon will give us better results but it will also cost more processing speed [3]. When a word has more than one tag, the correct tag is identified by hand written rules. Linguistic features of a word, such as its preceding word, its following word, and some other aspects help the tagger to analyze disambiguity. For example, if you want to do part-of-speech tagging for a word, and if its previous word is an article then a part-of-speech tagging for the word must be a noun

[12]. If a word ends with 'ed' then it is a past participle and a word is an adverb if it ends with 'ly' [3].

Let the sentence under the test be

"*We systematically analyze the performance of these techniques versus existing search results"*

After part-of-speech tagging, output will be

We~NN systematically~AV analyze~NN the~DT performance~NN of~IN these~DT techniques~NN versus~IN existing~VB search~NN results~NN

It is really difficult to completely automate part-of-speech tagging. Sometimes humans have difficulty finding a possible interpretation for a given sentence. In such cases, a corpus is used which is a set of documents tagged manually for its respective part-of-speech. A full list of common tags from the Brown corpus is used for tagging purpose [3]. The Brown corpus contains 500 samples of English-language text with roughly one million words [5]. We used an implementation based on the Brill tagger, which was described by Eric Brill in his 1995 thesis. It is also known as "an event-driven transformation-based tagger"[4]. It is widely used for English word processing [14].

Some examples of tags from the Brown corpus are as follows [5]:

Table 3: Tags and its Definition from Brown Corpus [5]

Tag	Definition
NN	Singular or mass Noun
NNS	Plural Noun
DT	Singular determiner/quantifier
JJ	Adjective
RB	Adverb
IN	Preposition

As shown in the WordNet output, it is divided into four different senses such as noun, verb, adjective and adverb for a given word. Instead of using the whole output, we implemented a part-of-speech tagging as a filtering technique to extract the relevant part. A part-of-speech tag is obtained for each word from user's input query. Then eventually from WordNet, we will fetch relevant information only for that sense.

For instance, consider a query in the search engine, as '*running dog*' then in the first step, it will use a part-of-speech tagger and generate output as '*running~VB dog~NN'*. If an input query has more than a word then WordNet will process word by word. Now if we search *running* in WordNet, its output will provide 5 senses for noun, 41 senses for verb, and 7 senses for adjective as shown in Figure 3 on the next page.

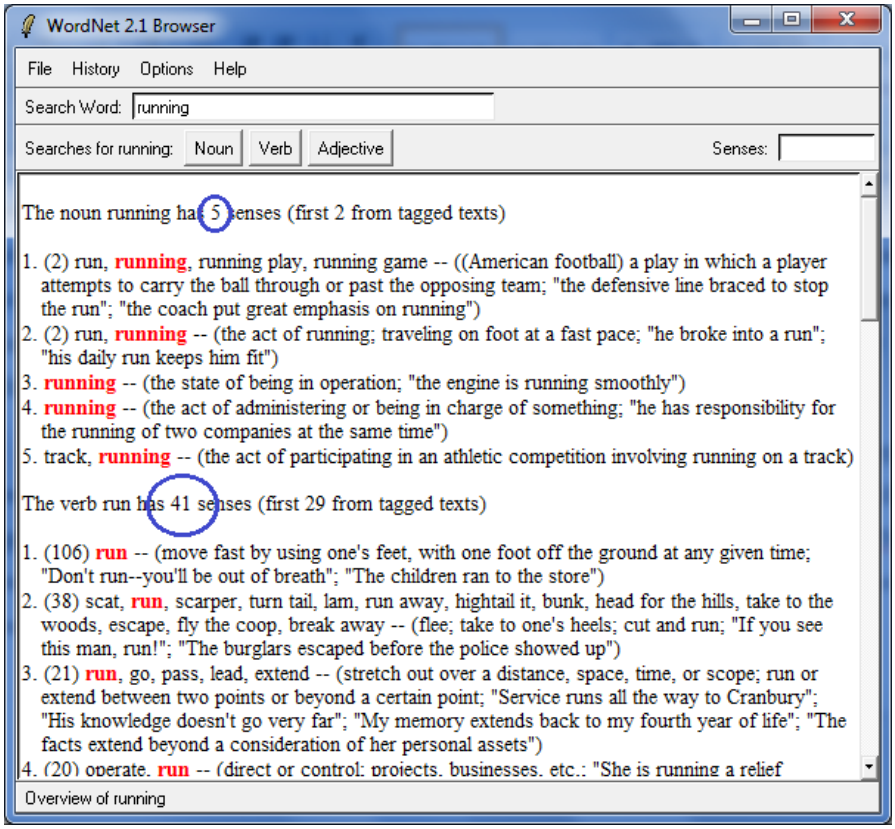

Figure 3: WordNet Output

If we consider the total output, we need to process all 53 senses, which is not efficient. But a part-of-speech tagger will make it efficient as it tags an input word *running* as verb, we will select only verb senses from WordNet output for query expansion. Chapter 6.1 contains step by step instructions to extract the similar words from WordNet output. Part-of-speech tagging gives us an efficient and effective way to extract relevant data from WordNet output.

In the absence of a part-of-speech tagger, we can use each and every sense from the WordNet output, however, it may affect efficiency.

5. Similarity Ranking Algorithms

Similarity ranking algorithms are used to find the similarity between two different sentences [6]. The WordNet output provides an array of sentences and a query term is provided by the Yioop search engine, we can rank these sentences from an array on the basis of similarity measures. In an information retrieval system, we often use a similarity measure to rank the documents. Some of them are cosine ranking, Okapi BM25, intersection ranking, Euclidean distance, etc. [9] [10].

In our implementation, we used cosine similarity ranking, intersection ranking and BM25. More information about the implementation can be found in Chapter 6.

5.1.Cosine Similarity Ranking Algorithms

The cosine similarity is a measure of a similarity between two vectors of an inner product space that measures cosine of the angle between them [17]. Since it is a judgment of orientation, if two vectors have the same orientation, cosine will be 0° then cosine similarity between two vectors will be 1. If two vectors are at 90° then similarity will be 0. The cosine is less than 1 for any other angle between 0° to 90°. When two vectors have diametrically opposite orientation, then a similarity will be -1. Cosine similarity is independent of their magnitude. Practically, cosine similarity should be in positive space, i.e., the outcome should be neatly bounded to a range of 0 to 1. High-dimensional positive spaces are most suitable for cosine similarity [17].

5.1..1. TF (Term Frequency) & IDF (Inverse Document Frequency)

Term Frequency is a measure to find how common the term is in the given document from corpus [6].

Term Frequency is formulated as [6]:

$$TF = \log(f_{t,d}) + 1 \quad if\ f_{t,d} > 0\ \&\ 0\ otherwise$$

where, $f_{t,d}\ is\ frequency\ of\ term\ t\ in\ document\ d$

Inverse Document Frequency is the relationship between document frequencies to total number of documents in a corpus [6].

