German University in Cairo
Faculty of Media Engineering and Technology
Computer Science Department

Ulm University
Institute of Software Engineering and Compiler Construction

A Clustering-based Approach to Summarizing Google Search Results using CHR

馳

Bachelor Thesis

Author:	Sharif Ayman
Supervisor:	Prof. Thom Fruehwirth
Co-supervisor:	Amira Zaki
Submission Date:	13 July 2012

This is to certify that:

(i) The thesis comprises only my original work toward the Bachelor Degree.

(ii) Due acknowledgment has been made in the text to all other material used.

Sharif Ayman
13 July, 2012

Acknowledgements

First and above all, I thank God, the almighty for all His given blessings in my life and for granting me the capability to proceed successfully in my studies. I am heartily thankful to all those who supported me in any respect toward the completion of my Bachelor thesis.

My **family** for always being there for me, and giving me motivation and support towards completing my studies.

Prof. Thom Frühwirth for accepting me in ulm university to do my bachelor, and for his supervision, patience and support.

My co-supervisor, **Amira Zaki**, for her constant encouragement, insightful comments and her valuable suggestion.

My **Friends** in **ulm** for all the fun we had together.

Abstract

Summarization is a technique used to extract most important pieces of information from large texts. This thesis aims to automatically summarize the search results returned by the Google search engine using Constraint Handling Rules and PHP. Syntactic analysis is performed and visualization of the results is output to the user. Frequency count of words, N-grams, textual abstract, and clusters are presented to the user.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

As the amount of available data on the internet grows, the problem of managing the information becomes more difficult, which can lead to information overload. This phenomena is called Explosion of Information, which is the rapid increase in the amount of published information. Search engines have entered our lives, to help in neutralizing the information overload problem and help users find the information they need. They build a huge cache of websites which represent only a portion of the internet. Search engines allow users to reduce the information overload by allowing them to perform a centralized search. However, another problem arises, too many web pages are returned to the user upon searching for a single query. The user often has to examine tens and hundreds of pages to find out that only a few of them are relevant. Furthermore, skimming through the huge list of returned results containing the titles and short snippets is extremely time consuming since the results contain multiple topics which are all mixed together. This task would be made worse if one topic is very overwhelming but it is not what the user desires.

Figure 1.1: Top 7 returned image search results from Google for the search query "Apple"

An example to this would be when a user, for instance, would search for "Apple". Figure 1.1 shows the returned Google image search results for the query [1]. "Apple" is considered to be a hot topic according to Google statistics. Interestingly, the returned search results from the Google search engine contain two distinct topics: the actual fruit and the logo for the Apple Inc.. In the case where a user would be looking for the company by the name Apple, the results about the actual fruit would be considered noise to

him. If a search engine were to classify the returned web pages into meaningful clusters and provide summary information, it would be very helpful to the users in the exploration of the returned set. In order to achieve this, CHR (Constraint Handling Rules) could be used since if offers text analysis and natural language processing capabilities. Constraint Handling Rules is a high-level programming language which was invented in 1991 by Thom Frühwirth. CHR is based on multi-headed, committed-choice, guarded multiset rewrite rules and was originally invented in order to extend a host language with constraints.

1.2 Aim of the Project

The aim of this project is to automate the summarization of Google search results which involves reducing the content of multiple results pages into a short set of words or paragraphs that conveys the main meaning of the text. Also one of the aims is to represent returned web pages in an easier format to the user through clustering them. There are tools that already provide search summaries with tag clouds consisting of keywords and content summaries with links, such as SenseBot [2]. This automation is to be realized through extraction of results, parsing them via CHR and presenting the output to the user.
To be able to provide those summarizations, the following must be done. The ideal method of extraction of data from Google search results should be found, whether its scrapping for the result or getting the data from the Google API. Researching and implementing different summarization techniques and trying to choose the best ones. Try to come up with the best visualization of the summaries for ease of analysing them. All of these previous steps will be discussed thoroughly throughout the next chapters.

1.3 Overview

The thesis consists of five chapters including the introduction and the conclusion. In Chapter 2, a background view is presented. This background view explains the important basics that are used in designing and realizing this project. Chapter 3 explains the design and realization of the summarization tool. Chapter 4 contains a detailed explanation of the implementation of the summarizer. Chapter 5 contains the conclusion along with some proposed future work.

Chapter 2

Background

2.1 Constraint Handling Rules (CHR)

Constraint Handling Rules (CHR) is a high-level programming language which was invented in 1991 by Thom Frühwirth [4]. CHR is based on multi-headed, committed-choice, guarded multiset rewrite rules and was originally invented in order to extend a host language with constraints. A CHR program, is a sequence of guarded rules for simplification, propagation, and simpagation (a mix of simplification and propagation) of conjunctions of constraints [3].

1. *Simplification* rules replace constraints by simpler constraints while preserving logical equivalence, e.g.
$$Head \Rightarrow Guard \mid Body$$
2. *Propagation* rules add new constraints that are logically redundant but may cause further simplification, e.g.
$$Head \Leftrightarrow Guard \mid Body$$
3. *Simpagation* rules are a mix of propagation and simplification where the rule's head is divided into two parts. The first part remains in the constraint store of CHR after the rule is fired while the second part is simplified, e.g.
$$Head1 \setminus Head2 \Leftrightarrow Guard \mid Body$$

Guards can be used in rules if the firing of these rules depends on some condition. A guard could be written inside the rule where it represents a condition that has to be true for the rule to be fired.

CHR is appealing for applications in computational logic as logical theories are usually specified by implications and logical equivalences that correspond to propagation and

simplification rules. CHR programs have a number of desirable properties guaranteed. Every algorithm can be implemented in CHR with best known time and space complexity, something that is not known to be possible in other pure declarative programming languages. The efficiency of the language is empirically demonstrated by recent optimizing CHR compilers that compete well with both academic and commercial rule based systems and even classical programming languages.

2.1.1 Prolog

Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics. It offers a library to host the CHR rules. Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. The main compiler for Prolog used in this project is SWI-Prolog [24].

2.2 Natural Language Processing

Natural Language Processing (NLP) is the computerized approach that explores how computers can be used to understand and analyse natural language text or speech to do useful things [6][7]. There are "levels of language" that humans use to gain understanding of a text, thus, the more levels used by an NLP system, the more capable it is. Some of the natural language processing applications are:

1. Summarization: Any implementation that needs to reduce a larger text into a shorter, yet rich in information representation of the original text, needs to have a very advanced level of NLP.

2. Information Extraction (IE): Focuses on the recognition, tagging, and extraction of certain key elements such as persons, locations, from a large number of documents into a structured representation. These extractions can then be utilized for a range of applications including question-answering, visualization, and data mining.

3. Machine Translation (MT): NLP has been used in machine translation applications, ranging from "word-based" approaches to applications that include higher levels of analysis.

2.2.1 Levels of languages in NLP

Phonology

At this level, spoken language is transformed into sound waves which are analysed and encoded into a digital signal that is interpreted by various rules or through comparison with another language model.

Morphology

In this level, every word is decomposed into the smallest units of meaning called morphemes. For example, the word "unkindly" can be morphologically analysed into three separate morphemes: the prefix "un", the root "kind", and the suffix "ly". Humans can break down an unknown word into its building morphemes in order to understand its meaning because the meaning of each morpheme remains the same across words. NLP systems also follow this logic in gaining meaning to an unknown word. For example, when the suffix "ing" is added to a verb, it is interpreted that the verb is taking place in the present.

Lexical

This level deals with the interpretation of single words individually. Words that have a single meaning are replaced with a semantic representation of that meaning, where the nature of the of the representation depends on the semantic theory used in the NLP system. Since there is almost certainly a set of simple semantic representations shared between words, more complex interpretations could be drawn, like humans do.

Syntactic

At this level, words in a sentence are analysed in order to discover the grammatical structure of the sentence. This typically requires both a grammar base and a parser. The representation of the sentence is returned after the processing of this level, which uncovers the structural dependency relationships between the words. Meaning is conveyed through syntax in most languages because order and dependency contribute to meaning. For example the two sentences: "The boy ate the apple." and "The apple ate the boy." differ only in terms of syntax, yet convey quite different meanings.