Inverse Document Frequency is formulated as [6]:

$$IDF = \log\left(\frac{N}{N_t}\right)$$

where, $N\ is\ total\ number\ of\ document\ in\ corpus$

$N_t\ is\ total\ number\ of\ document\ in\ corpus\ containing\ term\ t$

Consider two |V|- dimensional vectors, $\vec{x}$ and $\vec{y}$.

$\vec{x} = (x_1, x_2, x_3, x_4, \dots\dots, x_{|v|})$ and $\vec{y} = (y_1, y_2, y_3, y_4, \dots\dots, y_{|v|}$

$\vec{x}$ is for a query vector and $\vec{y}$ will be for a document vector. We will take their dot product to measure closeness between query and document [6].

The length of a vector is computed by the Euclidean distance formula [6]

$$|\vec{x}| = \sqrt{\sum_{i=1}^{|v|} x_i . y_i}$$

According to linear algebra, a dot product is calculated using the following formula [6]:

$$\vec{x}.\vec{y} = \sum_{i-1}^{|v|} x_i.y_i$$

The above equation is defined by the following geometric meaning as [6]

$$\vec{x}.\vec{y} = |\vec{x}||\vec{y}|\cos\theta$$

where, θ = angle between $\vec{x}$and $\vec{y}$.

The length of the vector $|\vec{v}|$ is calculate as follows [6]:

$$|\vec{v}| = \sqrt{\sum_{i=1}^{|V|} v_i^2}$$

To calculate the angle, we have the following equation [6]:

$$\cos\theta = \frac{\sum_{i-1}^{V} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{|V|} x^2}\sqrt{\sum_{i=1}^{|V|} y^2}}$$

when $\theta = 0°, \cos\theta = 1$ then the two vectors are collinear and

if $\theta = 90°, \cos\theta = 0$ then the two vectors are orthogonal.

Consider we have a query vector $\overrightarrow{q}$ and document vector $\overrightarrow{d}$, then the similarity is defined as the cosine of the angle between them [6]. i.e.

$$sim(\overrightarrow{d},\overrightarrow{q}) = \frac{\overrightarrow{d}}{|\overrightarrow{d}|}.\frac{\overrightarrow{q}}{|\overrightarrow{q}|}$$

We will not get negative values for cosine similarity, as we are using only positive components of the vector [6].

5.2.Intersection Ranking Algorithms

This is another technique used to calculate a similarity between two statements. We split both sentences into an array of words, also known as tokens, and then we count the number of common tokens in between them. An average length of two sentences is used to normalize the score [1].

The Intersection ranking is computed as follows [1]:

$$f(s_1, s_2) = \frac{|\{w|w\ in\ s_1 \& \ w\ in\ s_2\}|}{(|s_1| + |s_2|)/2}$$

where $|s_1|\ and\ |s_2|\ is\ the\ length\ of\ documents\ s_1 and\ s_2\ respectively$ [1]

5.3.BM25 Ranking Algorithms

A search engine determines the set of matching documents from a corpus to a given user query by using boolean interpretation [6]. Consider a query as

$$Q = (\text{"san"}, \text{"}jose\text{"}, \text{"}computer\text{"})$$

Conjunctive Boolean query for Q is

"san" AND "jose" AND "$computer$"

Disjunctive Boolean query for Q is

"san" OR "jose" OR "$computer$"

Then a search engine will retrieve the results using a conjunctive and a disjunctive query.

A conjunctive retrieval method is faster than a disjunctive retrieval method since few documents will be scored and ranked in case of disjunctive retrieval method [6].

Okapi BM25 is one of the similarity ranking functions used in an information retrieval by search engines [6] [13]. When we provide a query to the search engine, it will return the list of relevant documents. To rank these relevant documents, Okapi BM25 ranking function is used. Stephen E. Robertson, Karen Spärck Jones, and others developed this algorithm in 1970 and 1980. This ranking function is actually named as BM25, but it is known as Okapi BM25 because the Okapi information retrieval system was the first one to implement this ranking function. The Okapi information retrieval system was implemented at London's City University in the 1980s [13].

BM25 is a retrieval function for a bag-of-words i.e. multi set of words, regardless of the grammar rules [13] [2]. BM25 will rank a set of documents depending upon the appearance of the query terms in each document. This ranking function is independent of relative proximity of the query words in a document [13].

For the calculation of BM25, we use TF (Term Frequency) and IDF (Inverse Document Frequency) functions [6].

The Term Frequency (TF) is calculated using following formula [6]

$$TF_{BM25} = \frac{f_{t,d}.(k_1 + 1)}{f_{t,d} + k_1.\left((1 - b) + b.\left(\frac{l_d}{l_{avg}}\right)\right)}$$

where, $f_{t,d} = frequency\ of\ query\ term\ t\ in\ document\ d$

$k_1,\ b = free\ parameters\ usually\ set\ to\ 1.2\ and\ 0.75\ resepctively$

$l_d = the\ length\ of\ a\ document\ d$

$l_{avg} = \ average\ length\ of\ documents\ in\ a\ corpus$

$Variable\ b\ is\ used\ to\ control\ the\ degree\ of\ normalization.$

The Inverse Document Frequency (IDF) is calculated by using the following formula [6]:

$$IDF(t) = \log\left(\frac{N}{N_t}\right)$$

where $N = total\ number\ of\ documents\ in\ corpus$

$N_t = total\ number\ of\ documents\ in\ which\ term\ t\ occurs$

The BM25 scoring function is defined as [6]:

$$Score_{BM25}(q,d) = \sum_{t \in q} IDF(t).TF_{BM25}(t,d)$$

Consider we have a query Q with n words such as $q_{1,}q_{2,}q_{3,}..,q_n$, for each query word, term frequency and inverse document frequency will be calculated using the above formula and the score will be generated using BM25. Once we get the scores for each document, we can arrange the documents in an increasing order of BM25 score.

Once we get the top two similar words by processing the WordNet output, we use them as query words in a query expansion. Yioop will return the relevant documents to a given query from the corpus of documents. Then we can use the BM25 scores to rearrange them if we get similar words for a given query. More explanation on the implementation is provided in Chapter 6.

6. Design & Implementation

This chapter briefly describes an implementation of the WordNet feature in the Yioop using information from previous chapters. We used the Yioop as a base platform to incorporate the WordNet in a search engine. The aim of using WordNet is to get similar words for a given query and rewrite the input query. These rewritten queries using similar words are used to expand a query [11]. A sequence diagram for integrating WordNet in Yioop is shown in Figure 4 on next page.

When we enter a query in the WordNet, the software provides an output in four different senses such as noun, verb, adverb, and adjective. The output is as shown in Figure 1 from Chapter 3.3. There are no PHP API's (Application Programming Interface) provided for WordNet [16]. We need to find another way to integrate WordNet in Yioop. In a preliminary work, we found out that WordNet supports the command line instructions.

The instruction is as follows:

$$wn < queryWord > over$$

Output of this command is the same as the output obtained from the GUI (Graphical User Interface). To run this command, a user needs to reach the binary folder of WordNet from the installation directory or the user can set WNHOME as an environmental variable.