Semantic

This level focuses on the determination of the possible meanings of a sentence by concentrating on the interactions among word-level meanings in the sentence.

Semantic disambiguation permits one and only one sense of multi-meaning words to be selected and included in the semantic representation of the sentence. For example, amongst other meanings, "file" as a noun can mean either a folder for storing papers, or a tool to shape one's fingernails, or a line of individuals in a queue. The meaning of the word is then determined through the consideration of the local context, or through utilizing the knowledge domain of the document.

Discourse

This level works with units of text typically longer than a sentence, on the contrary to syntax and semantics, which work with sentence long units. At this level, multi-sentence texts are not interpreted as just concatenated sentences, each of which are interpreted individually. Rather, through making connections between the sentence units, a general meaning is extracted from the whole text. One of the used discourse processing approaches is text structure recognition. This approach determines the functions of sentences in the text, which, in turn, adds to the meaningful representation of the text. For example, newspaper articles can be de-constructed into text components such as: Lead, Main Story, Previous Events, Evaluation, Attributed Quotes, where a set of sentences belong to each of these components.

Pragmatic

This level deals with the usefulness of the used language in particular situations. At this level, context reigns supreme over the contents of the text in understanding its conveyed meaning. Through the availability of world knowledge, including the understanding of intentions, and plans, extra meaning could be extracted from texts without actually being encoded in them.

2.2.2 N-Gram

An N-gram is an adjacent N-word slice of a longer string [25]. The N-gram model is one of the language independent approaches in statistical Natural Language Processing. An N-gram of containing one word is called a "unigram", two words would be referred to as a "bigram" and a three-word sequence is called a "trigram". For higher order N-grams, they are referred to by their values, e.g., "four-gram", "five-gram" and so forth. The string "The Eiffel Tower is in Paris" could be sliced into a set of overlapping N-grams:

- Bigrams: "The Eiffel", "Eiffel Tower", "Tower is", "is in", "in Paris"
- Trigrams: "The Eiffel Tower", "Eiffel Tower is", "Tower is in", "is in Paris"
- Four-grams: "The Eiffel Tower is", "Eiffel Tower is in", "Tower is in Paris"

The N-gram model is used to help in the clustering of the Google search results in this thesis.

2.3 Web Development

2.3.1 API

Application Programmable Interface (API) is a set of methods, protocols, and tools that aid in the building of software applications. An API can be language-dependent or language-independent. An example of an API is the Google API.

Google API

The Google APIs allow developers to create web applications that read and write data to and from the Google services. As of now, these include APIs for Google Apps, Google Analytics, Blogger, Google Base, Google Book Search, Google Calendar, Google Code Search, Google Earth, Google Spreadsheets, Google Notebook, and Picasa Web Albums.

2.3.2 PHP

PHP, which stands for "PHP: Hypertext Preprocessor" is a widely-used open source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Its syntax draws upon C, Java, and Perl, and is easy to learn. The main goal of the language is to allow web developers to write dynamically generated web pages quickly. One of the used capabilities of PHP, is the parsing function which allows the developer to extract certain strings from a document using a regular expression describing them [23]. Also, PHP was used as an interface which allowed for the consulting of Prolog programs and the passing of inputs to them, and receiving the output.

2.3.3 JSON

JSON (JavaScript Object Notation) is a data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language [8].

JSON is built on two structures:

1. A collection of name/value pairs. In various languages, this is realized as an object, record, structure, dictionary, hash table, keyed list, or associative array.

2. An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An object is an unordered set of name/value pairs. An object begins with (left brace) and ends with (right brace). Each name is followed by: (colon) and the name/value pairs are separated by , (comma) as seen in figure 2.1.

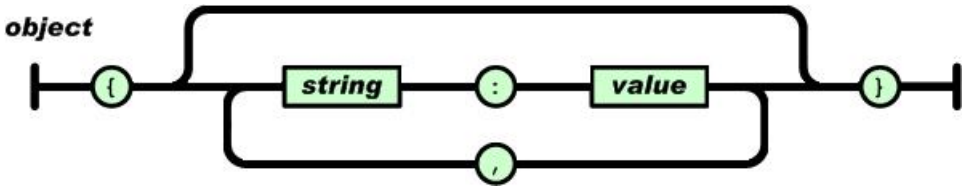

Figure 2.1: An object in JSON [8]

An array is an ordered collection of values. As illustrated by figure 2.2, an array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).

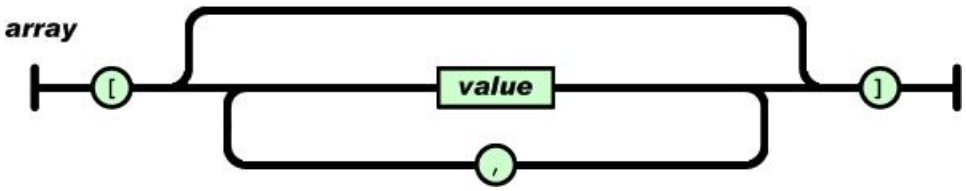

Figure 2.2: An array in JSON [8]

2.3.4 Atom

The Atom Syndication Format is an XML language format, wherein XML stands for Extensible Markup Language. This format is mainly used to describe feeds which are lists of related information [9].

2.3.5 cURL

cURL is a free command line tool for transferring data with URL syntax, supporting DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet and TFTP. cURL supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user-plus-password authentication (Basic, Digest, NTLM, Negotiate, kerberos...), file transfer resume, proxy tunneling [10]. It is mainly used to fetch the source code for HTML pages.

2.3.6 D3.JS

D3.js is a JavaScript library for manipulating documents based on data [11]. This library provides amazing flexibility, and brings out the full potential of web standards such as CSS3, HTML5 and SVG. Binding arbitrary data to a Document Object Model (DOM), and then applying data-driven transformations to the document is its main feature. Since its a low-level framework, full control of the data visualization is achieved, although more work on the behalf of the developer would be needed. Some of the visualizations that can be created with this powerful tool are shown in the figure 2.3, such as trees, force-graphs and charts. The tree structure has been used to produce the cluster tree which will be illustrated in the implementation chapter.

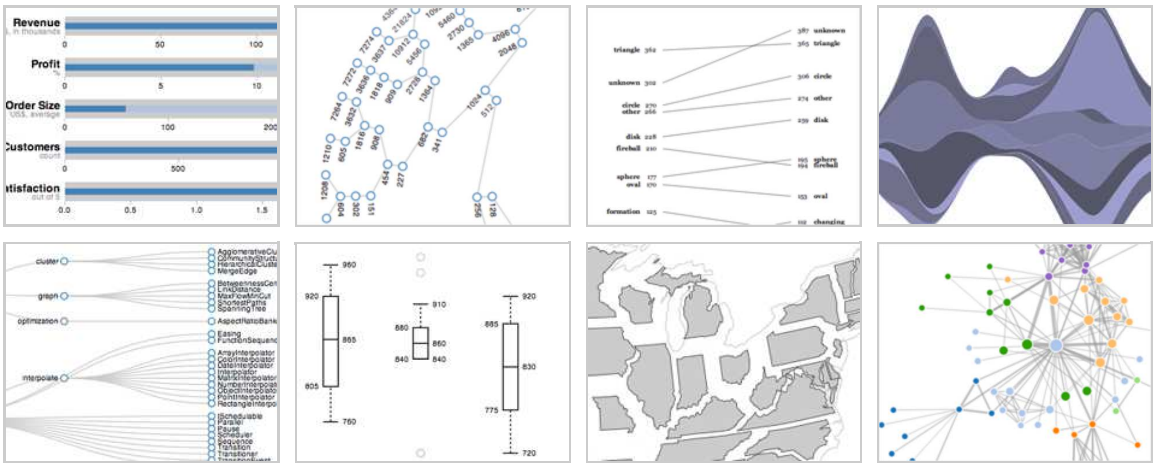

Figure 2.3: Examples of the visualizations performed by D3.js [11]

Chapter 3

Design decisions and the different approaches

In this chapter, the different approaches tried throughout the project, and the final system architecture will be discussed. The first main part which describes the approaches contains four sections, the methods of extracting data, the general summarization techniques, the more advanced summarization techniques and finally a comparison between two-dimensional (2D) and three-dimensional (3D) visualization techniques is presented. The second main part will talk about the final system architecture in detail.

3.1 Extracting data

3.1.1 Google Search Results

Google Custom Search API

The Google Custom Search API allows developers to create web applications to retrieve and display search results from Google Custom Search programmatically. With this API, requests can be used to get either web search or image search results in JSON or Atom format. It provides an extensive interface which allows for the complete manipulation of the search results. However, it is limited to ten pages of search results per query and only up to one hundred queries per day.

Scraping

Web scraping (also called web harvesting or web data extraction) is an algorithmic method of extracting information from web pages. Web scraping may be against the terms of use of some websites. The process fetches data from the internet by either data mining

algorithms, or regular expression matching facilities of programming languages such as (PHP or Python).
Web scraping transforms the web content of an unstructured nature which is typically in HTML format, into structured data that can be formatted, analysed and displayed. This is the main technique to extract the data from the Google search results HTML page, since it allowed a maximum of one thousand pages of search results per query to be extracted.

3.1.2 Google Alerts

Google Alerts are emails sent to the user whenever Google finds new results – such as web pages, newspaper articles, or blogs – that match the specified search term. Google Alerts could be used to monitor anything on the Web. For example, people use Google Alerts to:

1. find out what is being said about their company or product.
2. monitor a developing news story.
3. keep up to date on a competitor or industry.
4. get the latest news on a celebrity or sports team.
5. find out what's being said about themselves.

Through the accumulated Google alerts emails, a large dataset was extracted and used as an information pool to extract the summaries from [12].

3.2 General Summarization Techniques

3.2.1 Extraction

Extraction works by selecting a subset of existing words, phrases, or sentences in the original text to form the summary. The salient, i.e., relevant text units (typically sentences) are determined through the lexical and statistical relevance. The weight of each text unit is calculated, during the preprocessing phase of the document, according to features such as the unit's position in the source document, how often it occurs in the text, the usage of cue phrases in such text unit. The sum of these individual weights, is the overall weight of the text unit U:

$$Weight(U) := Location(U) + Frequency(U) + CuePhrase \tag{3.1}$$

The model determines location weight according to whether the text unit is in the initial, middle, or final position in a paragraph or the entire document, or whether it occurs in

prominent sections, such as the document's introduction or conclusion. The cue phrase is a lexical or phrasal summary cue such as "in conclusion," "in this paper," "our investigation has shown," or "as a result". In conclusion, extraction approaches are easier to adapt to larger text sources. Because they are limited to the extraction of passages, sentences, or phrases, however, the resulting summaries may be incoherent.

3.2.2 Abstraction

Abstraction builds an internal semantic representation and then use natural language processing including grammars and lexicons for parsing and generation. It also requires some common sense, i.e., artificial intelligence, for reasoning during analysis and salience computation. Abstraction has two main approaches.

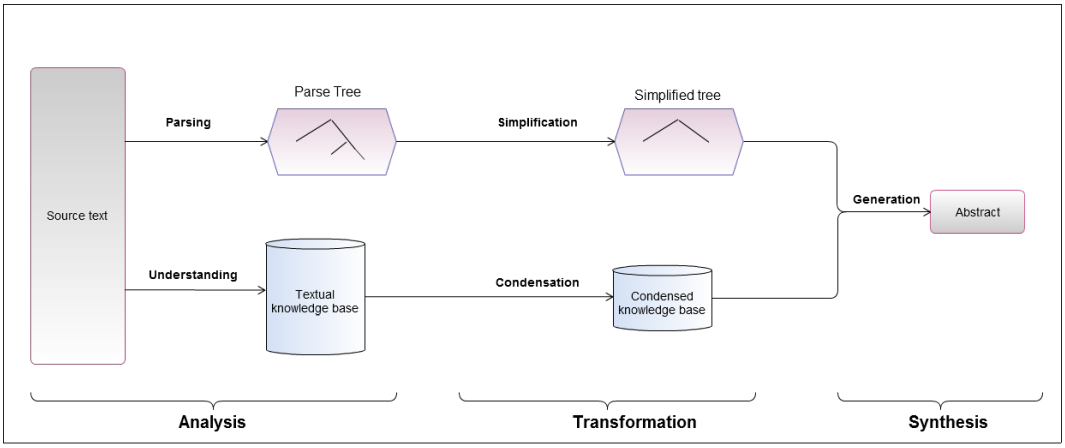

Figure 3.1: The main two approaches of Abstraction

The first approach (top of figure 3.1) parses sentences using a linguistic method into parse trees. These trees are then compacted by removing and regrouping parts of them. After simplification, the original parse tree is considerably smaller becoming in essence a structural condensate. The second abstraction (bottom of the figure) approach relies heavily on machine reasoning and focuses on natural language understanding [13]. Analysis also includes parsing, however, the outcome is not parse trees. Instead they are text knowledge bases which consist of the conceptual representation of the source text. In both approaches, the transformation phase outputs an abstract representation structure of the summary. As figure 3.1 shows, the synthesis phase is the same for both approaches: a text generator translates the structural or conceptual representation to produce a fluent natural language abstract. Abstraction approaches provide more sophisticated summaries, which often might contain words not explicitly present in the original text. Because they are based on a formal representation of the document's content, they adapt well to high compression rates [14].

3.3 Advanced Summarization Techniques

3.3.1 Word Frequency

It was the first method used to produce summaries by Luhn [15] where it relies on the assumption that the frequency of a particular word in a document provides a useful measure of its significance. As a first step, stop words were deleted. Stop words are words that are commonly used, yet have no significance by themselves. These words are removed from the source document prior to, or after, processing of natural language data (text). There is no definitive list for these words, therefore any group can be categorized as stop words. Some examples of these stop words are “the”, “is”, “a”, “in” and so on. Luhn then compiled a list of content words sorted by decreasing frequency, the index providing a significance measure of the word. Even though this method is very simple, it is still used in combination with other methods [16].

3.3.2 Positioning of Sentences

Term weights are differently weighted by the location of a term, so that the structural information of a document was applied to term weights. But this method supposes that only several sentences, which are located at the front or the rear of a document, have the important meaning. Hence it can be applied to only documents with fixed form such as articles. The benefits of this approach are minor since the search results do not have a structure with observable patterns [17].

3.3.3 Sentence Weight

Firstly, the measurement of the importance of terms via the WF (Word Frequency) statistic values is performed. The importance of the sentence is then calculated through the summation of the importance values of terms in the sentence. Thus, sentences with more important terms are assigned higher importance. Then, all sentences are ranked in order of their significance factor, and the top ranking sentences are finally selected to form the summary.

3.3.4 Similarity of Sentences

Since the summary is based upon the Google search results, there are bound to be largely similar, if not identical snippets. Thus to be able to detect similar strings, a comparison between the two of them is performed and the number of characters they have in common is returned. Finally a percentage of similarity is computed, and with this percentage, identical strings or largely similar ones are excluded from the final summary [18].

3.3.5 Positioning of Common Words in Sentences

Word positioning plays an important role in measuring emphasis level intended by the author. There are two general word-positioning rules for emphasis [19]:

1. the initial and end positions of sentences are by nature more emphatic than their middles.
2. the most prominent word, or group of words, is usually at the end of the sentence.

Also, the visualization of the word positioning might lead to the discovery of text patterns such as lexical chains which might contribute to better understanding of the search results.