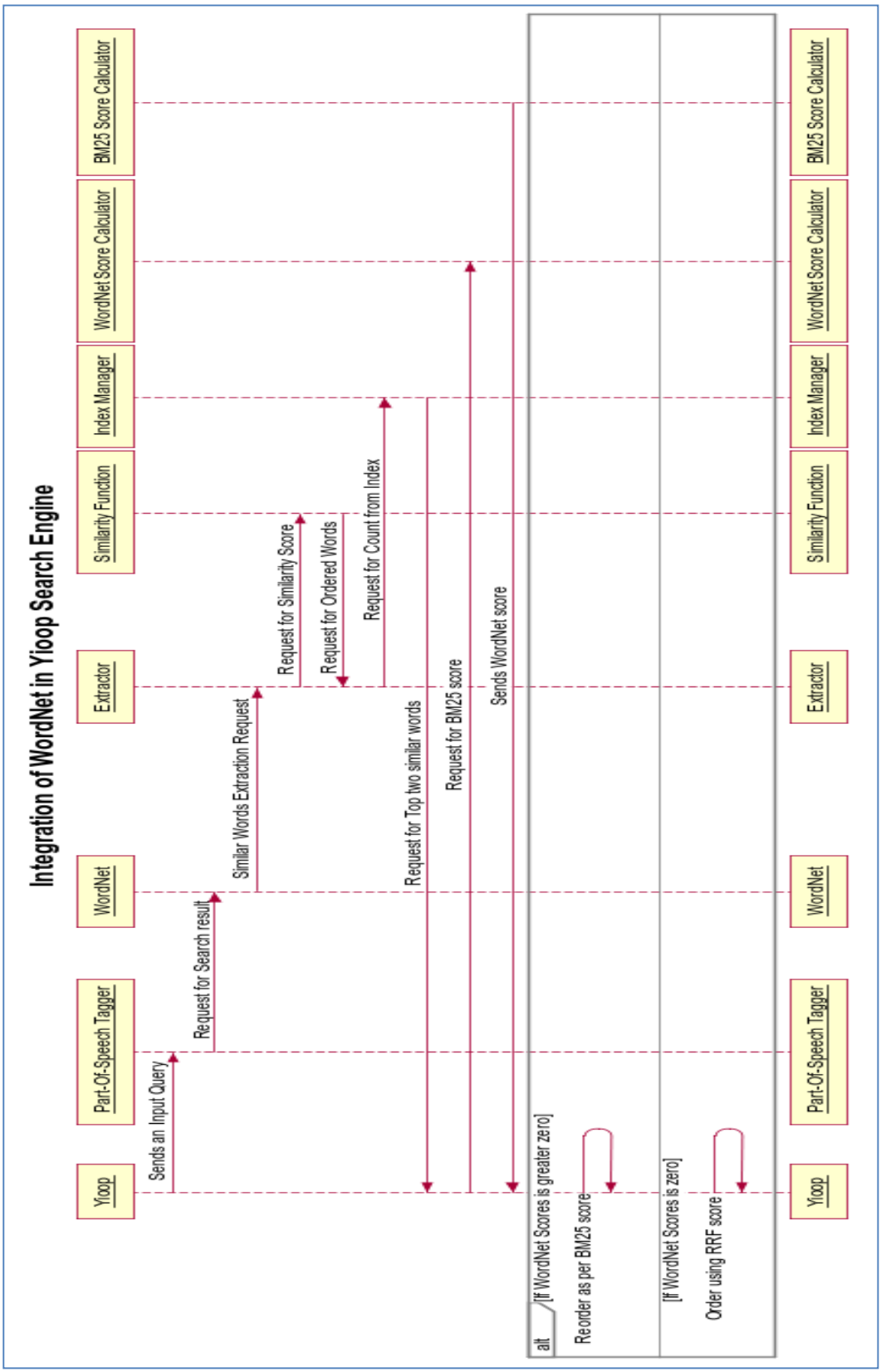

Figure 4: Sequence Diagram for Integration of WordNet in Yioop

We will discuss more about the WordNet result format. The results for each input word is written in a standard output format. Every output contains a description for the similar word in one line. In the output, we get an overview of all senses of a search word in all syntactic categories. Each result starts with synset offset, followed by its synonyms, i.e., similar words, followed by an elaborated meaning of the synonym, and in a further part, we can see the usage of that synonym in a sentence or sentences. These sentences are used to find out the exact synonym using similarity ranking methods.

We will discuss the importance of WordNet output from query expansion point of view. When a user enters a query in Yioop, we will use that query as input for part-of-speech tagger. A part-of-speech tagger will process the input and will return a tagged result. Let us consider an example with the input query 'running dog'.

A part-of-speech tagger will provide the following tagged output:

running~VB dog~NN

We are interested in four types such as noun, verb, adjective and adverb. So if we get any other tag than these four tags, we will ignore them. Now, we know in which syntactical category we need to concentrate for further processing from WordNet output.

We input this query to WordNet, one word at a time from the query. For a given word, WordNet provides a search result from its database. The result is written to a text file named 'wordnet_output.txt'. We read this output file and store all the data in a variable. Now we need some extraction process to extract similar words for us from the search result. If we take a closer look at the WordNet output, we can see that there is a common format. We use regular expressions to extract all the similar words from the tagged sense. In this step, we filter a search result to its corresponding part-of-speech. We need to find out similar words along with their use in sentence or sentences. We store all similar words as all the similar words are placed before '--' sign in the WordNet output. Sentences are extracted from WordNet result using regular expression as all the sentences are stored in between '("<sentence>")'.

After extraction of similar words and their sentences in an array as per their part-of-speech tag used in user query, we need to find an exact similar word for a given query word. Now here similarity ranking functions will play a major role. To extract similar words using their respective sentences, we use the cosine similarity ranking function. Cosine similarity ranking is carried out on two input two sentences. One sentence is the input query from user and the other sentence is from WordNet output.