3.3.6 Relativity between Words

In order to measure relativity, the distancing between two pairs of words in a sentence is used.

$$Relativity = \begin{cases} (\frac{1}{2})^g \end{cases}$$

where g is the positive distance between two words in the same sentence

3.3.7 Clustering

Since there are hundreds upon thousands of URLs returned from search engines for any given query. The user is forced to sift through them to find the pages he/she needs. This limitation of search technology can be attributed to the following:

1. Polysemy: the words involved in the search query have multiple meanings. For example, a user searching for "jaguar" may be interested in either the car company or the actual animal.
2. Phrases: a phrase may be different from words in it. e.g., the meaning of the phrase "partition magic" (a disk partition management tool) is quite different from the meaning of the individual words "partition" and "magic".
3. Term dependency: the words in the query are not entirely independent of each other. For example, a user may look for details about a product made by a particular company and type in "Adobe Acrobat Reader". Obviously, each word in this term is dependent on each other.

One possible solution to this problem is to group together results from the search engine into meaningful clusters. If the user is presented with these clusters, with some keyword type descriptions, they can select one (or more) that fit their perceived interests [20][21]. In order to realise this clustering effect, n-grams sharing the same topic should be grouped together according to the measure of similarity of sentences between them.

3.4 Comparison of 2D and 3D Visualization

A study had been conducted by Sebrechts,Vasilakis, Miller, Cugini, Laskowski [22] to compare between the usability of two-dimensional (2D) and three-dimensional (3D) user interfaces. In regards to mental workload, navigation in text or two-dimensional (2D) models was relatively low compared to their three-dimensional 3D counterparts. Also user experience affected the performance of the test subjects. Response times were greatly reduced over time under the 3D conditions and in the end was comparable to the 2D and text conditions. It was concluded that computer skills mattered in response time under the 3D conditions, such that with greater computer skills comes faster response times.

Furthermore, colour-coding turns out to be an important factor in usability, because it provided a quick method of traversing the clusters due to grouping regardless of the visualization tool. Searching for a title through a text list was often easier than having to locate a title by first identifying a group. In contrast, if documents were linked in "neighbouring" clusters, the 2D and 3D tools were easier than a text-based search. With larger documents, 2D and 3D visualization interfaces tend to outperform the scrolling and scanning of documents under the text based versions. Thus it was decided that the user interface for the summarizer was to be made using a combination of a 2D and text environment.

3.5 System Architecture

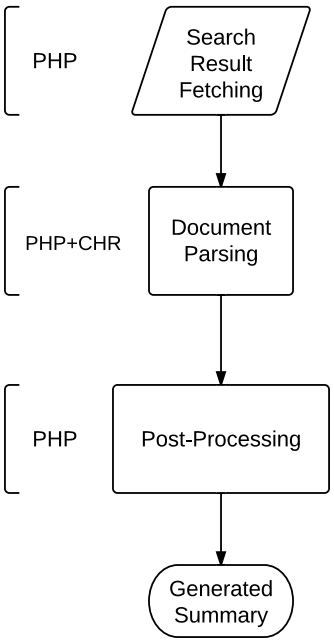

Figure 3.2: Flowchart for Summarization

As figure 3.2 shows, firstly, the Google search results are captured via scrapping. These web pages are then analysed by PHP and result items are extracted. Generally, only query-dependent snippets are available for each result item. Afterwards, several properties for each distinct text unit are calculated during the parsing. These properties were described in detail in the previous sections. In the parsing, the following processes take place:

1. stop words are removed from the text using PHP.
2. frequency count of each word is calculated using CHR.
3. indexing of the words in their respective sentences is created using CHR.
4. relativity between words is computed using CHR.
5. sentence weight is calculated based on the word frequency information using PHP.
6. calculation of sentence pairs' similarity is carried out using PHP.
7. n-gram generation up to the fourth tier using PHP.

In the post-processing, similar sentences which have a similarity factor of 100% are removed. Clustering is performed and then the descriptions of the clusters are created through the union of the n-grams that shared the same topic. A tag cloud containing the sorted words, where the size of the word is directly proportional to its frequency. An abstract is created using the descendingly sorted relevant sentences. After being sorted in a descending order, N-grams are listed to the user. The indexing of the top most frequent words is plotted onto a scatter graph. From the clustering data collected, a cluster tree is drawn, where the user has the freedom to traverse the leaves. All of these visualizations will be later illustrated in the implementation chapter.

Chapter 4

Implementation

In this chapter, the implementation of the summarization web application, the output to the user and the evaluation of the project shall be discussed.

4.1 Implementation

Firstly, the Google search results are captured via scrapping. These results are then analysed by PHP, some operations are performed, then the analysed results are written into a text file. PHP would execute a Prolog predicate through a command prompt-like function. The predicate will then output its computations to PHP. Using these computations, PHP would perform more analysis and output the results to the user in either a text-format or a visualization. After certain code snippets, the respective output to the user is displayed. The search query used during the output presentation is the word "eiffel".

4.1.1 Extraction of data

In order to get the data that would be analysed for summaries, a request is sent and data is received using a cURL object to the Google servers. The following code snippet shows how to instantiate such an object.

```
$options = array(CURLOPT_RETURNTRANSFER => true,     // return web page
  CURLOPT_HEADER         => false,    // don't return headers
  CURLOPT_FOLLOWLOCATION => true,     // follow redirects
  CURLOPT_ENCODING       => "",       // handle all encodings
  CURLOPT_AUTOREFERER    => true,     // set referer on redirect
  CURLOPT_CONNECTTIMEOUT => 20000,      // timeout on connect
  CURLOPT_TIMEOUT        => 20000,      // timeout on response
```

```
 CURLOPT_MAXREDIRS      => 10,       // stop after 10 redirects
 CURLOPT_COOKIEFILE     =>    "cookie.txt",
 CURLOPT_COOKIEJAR      =>    "cookie.txt",
 CURLOPT_USERAGENT      =>    "Mozilla/5.0
 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.3)
  Gecko/2008092417 Firefox/3.0.3",

 CURLOPT_REFERER        =>   "http://www.google.com/",    );
    for ($page = $start; $page < $npages; $page++){
     $ch = curl_init($gg_url.$page.'0');
     curl_setopt_array($ch,$options);
     $scraped="";
     $scraped.=curl_exec($ch);
     curl_close( $ch );
     $results = array();//temp array to store all the page results
     $resultsUrl = array();//temp array to store all page urls
.............................................................
```

The received data is in the form of a basic HTML code which shall be scrapped. This HTML code is then saved in an array.

```
preg_match_all('/<div class="vsc".+?<a href="([^"]+)"
class=l.+?>.+?<\/a>/',$scraped,$resultsUrl);
// This method retrieves the URL of the individual
search results through matching with the <a> tags.

preg_match_all('/(?<=span class="st">).*?<\/span>/'
,$scraped,$results);
// This method retrieves the content of the individual
 search  results through matching with the <span> tags.
```

The `$scrapped` array contains the source code of the html returned from the curl execution. Parsing of the array is carried out using the function
`preg-match-all(regular expression,original array ,modified array)`
which matches with the regular expression from the original array and keeps the relevant data in the modified array. Urls of all snippets are saved in the `$resultsUrl` array, and the actual texts of the Google search results are saved in the `$results` array.

4.1.2 Removal of Stop Words

Now this modified array still contains useless html tags, these are removed using the `strip_Html_tags()` function. Then unnecessary characters such as dots, commas and some non-important vocabulary such as stop words were compiled into lists.

```
$stringData =strip_html_tags($stringData);
$invalid = array("#39;", "&'");
$unimp1 = array("And", "The","The","It","He",.....);
//list containing stop words
$unimp2 = array("..." ,"\n","&amp;","'","\"" ,":" ,.....);
//unnecessary characters list
```

Next `str_ireplace()`and `preg_replace()` are used to remove the words which match any elements from the above lists. `str_ireplace()` is a predefined PHP function which searches for a certain subset of a string and replaces it with another. `preq_replace` is also a predefined PHP function which performs similarly to the `str_replace`, however, it does have an added feature of defining a regular expression for a pattern match. As shown in the next code snippet, the `\b` tags are used to limit the pattern matching to whole words, not any subsets, i.e., "the" is only matched, not the subset of another word like "therefore". The `i` tags on the other hand, are to apply case insensitivity matching.

```
$stringData = str_ireplace($invalid, "'", $stringData);
$stringData = str_ireplace($unimp2, "", $stringData);
$stringData = preg_replace('/\b('.implode('|',$unimp1).')\b/i',
'',$stringData);
```

Finally the array is split on white spaces and each cell (which is a word)is printed into a new line in an external text file using the function `fwrite`. `fwrite` is a predefined PHP function used to write into a file. However, to be able to write to the file, it must be opened first through `fopen`, which is also a predefined PHP function. Given a path to the file, the `fopen` function, opens a stream to the file for further use.

```
$stringData =split(" " , $stringData);

$myFile =
"C:\\Programme\\Apache Software Foundation\\Apache2.2\\htdocs\\text.txt";
$fh = fopen($myFile, 'w') or die("can't open file");
foreach( $stringData as $line)
{

$line =ucwords($line);
fwrite($fh, "$line\n");

}
```

4.1.3 Calling the Prolog Program from PHP

Usually, command prompt could be used to call SWI-Prolog [24]. PHP has a predefined function which simulates the command prompt, called `shell_exec`. This function is used

to execute a command via shell and return the complete output of the command in a string. The command used to call SWI-Prolog has many parts:

1. The SWI-Prolog filepath
2. The compiler options: The `-f file` option indicates that a file is used as the initialisation file instead of the default .plrc (Unix) or pl.ini (Windows).
3. The filepath of the Prolog file to be consulted
4. The name of the predicate or group of predicates to be executed
5. The stack-sizes:
   (a) `-Gsize`: Limit for the global stack which is also called term-stack or heap.
   (b) `-Lsize`: Limit for the local stack which is also called environment-stack.
6. The running goals: The `-g goal` controls which predicate gets to be executed before entering the top level. The default is a predicate which prints the welcome message.

```
$output = shell_exec("C:\\\Programme\\pl\\bin\\swipl
//filepath for the SWI-Prolog executable
 -f C:\\\Programme\\pl\\bin\\readfile.pl
 //filepath of the consulted Prolog file
  -g test,halt"); //test is the predicate to be executed,
   and then after test is done,
   halt which is a built-in predicate is called to terminate the program.
```

4.1.4 Reading Parsed Data from PHP by Prolog

The text file that was written to by PHP is accessed by Prolog through the function the `readFile` predicate, which opens the file, gets its content and closes it calling the `readText` predicate. The idea was to read each line (which contains normally one word) from the text file into a cell from the list.

```
readFile(FilePath,List):-
                        open(FilePath, read, Stream),
                        readText(Stream, List),
                        close(Stream),
                        wording(List,0,0).
```

The `open` predicate is a predefined predicate which contains three attributes, filepath, mode, stream. The `mode` attribute has three possible states:

1. `read`: Open the file for input.
2. `write`: Open the file for output. The file is created if it does not already exist, the file will otherwise be truncated.
3. `append`: Open the file for output. The file is created if it does not already exist, the file will otherwise be appended to.

The `readText` is a predicate that is used to read an input stream to a list of character codes. Reading stops at the newline or end-of-file characters.

```
readText(Stream, [Head|Rest]):-
                                read_line_to_codes(Stream, Codes),
                                dif(Codes, end_of_file), !,
                                atom_codes(Head, Codes),
                                readText(Stream, Rest).

readText(Stream, []):-
                                read_line_to_codes(Stream, end_of_file),
                                !.
```

4.1.5 Indexing of Words

Now the list containing all the unigrams is traversed. For each word in the list, a corresponding constraint `word/1` is produced, where its attribute is the actual word itself. The `atom_codes` predicate has two attributes, the atom and the resulting array of ASCII codes representing the atom. The `indexing` constraint has three attributes, word, sentence id (`S`), and index of word in the sentence (`Sid`).

```
wording([Head|Rest],S,Sid):-%The list is the list of words yet to be
 converted to CHR constraints
                            atom_codes(Head, Codes),
                            Codes \= [46],%check if the character is a dot.
                            word(Head),
                            indexing(Head, S,Sid),
                            Sid1 is Sid+1,%incrementing index in sentence
                            wording(Rest,S,Sid1).
wording([Head|Rest],S,Sid):-
                            atom_codes(Head, Codes),
                            Codes = [46],
                            S1 is S+1,%incrementing sentence id
                            Sid1 is 0,%reseting sentence index to 0
                            wording(Rest,S1,Sid1).
```

The algorithm iterates over the `wording` list, generating a `word` constraint and an `indexing` constraint for every element which is not a dot. The dot in the `wording` list signifies that the current sentence has ended and a new sentence will begin in the next iteration. The counter for the index of the word in the sentence is incremented only if the current element in the list is not a dot, otherwise, sentence id is incremented and the counter for the index of the word is reset to 0.

4.1.6 Word Frequency Calculation

The frequency of the word is represented by the constraint `count/2`. The attributes of count are the word and its frequency count. The calculation of the frequency is based upon two conditions:

1. If, one `word` constraint representing the word, is available in the constraint store, simplify into a `count` constraint describing that word's frequency where its frequency is one.
2. If, two `count` constraints for the same word are in the constraint store, sum up their frequencies into a new frequency, simplify both constraints into a new `count` constraint with the new sum.

```
word(X) <=> count(X,1).
count(X,C), count(X,C1) <=>
                            C2 is C1+C,
                            count(X,C2).
```

4.1.7 Relativity between Words

Having the `indexing` of all the words, the `relativity` is calculated through measuring the distance `D` between two words, `W1` and `W2`, in the same sentence `Sid`, with indexes in the sentence, `S1` and `S2` respectively. The relativity is calculated only if the word `W1` has a lower index `S1` than the index `S2` of word `W2`, and the distance `D`, calculated by subtracting `S1` from `S2` is lower than four. The `relativity` constraint contains three attributes, the first word `W1`, the second word `W2`, and the actual relativity `Dis`.

```
indexing(W1,S1,Sid),indexing(W2,S2,Sid) ==>
                                            S1<S2,
                                            D is S2-S1,
                                            D < 4,
                                            Dis is 0.5 ** D
                                            | relativity(W1,W2,Dis).
relativity(W1,W2,Dis1),relativity(W1,W2,Dis2) <=>
                                            Dis is Dis1+Dis2,
                                            relativity(W1,W2,Dis).
```

4.1.8 Outputting Results to PHP

When the initial list of words is empty, and the final `indexing` and `word` constraints are in the store. The `printing/0` constraint is issued into the store to start the printing process. This rule prints the count of each word in the store into the output stream, to be read by PHP and deletes them through the simplification rule. The printing of the `indexing` constraint is the same as the printing of the `count` constraint.

The stream where the two rules have written onto shall be printed on the command window, and the Prolog program is terminated. The output is then written to a string in PHP, which is afterwards split into two main entities:

1. The word and its frequency which is represented by two arrays which are mapped to each other.
2. The word, its index in the sentence, and the sentence id are all mapped into three arrays.

```
$output = explode("[116]", $output);
//initial array containing the frequency information.
$indexOutput=explode("[]", $output[0]);
//initial array containing the indexing information.
//These arrays are then split to their respective arrays.
//One of the splitting processes is shown below.
$count=array();//array to store frequencies from Prolog
  $word=array();//array to store matching word from Prolog

  //storing the words and their frequencies from Prolog
  foreach( $output as $line){

      $split = explode("," , $line);
      $split2[0]=str_replace("count(","",$split[0]);
      $split2[1]=str_replace(")","",$split[1]);
      $word[$j]=$split2[0];
      $count[$j]=$split2[1];
      $j++;

  }
```

After calculating and saving the word frequencies, a tag cloud is drawn where the size is directly proportional to the frequency count as shown in figure 4.1. The user is then able to click the word and get the links where it has occurred.