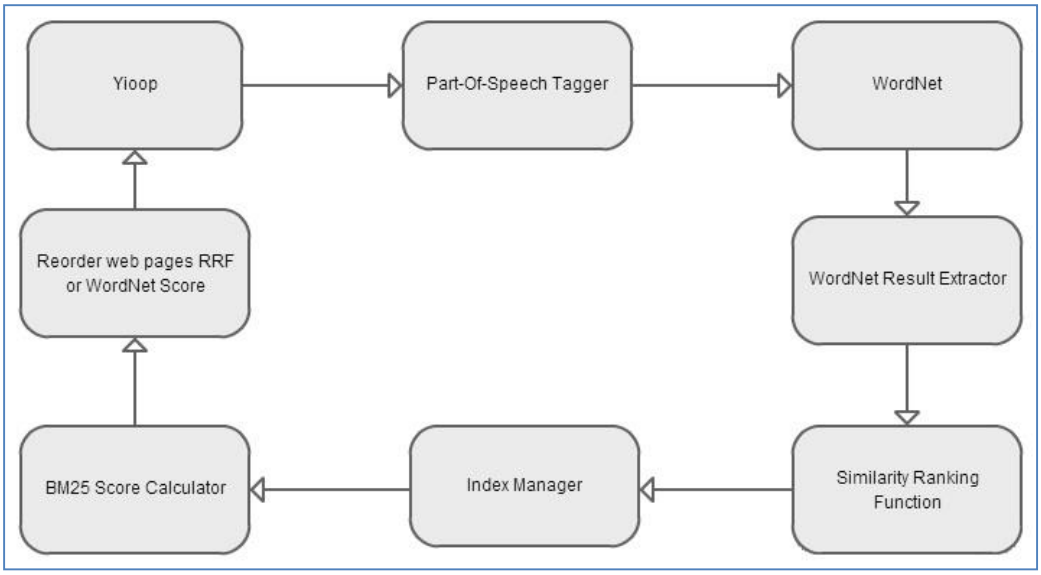

Figure 5: Block Diagram of Integrating WordNet in Yioop

After performing the cosine similarity ranking, we can reorder the arrays for similar words according to their cosine similarity score. We repeat the same process for the next word from a given input query. As per the example, we get similar words for 'running' and 'dog'.

If we get four similar words for 'running' and three similar words for 'dog', then we will see all possible combinations for a given query so we can generate twelve combinations for it. We cannot use all generated combinations for further processing. Therefore, we will make use of statistics from an inverted index.

An inverted index is a mapping between terms and their location in a corpus, i.e., a text collection. Every inverted index has two main components: a dictionary and a posting list. The dictionary lists the terms from a collection vocabulary. A posting list contains the position of each word in a collection [6].

Now for further filtration, we can use an inverted index. We get the frequency for each combination from previous processing. The index manager returns a count for each combination. We sort them according to the frequency of each combination in an inverted index. As the top two combinations have higher frequencies in the inverted index, it means more documents have that combination, so we select the top two combinations. These two words will be used for reordering of documents from the search result. For reordering, we use the BM25 score.

While crawling web pages, a summary is generated for a page and it is stored in a database. This summary contains the most important and frequent words from the web page. This summary for a webpage itself will describe the content of that web page. So for BM25, we will use input as an array of summaries from the web pages and two similar words we extracted from the WordNet result. To normalize the BM25 score, for two input words from WordNet result, we multiply the first score by 2/3 and the second score will be divided by 1/3. As the first word is more relevant than the second one, we gave higher preference for the first BM25 score before adding.

This is a very effective way of calculating the similarity ranking. The search results will be reordered by the BM25 score. If the BM25 score for documents is zero, the search results from Yioop will be ordered by the reciprocal rank fusion score.

The WordNet results will be displayed on the Yioop search result as shown in Figure 6. Under the WordNet Results we will show the top two words used by Yioop in query expansion.

Figure 6: Similar Words Returned By WordNet

Also you can see the BM25 score for those two words for a document on the Yioop screen under the Score tab.

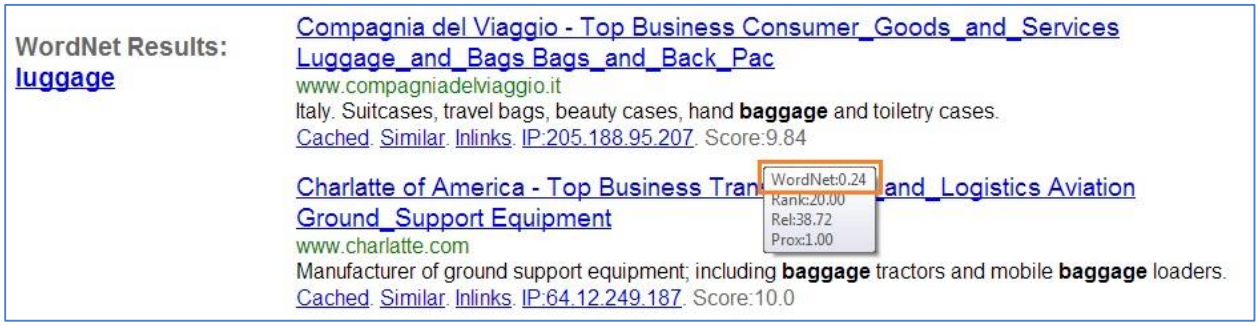

Figure 7: BM25 Score Listed as WordNet in Score

6.1.How to Use the WordNet Output

In the previous chapters, we learned about WordNet's output format. If in the future we want to replace WordNet by any other dictionary for a given word in any other language, then the implementation is flexible enough to adopt new dictionary software.

Here are some important parameters the software output must follow:

1. Dictionary output should contain similar words for a given word in different syntactical categories as noun, verb, adverb and adjective.
2. In every syntactical category, similar words should be placed before '--' and should be followed by use of those similar words in the sentence closed in "-".

   The generalized format for the WordNet output is as follows:

   $The\ noun < search - word > has\ n\ senses$:
   1. $similarWord1, similarWord2, \ldots\ldots$
   $--(meaning\ of\ similar\ word,$ Use of similar word in sentence)
   2. $similarWord1, similarWord2, \ldots\ldots$
   $--(meaning\ of\ similar\ word,$ Use of similar word in sentence)

   $The\ verb < search - word > has\ n\ senses$:
   1. $similarWord1, similarWord2, \ldots\ldots$
   $--(meaning\ of\ similar\ word,$ Use of similar word in sentence)
   2. $similarWord1, similarWord2, \ldots\ldots$
   $--(meaning\ of\ similar\ word, "Use\ of\ similar\ word\ in\ sentence")$

   $The\ adv < search - word > has\ n\ senses$:

1. $similarWord1, similarWord2, \ldots\ldots$

$--(meaning\ of\ similar\ word,$ Use of similar word in sentence)

2. $similarWord1, similarWord2, \ldots\ldots$

$--(meaning\ of\ similar\ word,$ Use of similar word in sentence)

$The\ adj < search-word > has\ n\ senses$:

1. $similarWord1, similarWord2, \ldots\ldots$

$--(meaning\ of\ similar\ word,$ Use of similar word in sentence)

2. $similarWord1, similarWord2, \ldots\ldots$

$--(meaning\ of\ similar\ word, "Use\ of\ similar\ word\ in\ sentence")$

For the above output format, we used regular expressions to extract similar words from search results according to their tags provided by a part-of-speech tagger. We used the cosine similarity ranking and intersection ranking function to calculate the similarity between a user input query and the sentences from the WordNet output.

6.2.Importance of Ranking Algorithms

Three ranking algorithms are used in this project, Cosine similarity ranking, Okapi BM25 and Intersection ranking.

1. After retrieving the search result from WordNet, Cosine similarity ranking found out exact similar words from n number of senses.

2. The intersection method is used along with Cosine similarity ranking. If the cosine similarity function returns a zero similarity score, then the intersection ranking function will be used to get the score.