Figure 4.1: Clickable Tag Cloud for search query "eiffel"

Finally due to the availability of the indexing of all words, a scatter plot featuring the indexing information of the top six most frequent words is plotted. The x-axis is the sentence id of the word, whereas the y-axis represents the index of the word in the sentence. This graph is quite helpful since it might lead to the discovery of text patterns such as lexical chains or points of emphasis.

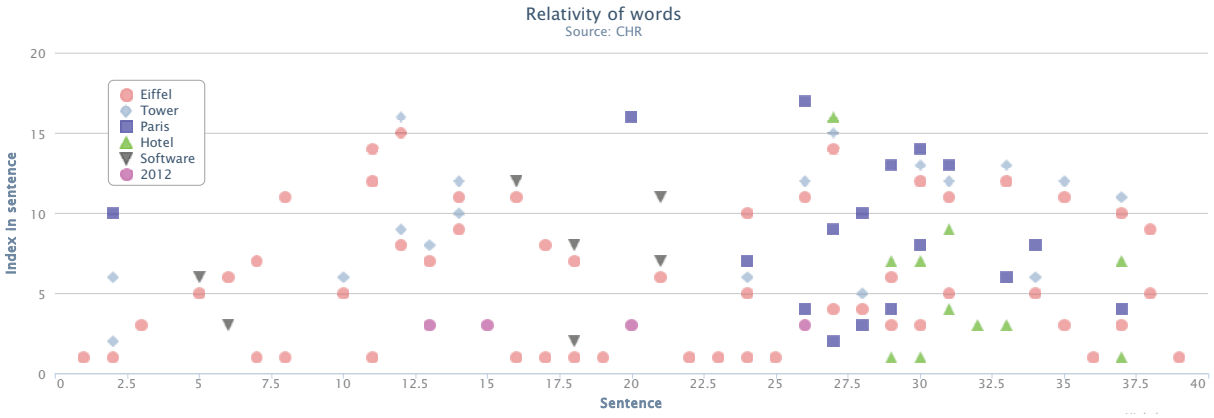

Figure 4.2: Scatter Plot for Positioning of top 6 Common Words for search query "eiffel"

4.1.9 Sentence Weight Calculation

The weight of the search result snippet is computed through the summation of the frequencies of the one grams that occur in the sentence. The array `$weightOfSentence` is an associative array which stores the weights of the sentences. An Associative array is a collection where each item is a `(key,value)` pair where the key only occurs once in the collection. The arrays `$wordin` and `$wordinx` which are mapped together, represent the word and its sentence id. While iterating over the word list, the count of the word is acquired through search the frequency map for the word. The count is then added to the weight of the sentence.

```
$weightOfSentence=array
```

```
for($z=0; $z<sizeof($wordin); $z++){

  if(in_array($wordin[$z],$word)){
    $j=array_search($wordin[$z],$word);
    $x=$count[$j];
    if(!isset($weightOfSentence[$wordinx[$z]])){
      $weightOfSentence[$wordinx[$z]]=$x;
    } else {
      $weightOfSentence[$wordinx[$z]]+=$x;
    }
  }
}

arsort($weightOfSentence);
```

After the loop terminates, all the weights would have been calculated. Afterwards, they are sorted in descending order according to the sentence weight through the use of the predefined PHP function `arsort($weightOfSentence)`. Thus, the highest weighted sentences are at the top, which correspond to their importance as discussed in the approach chapter and as shown in figure 4.3. Also, a textual abstract is drawn up from the weight

Sentence 24] importance is 191
Sentence 31] importance is 179
Sentence 27] importance is 174
Sentence 30] importance is 165
Sentence 13] importance is 163

Figure 4.3: Top five weighted sentences with their scores for search query “eiffel”

information, where the sentences are ordered by their scores descendingly as shown in figure 4.4.

Eiffel refer [edit] Engineering Eiffel Tower Paris designed Gustave Eiffel motif twin Eifel ridge redubbed Maria Pia Bridge previously .Hotel Ares Eiffel 4 star Boutique Hotel Paris 40 Designer rooms Eiffel Tower Paris Unesco . Novotel Paris Tour Eiffel 764 rooms located heart Paris overlooking Seine stones throw Eiffel Tower hotel .Explore Official Site Hotel Eiffel Seine threestar boutique hotel close Eiffel Tower Paris .Eiffel Analysis Design Programming Language 2nd edition June 2006 document reference Eiffel language Eiffel method .Hotel Regina Eiffel Paris fashionable 3star hotel 64 rooms Eiffel Tower Trocadéro .Hotel Sublim Eiffel Paris Tour Eiffel hotel contribute make wonderful experience stay Paris .Experience City Light live EarthCams live streaming Eiffel Tower Cam View HD video stream Eiffel Tower feel youre heart .930 2300 900 0000 summer Preparing visit Exploring Eiffel tower Eiffel tower glance News Professionals ? . Eiffel Software downloads page public downloads Eiffel Software products Downloads Choose .Erika Aya Eiffel née Erika LaBrie American woman famously married Eiffel Tower commitment ceremony 2007 founder OS .December November October September EIFFEL SOCIETY Facebook connect EIFFEL SOCIETY sign Facebook today Sign UpLog + .25 Jun 2012 PARIS — 25yearold man Israel jumped death Eiffel Tower thwarting security measures climbing Paris . Eiffel compiler Unix Win32 VMS supports Java threads product ESI Eiffel Software . possibly visit Paris Eiffel Tower visit world famous structure Paris . Eiffel Tower puddled iron lattice tower located Champ Mars Paris Built 1889 entrance arch 1889 Worlds Fair . EIFFEL initiative Support Action SA proposed 7th Framework Programme FP7 EIFFEL SA mobilizing European researchers . Eiffel Development FrameworkTM complete solution Developers choose Eiffel makes job easier increases .Eiffel scholarships information scholarship program Excellence Eiffel Campus France website update 06 192012 . Jardins dEiffel Hotel enjoys locations Paris Invalides district left bank steps Eiffel Tower ChampdeMars . rivets beams glass elevators Eiffel Tower replica Paris Las Vegas encompasses je ne sais quoi French counterpart .25 Jun 2012 Firefighters talk man Eiffel Tower failed jumped levels height .French cuisine picturesque views Eiffel Tower Las Vegas Includes menu . matter technology platform development pain Eiffel Software solution deliver quality software faster .Alexandre Gustave Eiffel December 15 1832 – December 27 1923 French civil engineer architect graduate École Centrale des

Figure 4.4: Abstract summary produced from sentence weighting for search query “eiffel”

4.1.10 N-gram generation in PHP

Sequences of N-grams are discovered through checking if they are all present in the same sentence and if they are consecutive. The algorithm illustrated below iterates over the list of all words `$allWords`. `$allWords` is a 2D array, where the first index points out the sentence id of the word and the second index is the index of the word in its respective sentence. During iteration, keys are generated from consecutive N-grams, and placed in associative 2D arrays.
These arrays have two attributes, the `$count`, which is the number of occurrences of the N-gram sequence and the `$stringId` which is the sentence. There are three arrays for the N-grams, `$ngram2`, `$ngram3` and `$ngram4` which represent 2, 3, 4-grams respectively. If however, a key already exists in the array, its respective `$count` attribute is incremented by one, and the sentence where it occurred is concatenated with the `stringId` attribute.

```
$ngram2=array();//stores 2 grams and their frequencies
 $ngram3=array();//stores 3 grams and their frequencies
 $ngram4=array();//stores 4 grams and their frequencies

 //counting and storing of ngram frequencies
 for($z=1; $z<=sizeof($allWords); $z++){
  for($x=0;$x<sizeof($allWords[$z])-1; $x++){

   $key = $allWords[$z][$x].",".$allWords[$z][$x+1];
   // generating 2-gram keys

   if(array_key_exists($key,$ngram2))  {
    $ngram2[$key]["counts"]++;

    if(!in_array("{\"name\":\"".$string[$z]."\"}"
    ,$ngram2[$key]["stringId"])){
     array_push($ngram2[$key]["stringId"],"{\"name\":\"".$string[$z]."\"}");
    }

   } else  {
    $ngram2[$key]=array('counts'=> 1,'words'=>$key,'stringId'=>
    array("{\"name\":\"".$string[$z]."\"}"));
   }

    }

  }
```

4.1.11 Filtering of Generated N-Grams

After the generation of the N-grams and the calculation of their frequencies, they are filtered out of low priority sequences. Low priority sequences are sequences that have a frequency lower than a certain threshold which is two in this project. Also, all N-gram arrays are then copied into one array called `$ngrams`. The algorithm below shows the process in the bigram array, the same process is also applied to the other N-gram arrays.

```
$ngrams=array();
foreach($ngram2 as $key => $element) {
  foreach($element as $valueKey => $value) {

    if($valueKey == 'counts' ){

      if( $value < 2){
        unset($ngram2[$key]);
      } else {
        $ngrams[$key]['counts'] = $ngram2[$key]['counts'];
        $ngrams[$key]['words'] = $key;
        $ngrams[$key]['stringId']=$ngram2[$key]['stringId'];
      }

    }

  }

}
```

Next, the N-grams that are subsets of other N-grams are removed. The check for subsets is completed through the use of the function `substr_count()`. `substr_count()` is a predefined PHP function which returns the number of times a substring has occurred in a String. Shown in the code snippet below is the process of filtering the bigrams which are subsets of trigrams, the same algorithm is used to remove trigrams subsets from 4-grams.

```
foreach($ngram3 as $key3 => $element3){
  foreach($ngram2 as $key2 => $element2){

    if(substr_count($key3,$key2)){
      unset($ngram2[$key2]);
      unset($ngrams[$key2]);
    }

  }

}
```

Finally, all N-grams are sorted descendingly according to their frequency count using the predefined PHP function `usort` as illustrated below. These N-grams are then displayed to the user as shown in figure 4.5.

2-Grams found
Eiffel,Software(5)
Eiffel,Society(3)
Las,Vegas(2)
Eiffel,Paris(2)
Jun,2012(2)
Measures,Climbing(2)
Bertrand,Meyer(2)
Gustave,Eiffel(2)
Programming,Language(2)
Eiffel,Compiler(2)

3-Grams found
Rooms,Eiffel,Tower(2)
Paris,Tour,Eiffel(2)
Eiffel,Tower,Paris(2)
22,Hours,Ago(2)

4-Grams found
Eiffel,Tower,Thwarting,Security(2)
Tower,Thwarting,Security,Measures(2)
Death,Eiffel,Tower,Thwarting(2)
Jumped,Death,Eiffel,Tower(2)
Man,Israel,Jumped,Death(2)
Israel,Jumped,Death,Eiffel(2)
25yearold,Man,Israel,Jumped(2)

Figure 4.5: Displayed N-grams for search query “eiffel”

```
usort($ngram2,'cmp');
//write the frequencies of 2grams to the html
echo "</br>2-Grams found</br>";
foreach($ngram2 as $result){
  echo $result["words"]."";
  echo "(".$result["counts"].")<br> ";
}
```

4.1.12 Clustering N-Grams

Since the sentence set for each N-gram has been has been compiled, the clustering algorithm is applied. The algorithm works through iterating of the N-gram pool `$ngrams`, and comparing two N-grams at a time using their sentence sets. If there is an intersection, and none of the N-grams are in any subarray in the array`$cluster`, a new subarray is containing these two N-grams is then pushed into `$cluster`. If however, there was an intersection, and one of the N-grams is in a subarray, the other N-gram is pushed into the same subarray.

```
$cluster=array();

foreach($ngrams as $key1 => $element1){
  foreach($ngrams as $key2 => $element2){

    if($key1 != $key2){

      if(array_intersect($ngrams[$key1]['stringId']
      ,$ngrams[$key2]['stringId'])){

        if(multi_search($key1,$cluster)==-2){

          if(multi_search($key2,$cluster)==-2){
            array_push($cluster,array($key1,$key2));
          } else {
            array_push($cluster[multi_search($key2,$cluster)],$key1);
          }

        }

        elseif(multi_search($key2,$cluster)==-2){
          array_push($cluster[multi_search($key1,$cluster)],$key2);
        }

      }

    }

  }

}
```

4.1.13 Visualizaing the Clusters

Using the clustering data made available by the clustering algorithm, a visual cluster tree is drawn using a modified D3.js visual library for a tree layout as shown in figure 4.6. The tree is divided into levels, wherein the root node, i.e., the top level represents the search query entered by the user, in this example, "eiffel". The first sublevel contains the actual clusters of the N-grams, wherein each cluster is represented by the union of the N-grams in that cluster. The clusters are:

1. Programming,Language - Bertrand,Meyer: This indicates that "eiffel" is a programming language which might be created by Bertrand Meyer.

2. Gustave,Eiffel - Eiffel,Tower,Paris: This points out that Gustave Eiffel is somehow related to the Eiffel Tower in Paris.
3. Eiffel,Software - Eiffel,Compiler: Eiffel compiler is a software.
4. Eiffel,Tower,Thwarting,Security - Tower,Thwarting,Security,Measures: This is an interesting cluster because it has detected news articles about a 25 year old Israeli who jumped from the top of the Eiffel Tower.
5. Eiffel,Paris - Paris,Tour,Eiffel - Rooms,Eiffel,Tower: This entry describes the rooms available in hotels near the Eiffel Tower.

The second sublevel decomposes the each cluster into its building N-grams. Finally, the third sublevel shows the snippets where the N-grams occurred.

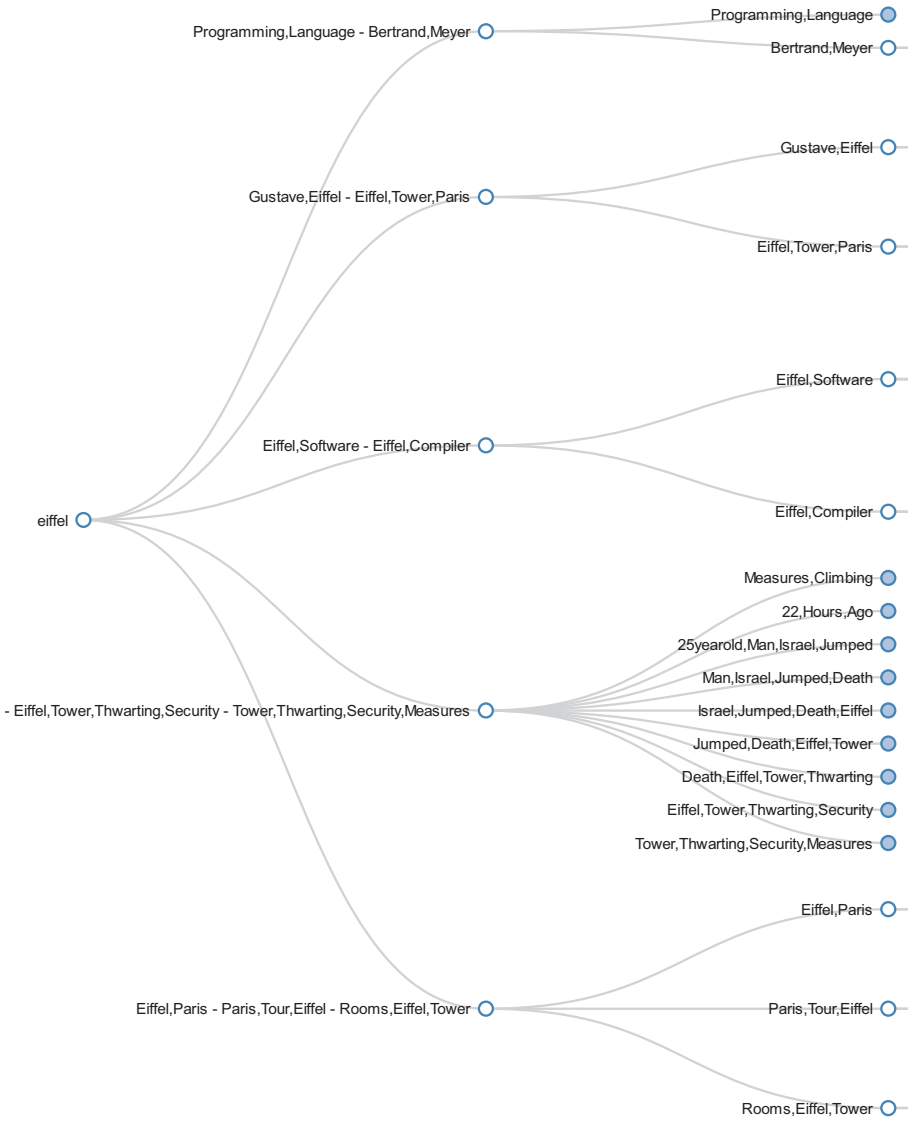

Figure 4.6: Cluster tree of the N-grams for search query “eiffel”

4.2 Evaluation

In this section, the developed web application will be compared feature-wise with other similar tools that perform summarization and clustering. After some researching, none of the online tools discovered had all the features of the web application, so a partial feature review is performed. Unfortunately, none of the reviewed tools show their approach.