3. Okapi BM25 is used to rearrange retrieved web pages in Yioop's search result. As per BM25 ranking function, if the frequency of a query word is higher in a web page summary then that web page is more relevant than other web pages. Yioop uses reciprocal rank fusion to score the web pages.

If WordNet returns no results from search results then all the web pages in the search result will be ordered by RRF (Reciprocal Rank Fusion). Reciprocal Rank Fusion is an addition of the Doc Rank score, Relevance score and Proximity score after normalization [15].

6.3.Steps to Add WordNet Feature in Yioop

WordNet is a standalone application [16]. It is a form of dictionary which will give search results for a given word in different syntactical parts as noun, verb, adjective and adverb. There is no specific API (Application Programming Interface) provided for the PHP language [16]. So we use the features from this tool using command line arguments. Once we get similar words from WordNet, we can use query expansion technique by query rewriting with those similar words returned by WordNet [11].

Here are the steps to add WordNet as feature in a Yioop search engine:

1. Download WordNet from the WordNet website

   For Ubuntu User: $sudo\ apt - get\ install\ wordnet$

   For Mac User: $brew\ install\ wordnet$

   Sometimes Linux users may need to install science-linguistics as it is dependency for WordNet.

For science-linguistics dependency: $sudo\ apt - get\ install\ science - linguistics$

2. Open the Yioop search engine and login as 'root' user.

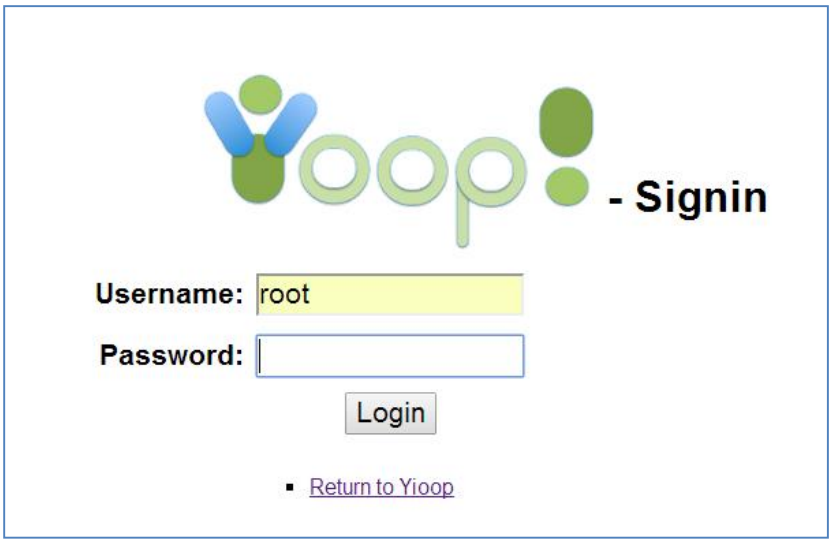

Figure 8: Yioop Login Page

3. Then on the left side of the screen, you can see different options. Select '**Page Options**' from **Crawls**.

Figure 9: Page Option from Yioop Admin web page

4. After selecting Page Options, you can see three different tabs on right hand side of the web page such as Crawl Time, Search Time, and Test Options. Out of these tabs, select the "**Search Time**" tab.

Figure 10: Search Time Tab

5. Once you select the Search time tab, you can see different options under **Search Page Element and Links** such as Word Suggest, IP Address, and WordNet etc. To use the **WordNet**, the user needs to check that option as shown in Figure 11.

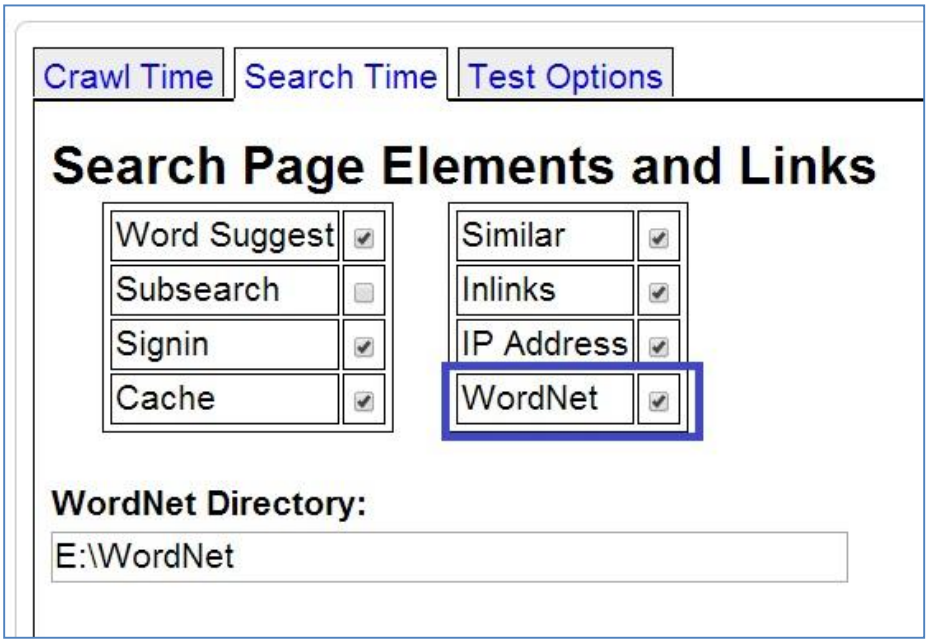

Figure 11: Search Page Elements and Links

6. The Windows user needs to provide a path of the WordNet directory but it is not required for the Ubuntu user.

7. Experiments

From an experimental point of view, we used a part-of-speech tagger with crawl time as well as query time. This experiment helped to decide whether or not the use of part-of-speech tagging with crawl time, i.e., at the time of creating the inverted index, will improve Yioop's search results. While crawling the webpage, the crawler requests the web page for information and generates the summary along with the inverted index. We tested it by including part-of-speech tagging before index creation. Experimental work was carried out on the original Yioop crawling index and index generated after part-of-speech tagging.

The experiments were carried out on different datasets as follows:

Table 4: Number of Web Pages Crawled

Name of Website	Number of crawled pages
SJSU CS	18156
Wikipedia dataset	100,000
dmoz dataset	972800

The effectiveness of the retrieved method is measured by the relevance provided by human assessment. When a search engine provides a search result, and the user visits any website listed in search result, and then backtracks quickly to search results then that document is no more relevant to user search query [6]. So to measure the effectiveness of search engine retrieval method, we use two standard measures named as Recall & Precision.

7.1.Recall and Precision

To calculate recall and precision, we need two input parameters named as [6]

$$Res = Set\ of\ documents\ which\ are\ returned\ by\ query\ q$$

$$Rel = Set\ of\ relevant\ documents\ available\ in\ the\ collection\ for\ the\ topic$$

Recall is computed as [6]:

$$Recall = \frac{|Rel \cap Res|}{|Rel|}$$

Precision is computed as [6]:

$$Recall = \frac{|Rel \cap Res|}{|Res|}$$

Recall means the fraction of relevant documents which appear in the result set. It is the measure of number of relevant documents returned by search engine from corpus [6].

Precision is defined as the fraction of the result set which is relevant. This measures how many documents in the result set are relevant to input query or topic [6].

On the Yioop search engine, under setting option, the user has a choice to select the number of web pages to be shown on a search result as 10, 20, 50, and 100. So recall and precision also can be calculated on the first 10, 20, 50, 100 results. For k pages, recall and precision measures are defined as recall@k and P@k i.e. precision@k respectively.

Recall-precision values in Yioop crawling without part-of-speech tagging are as follows:

Table 5: Recall and Precision in Yioop Crawling without Part-Of-Speech Tagging

Recall	Precision
0.82	0.12
0.66	0.25
0.61	0.43
0.45	0.62
0.21	0.76

The graphical representation is as follows:

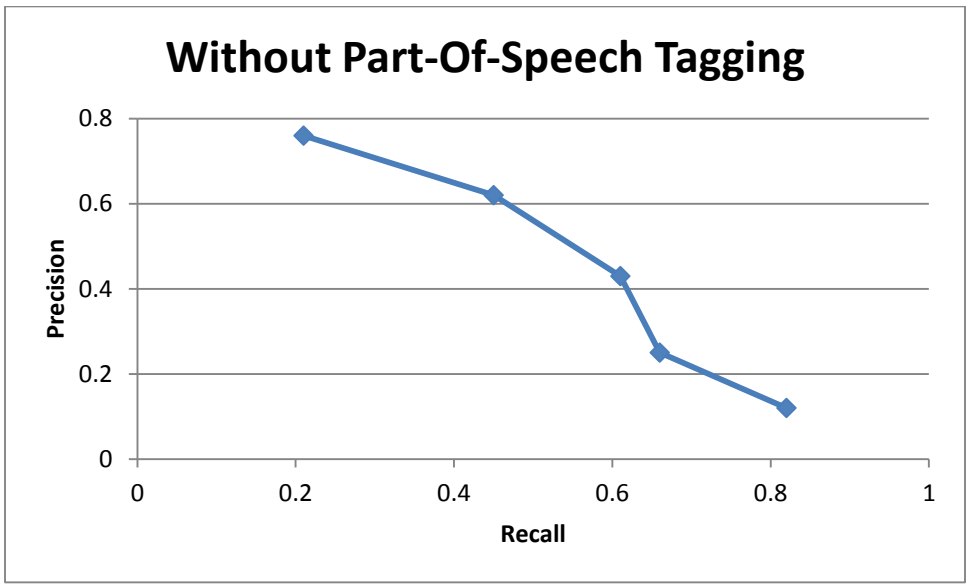

Figure 12: Recall vs. Precision without Part-Of-Speech Tagging During Crawl

A part-of-speech tagging layer is added in crawling, and experimented on the same dataset which is used for Yioop crawling, i.e., without a -part-of-speech tagging. After experimenting with part-of-speech tagging during crawl time, we get recall-precision values as follows:

Table 6: Recall vs. Precision with Part-Of-Speech Tagging During Crawl Time

Recall	Precision
0.61	0.15
0.42	0.25
0.38	0.42
0.25	0.49
0.2	0.41

The graphical representation is as follows:

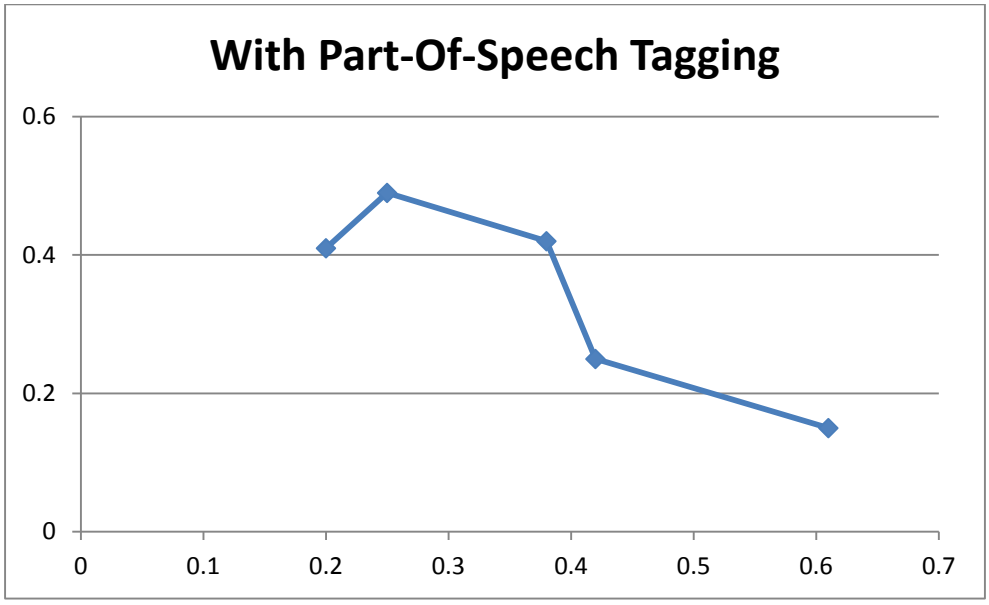

Figure 13: Recall vs. Precision using Part-Of-Speech Tagging during Crawl Time

From the above experiment, we realized certain findings which are listed as follows:

1. When recall increases, precision decreases and vice-versa.
2. The area under the curve in a part-of-speech tagging with crawl graph is less as compared to the area under the curve in graph where a part-of-speech tagging is not included in crawling. The efficiency of search results is directly proportional to the area under the curve. The graphical representation is as shown in Figure 14.