4.2.1 SenseBot

SenseBot is a semantic search engine, meaning that it attempts to understand what the result pages are about [5]. It uses text mining to parse Web pages and identify their key semantic concepts. It then performs multidocument summarization of content to produce a coherent summary. It provides the user with bullet points containing relevant topics, with their sources on the web. It also provides a simple tag cloud, which when any item is clicked, tries to focus the search on that item under the search topic the user originally entered.
In summary, SenseBot provides a tag cloud and a textual summary with links about a certain query.

4.2.2 Knowledge Graph - Google

Knowledge Graph is the latest refinement to Google's search engine product that seeks to provide users with more relevant and in-depth responses to search queries. Along with the standard search results you're used to seeing, Google's search results page now displays instant results related to the user queries - a search for Taj Mahal immediately brings up a list of facts, photos, and a map of the famous landmark, as well as quick links to other popular uses of the search term (like the musician or the casino in New Jersey). There are a multitude of sources behind this data - Google cites Freebase, Wikipedia, and the CIA World Factbook, and other commercial datasets, but also notes that "it is augmented at a much larger scale" and tuned based on what the average user searches for. As of now, the knowledge graph database currently holds information about 500 million people, places and things. More importantly, though, it also indexes over 3.5 billion defining attributes and connections between these items.
Searching for "Eiffel" on the knowledge graph quickly displays a list of facts, photos, and a map of the Eiffel tower. It also provides the user with a link to "Gustave Eiffel". The Knowledge graph does offer clustering as seen previously, but is yet limited to what is currently in its databases. With more time, that database might be expanded to provide more clustering to the search term.

Chapter 5

Conclusions

5.1 Conclusion

Since the advent of search engines, the information overload problem has been, to a certain extent, alleviated. Search engines allowed users to reduce the information overload by allowing them to perform a centralized search. However, another problem arises, too many web pages are returned to the user upon searching for a single query. The user often has to examine tens and hundreds of pages to find out that only a few of them are relevant.

The goal of this project was to automate the summarization of google search results. And to visualise the outcome from the semantic analysis performed by CHR and PHP.

The work done in this project is as follows, Google search results were extracted using scrapping. Stop words were removed by PHP, and the processed text was sent to Prolog. CHR rules in Prolog were used to compute the frequency, indexing, and the relativity between words. These computations were sent back to PHP to be visualized and used in other calculations such as generating the N-grams, calculating the sentence weight, and clustering the N-grams. All of this analysis is presented to the user using a variety of visualizations, such as text based, graph based and tree based.

5.2 Future Work

In this section, future enhancements are presented by improving some of the current features.

5.2.1 Optimization and Parallelization of CHR rules

When faced with large amounts of data for analysis, the project's CHR implementation took an increasing amount of time. In order to enhance the speed of CHR execution,

first, optimizations to the code must be considered. If optimizing did not improve execution time, multi-threading could be implemented for better user interaction with the application.

5.2.2 Improving the similarity measure between sentences

Since the similarity measure is reliant on character order when comparing strings, a new enhanced version should be implemented where its not affected by the word position in the sentence. This would give probably higher similarity percentages between sentences allowing their removal or merging with others.

5.2.3 Enhancing the Produced Textual Summary

Instead of displaying all the search result snippets in a paragraph ordered in descending order according to their weights. Merging of mostly similar sentences should be performed. Also, some natural text language processing abstraction based summary could be implemented to provide much shorter, yet more informative summaries, rather than concatenated sentences produced through extraction.

5.2.4 Improving the User Interface

The web application does contain a lot of statistics in a text based format, so the text must be reduced through the use of charts, graphs or other visualization techniques for better traversal of the data presented.

References

[1] "Google Hot Searches, "http://www.google.com/trends/hottrends, May 2012.

[2] Jon Sneyers, Peter Van Weert, Tom Schrijvers, and Leslie De Koninck, "As time goes by: Constraint Handling Rules A survey of CHR research between 1998 and 2007," TPLP, 10(1):147, 2010.

[3] Thom Frühwirth, "Constraint Handling Rules," Cambridge University Press, 2009.

[4] Thom Frühwirth, "Introducing simplification rules," Technical Report ECRC-LP-63, European Computer-Industry Research Centre, Munchen, Germany, October 1991.

[5] "Sensebot - The semantic search engine that finds sense on the web," http://www.sensebot.net, March 2012.

[6] G.Chowdhury, "Natural language processing," Annual Review of Information Science and Technology, pp. 51-89, 2003.

[7] E. Liddy, "Natural Language Processing," In Encyclopedia of Library and Information Science, 2nd Ed. Marcel Decker, Inc., 2003.

[8] "Introducing JSON," www.json.org, April 2012.

[9] "The Atom Syndication Format," http://tools.ietf.org/html/rfc4287, June 2012.

[10] "cURL and libcurl," http://curl.haxx.se, April 2012.

[11] "D3.js - Data-Driven Documents," d3js.org, May 2012.

[12] "What are Google Alerts? Alerts help," https://support.google.com/alerts/bin/answer.py?hl=en&answer=175925&topic=28415&parent=28413&rd=1, May 2012.

[13] J. Hutchins, "Summarization: Some Problems and Methods," Proc. Informatics 9: Meaning-The Frontier of Informatics, K.P. Jones, ed., Aslib, London, pp. 151-173, 1987.

[14] U. Hahn, I. Mani, "The Challenges of Automatic Summarization," IEEE Computer, 2000.

[15] H. P. Luhn, "The automatic creation of literature abstracts," IBM Journal of Research Development, 1958.

[16] Andre F.T. Martins, Dipanjan Das, "A Survey on Automatic Text Summarization," Language Technologies Institute , Carnegie Mellon University , November 21, 2007.

[17] P. Baxendale, "Machine-made index for technical literature - an experiment," IBM Journal of Research Development, 1958.

[18] Ian. Oliver, "Programming Classics: Implementing the World's Best Algorithms," Prentice Hall, 1993.

[19] Jose A. Carillo, "English Plain and Simple," The Manila Times, March 9, 2004.

[20] O. Zamir, "Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results," PhD thesis, University of Washington, 1999.

[21] Anupam Joshi, Zhihua Jiang, "Retriever: Improving Web Search Engine Results Using Clustering," Managing Business with Electronic Commerce: Issues and Trends, pp. 59-81, 2002.

[22] Marc M. Sebrechts, Joanna Vasilakis, Michael S. Miller ,John V. Cugini, Sharon J. Laskowski, "Visualization of Search Results: A Comparative Evaluation of Text, 2D, and 3D Interfaces," SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, pp. 3-10, 1999.

[23] "PHP Manual," `http://www.php.net/manual/en/preface.php`, March 2012.

[24] "SWI-Prolog," `http://www.swi-prolog.org`, March 2012.

[25] Cavnar, W. B. and Trenkle, J. M., "N-gram-based text categorization," In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, pp. 161175, 1994.