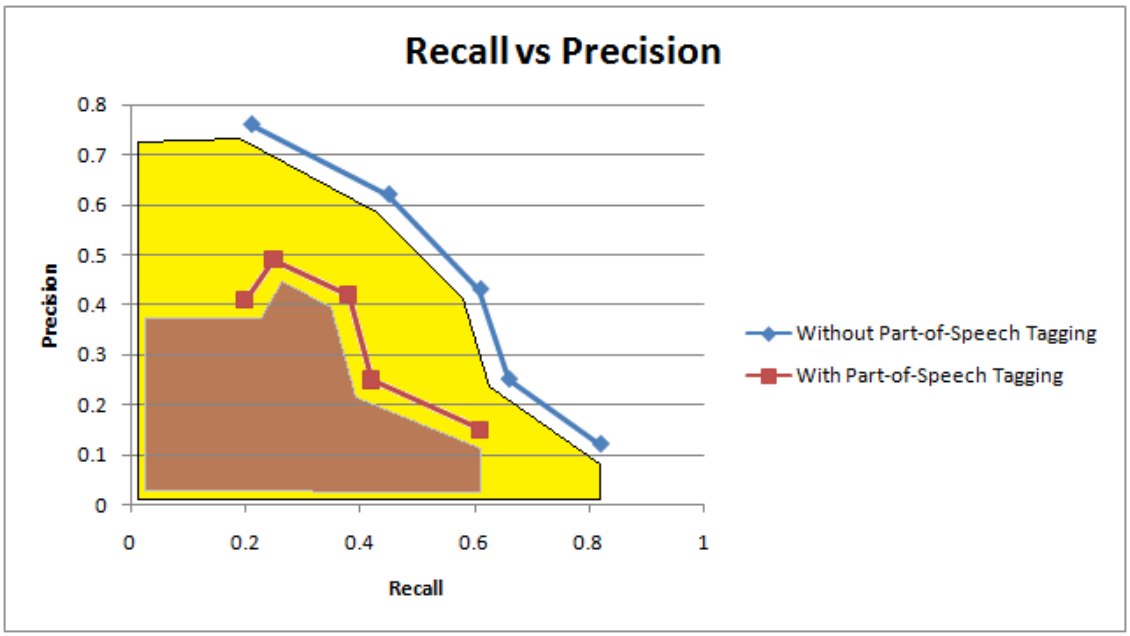

Figure 14: Area under the Curve for Both Observations

3. Number of URLs visited

- Using a part-of-speech tagging during crawl time = 660 / hour
- Not using a part-of-speech tagging during crawl time = 800 / hour

4. After adding a part-of-speech tagging layer in crawl process, the efficiency was not affected. To tag the total web page information, it takes on average 0.4 seconds.

The above results show that, the part-of-speech tagging is not effective during crawl time, but it is very important during query time.

All these experiments were carried out on the Wikipedia, SJSU CS and dmoz dataset.

Table 7: Recall Precision Values For Different Search Queries

Search Query	Recall@10	P@10	Recall@20	P@20	Recall@50	P@50	Recall@100	P@100
Computer	0.62	0.17	0.45	0.3	0.21	0.53	0.19	0.56
technology	0.71	0.21	0.57	0.41	0.27	0.69	0.24	0.64
field	0.64	0.2	0.5	0.25	0.43	0.34	0.3	0.43
world	0.8	0.19	0.63	0.34	0.57	0.4	0.3	0.55
Movies	0.77	0.15	0.56	0.26	0.45	0.34	0.3	0.46

The behavior we found is, when recall increases, precision decreases and when precision increases, recall decreases.

So $Precision \propto (1/Recall)$,

i.e. $Precision\ is\ inversally\ proportional\ with\ Recall\ and\ vice\ versa$[6]

We used the Okapi BM25 ranking function to get the scores for each document from the result set. Another input we used where the top two similar words were retrieved from the WordNet output for each word from a user input query.

Once we get the BM25 score for each webpage summary, the search results are arranged in decreasing order of the WordNet score. If WordNet output is unable to

provide the top two similar words from a given query word, then all search result web pages will be ordered with Reciprocal Rank Fusion (RRF) score[15].

We performed some experiments on the search results using the WordNet score.

Consider search query as - *information technology*

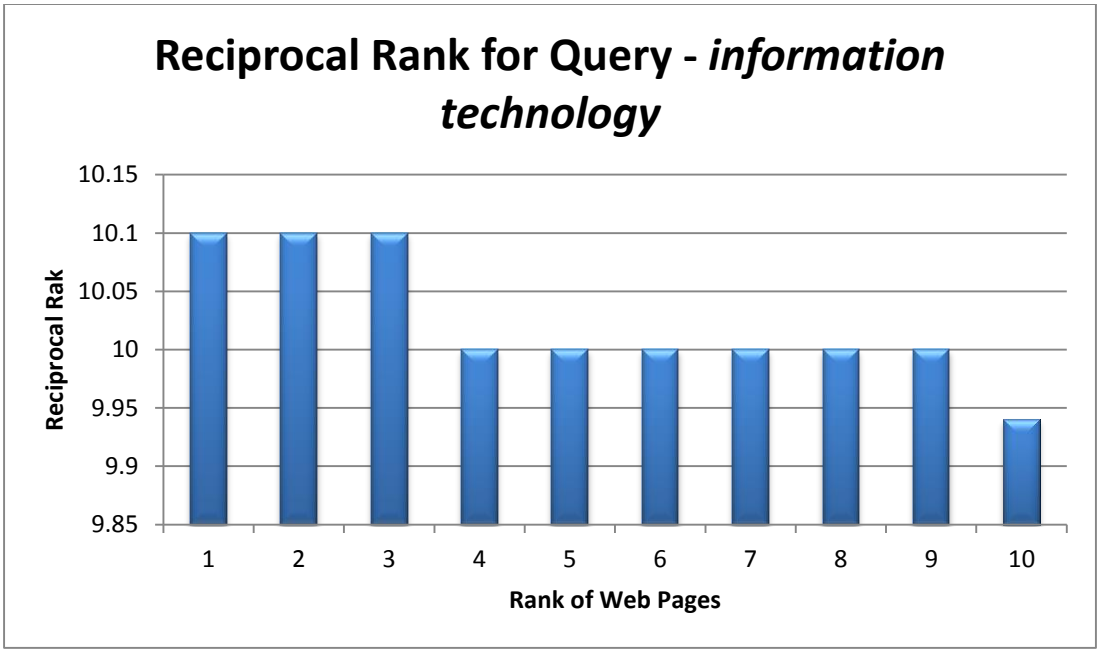

Figure 15: RRF for Web Pages from Yioop Search Result

After experimenting with the WordNet score, we got the results as follows:

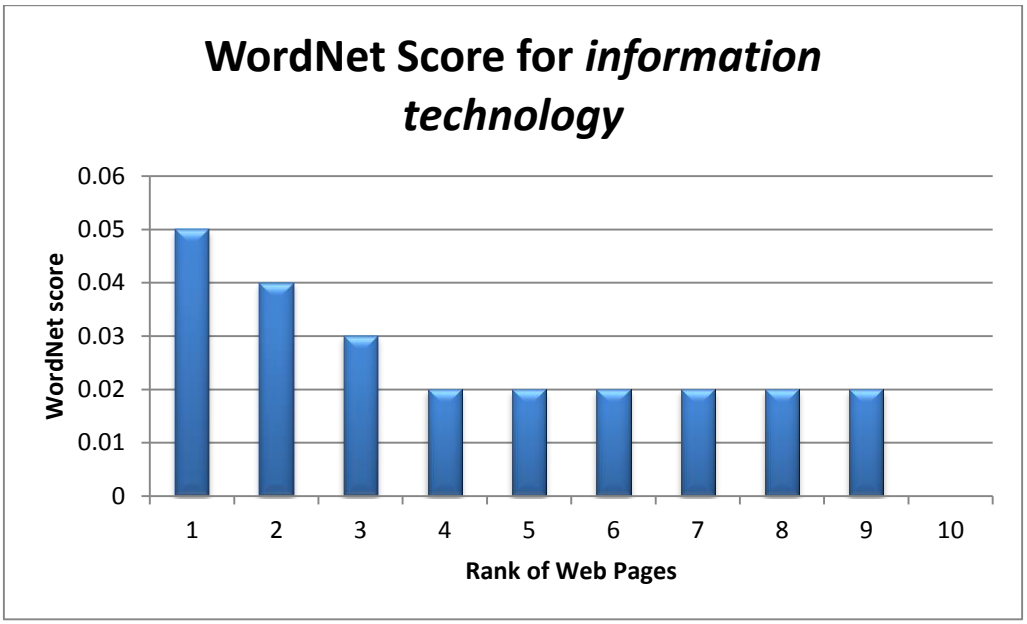

Figure 16: WordNet Score for Web Pages from Yioop Search Result

Similar experiments were carried out with different user queries and observations are as follows.

Table 8: Score Comparison between WordNet and RRF score

iron man			computing device		
Rank of Web Pages	**Yioop RRF score**	**WordNet Score**	**Rank of Web Pages**	**Yioop RRF score**	**WordNet Score**
1	10	0.13	1	9.95	0.1
2	9.64	0.07	2	9.94	0.09
3	9.64	0.03	3	10.1	0.08
4	9.94	0.03	4	10	0.06
5	9.85	0.02	5	10.1	0.06
6	9.68	0.02	6	10	0.06
7	9.68	0.02	7	10	0.04
8	10	0.02	8	9.89	0.02
9	10	0.02	9	10	0.02
10	9.69	0.01	10	10	0.02

After rearranging the web pages according to WordNet score in the Yioop search result, more relevant web pages for a given search query moved to the top in Yioop's search result. Normally, when the user inputs any query in search engine, user will prefer to select top four or five web pages. With this experiment, we got relevant documents from search result on top as per their WordNet score. So WordNet played a vital role in the query expansion technique using a query rewriting algorithm.

8. Conclusion

A query expansion is achieved using a query rewriting algorithm with the help of WordNet's search result in Yioop. Until now, WordNet has no direct API to support PHP [16]. In order to achieve that, we used WordNet command line arguments. A part-of-speech tagging and similarity ranking functions extracted the required data from WordNet output which is used for query expansion in the Yioop search engine.

After entering a query, a user will go for the top three or four search results. The search engine should be efficient enough to put those relevant web pages on top. This is achieved in the Yioop search engine by integrating WordNet. As a result, web pages in Yioop's search results are ordered by their relevance using the WordNet score. The Yioop search engine will use Reciprocal Rank Fusion (RRF) score in absence of WordNet score.

Many experiments were carried out with different user input queries on three different data sets, such as Wikipedia, SJSU CS and dmoz. The dmoz dataset is mostly used by Google for their search engine functionality testing [7]. After using query expansion technique with the help of WordNet, more relevant results from the result set were rearranged as per their WordNet rank.

We achieved a query expansion using a query rewriting algorithm with the help of the WordNet. For example, the query expansion for an input query *'computer science*' will be *'computing machine skill'*, *'information processing system scientific discipline'*, etc. A part-of-speech tagging helped us to extract meaningful data from the

WordNet output, this improved the efficiency of extraction. Similarity ranking functions such as cosine similarity and intersection ranking, effectively return similar words. Okapi BM25 Ranking functions were used to calculate the WordNet score for each web page from the search result.

The whole WordNet process is completed within 0.20 seconds on an average with standard deviation of ±0.3 with Intel duel core at 2.5 GHz system. After adding WordNet in Yioop, the efficiency of the throughput is maintained.

This project gave us a good understanding of WordNet's internal structure and how it works. The command line arguments are more beneficial to extract the data from generated output. The part-of-speech tagging helped us to reduce our work to find similarity with each sentence from the output. We used the cosine similarity ranking and intersection method to find out similarity between two sentences.

Future Work:

WordNet is limited to the English language database [16]. In the future, we can enhance the current application for any other language dictionary such as Chinese, Spanish, Hindi, etc. As Yioop is a multi-language search engine, a query expansion will work on different languages as well.

Bibliography

[1] Babluki, S. (n.d.). Build your own summary tool! The Tokenizer. Retrieved February 12, 2014, from http://thetokenizer.com/2013/04/28/build-your-own-summary-tool/

[2] Bag of words model. (n.d.). Wikipedia. Retrieved February 26, 2014, from http://en.wikipedia.org/wiki/Bag_of_words_model

[3] Barber, I. (2009, November 20). PHP/ir. Part Of Speech Tagging -. Retrieved December 10, 2013, from http://phpir.com/part-of-speech-tagging

[4] Brill tagger. (n.d.). Wikipedia. Retrieved February 12, 2014, from http://en.wikipedia.org/wiki/Brill_tagger

[5] Brown Corpus - Wikipedia, the free encyclopedia. (n.d.). Wikipedia. Retrieved December 10, 2013, from http://en.wikipedia.org/wiki/Brown_Corpus#Part-of-speech_tags_used

[6] Büttcher, S., Clarke, C. L., & Cormack, G. V. (2010). Information retrieval: implementing and evaluating search engines. *Cambridge, Mass.: MIT Press*

[7] DMOZ. (n.d.). Wikipedia. Retrieved March 20, 2014, from http://en.wikipedia.org/wiki/DMOZ

[8] George A. Miller (1995). WordNet: A Lexical Database for English Communications of the ACM Vol. 38, No. 11: 39-41

[9] Gupta, Y., Saini, A., Saxena, A., & Sharan, A. (2014). Fuzzy logic based similarity measure for information retrieval system performance improvement. *Distributed Computing and Internet Technology*

[10] Huggett, M. (2009). Similarity and ranking operations. *Encyclopedia of Database Systems*

[11] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval. *New York: Cambridge University Press*.

[12] Natural Language Processing. (2009, December 15). Natural Language Processing RSS. Retrieved February 15, 2014, from http://language.worldofcomputing.net/pos-tagging/rule-based-pos-tagging.html

[13] Okapi BM25. (n.d.). Wikipedia. Retrieved March 3, 2014, from http://en.wikipedia.org/wiki/Okapi_BM25

[14] Part-of-speech tagging. (n.d.). Wikipedia. Retrieved December 10, 2013, from http://en.wikipedia.org/wiki/Part-of-speech_tagging

[15] Pollett, C. (2010, August). Open Source Search Engine Software - Seekquarry: Home. Retrieved November 12, 2013, from https://seekquarry.com/?c=main&p=documentation

[16] Princeton University "About WordNet." WordNet. Princeton University. 2010. <http://wordnet.princeton.edu>
[17] Wikipedia for Cosine Similarity Ranking Retrieved on December 10, 2013, from website: http://en.wikipedia.org/wiki/Cosine_similarity
[18] WordNet Alternatives and Similar Software - AlternativeTo.net. (n.d.). WordNet Alternatives and Similar Software - AlternativeTo.net. Retrieved March 5, 2014, from http://alternativeto.net/software/wordnet/
[19] WordNet. (n.d.). Wikipedia. Retrieved October 5, 2013, from http://en.wikipedia.org/wiki/WordNet