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

Ulm University
Institute of Software Engineering and Compiler Construction

CHR-based Text Mining and Classification of Google Search Results

Bachelor Thesis

Author:	Aly Saleh
Supervisor:	Prof. Thom Frühwirth
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.

Aly Saleh
13 July, 2012

Acknowledgments

I would like to thank all those who have helped carry out this thesis for their support, guidance and encouragement. Without them, this thesis would not have been completed. I would like to show appreciation and gratitude to those people below.

- My **mom**, **dad** and my **brother**, for their continuous motivation and encouragement, and for always being there for me.
- **Amira Zaki**, my co-supervisor, for her follow-up and continuous support, guidance. For her tremendous effort all over this semester and for always being there, in matters of education and life, and constantly and patiently encouraging us.
- **Prof. Dr. Thom Früwirth**, my supervisor, for accepting me to do my bachelor thesis with him, and for his great advices, support and guidance throughout this semester.
- My **friends** in Ulm, for their great support and company.

And above all, I would like to thank God, for helping me finish my bachelor and for getting this far.

Abstract

The paper describes a tool which summarizes and classifies Google search results using Constraint Handling Rules (CHR). The tool extracts data from the Google API and combines data extraction, linguistic analysis and web development techniques. Linguistic analysis was implemented using CHR and includes counts of important words, non sequential and sequential patterns, known as sequences. Moreover, PHP and JavaScript were used for applying web development techniques. The tool features clustering, ranking results and viewing the sequences and important words' counts as charts. The tool simplifies finding specific query, especially if the query has several synonyms.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, Google is one of the most popular search engines globally. It has been online for more than fourteen years. It is the number one most visited website according to Alexa [3] and it is visited monthly by 175 million unique viewers in the US [4]. Exploiting Google power, one can use Google search results to build an informative and comprehensive summary. Moreover, one can use Constraint Handling Rules (CHR) which is a declarative programming language following the constraint based programming paradigm for parsing and analysing Google's search results and for distinguishing similar patterns.

1.1 Motivation

Harnessing Google power, one can build informative summaries that try to comprehend and cluster Google search results. Summaries are essential to declare the important details and remove unnecessary information. It is also important to grasp the main points within a story. Furthermore, by clustering the search results, one can classify patterns into groups (clusters), in which every group of related results are viewed together and separately from the other groups.

1.2 Aim and Problem Statement

Our aim is to develop a web tool that mines and classifies data extracted from Google search results, using CHR as a base for its implementation. Combining content of different search results and try to cluster them. Search results can have its reference links presented in the text summary. Moreover, words which appear the most among all the search results can be viewed differently, also data can be analysed to generate statistics about words and sentences. There are three issues that have to be addressed to implement this tool:

1. Searching for the best way to extract data from Google.
2. Performing linguistic analysis such as counting word frequencies and detecting repetitive sentences using CHR.
3. Trying different web development techniques to improve the tool.

1.3 Outline

This thesis is classified into five chapters. This chapter contains a brief introduction, motivation, aim and problem statement. In chapter 2, the background knowledge that influenced the approach decisions is discussed. Chapter 3 deals with the approach taken to solve the main issues defined in the problem statement. The fourth chapter explains the detailed implementation of the approach taken, also it contains a users guide section and a tool evaluation and limitation section. Finally, chapter 5 has a conclusive summary about this work and the tool, it also gives suggested ideas about future work that can be done on this tool for more enhancement.

Chapter 2

Background

2.1 CHR

Constraint Handling Rules (CHR) is a declarative programming language designed by Frühwirth in 1991. It is a high-level, concurrent committed-choice and constraint based programming language that was used previously as a specific purpose language for solving constrained based problems, but now it has developed to solve broad types of problems, to serve general concerns. CHR is embedded in host programming languages as Java and Haskell, but the most commonly used implementation is Prolog implementation. A CHR program typically consists of a set of rules that transform constraints until they are solved.[5].

2.1.1 Syntax

Constraints

CHR contains two types of constraints: built-in constraints and CHR constraints(user-defined constraints). Built-in constraints are predefined in the host language, or imported CHR constraints from other modules. CHR constraints on the other hand, are in the current CHR program and defined by CHR rules.

Rules

CHR has three types of rules: simplification rules, propagation rules and simpagation rules.

- **Simplification rule:** Replaces the already existing constraints with simpler ones, thus causing reduction to the problem.
- **Propagation rule:** Adds new constraints to the already existing ones to clarify and give more information about the problem, which may cause further simplification.
- **Simpagation rule:** Combines both simplification and propagation. It separates the head of the rule into two parts containing constraints, where the constraints on the left of the backslash \ are propagated and the ones on the right are simplified.

```
SimplificationRule: Name @  Head <=> [Guard |] Body
PropagationRule:    Name @  Head ==> [Guard |] Body
SimpagationRule:    Name @  Head \ Head <=> [Guard |] Body
```

Where *Name* is an optional unique identifier, the *Head* is one or more CHR constraints, the *Guard* is built-in constraints and the *Body* is the *Goal*, where a *Goal* is a query that contains a mixture of built-in constraints and user-defined constraints separated by commas [5].

```
Head --> CHRConstraints
Guard --> BuiltInConstraints
Body --> Goal
```

2.2 Text Mining

Text mining is the process of analyzing, extracting information and discovering important patterns in a text document. It has very high commercial value which makes high-tech companies interested in text mining. A recent study shows that 80% of companies information is stored in text. Text mining can be considered as an extension for data mining, since text mining deals with unstructured data and involves multiple fields such as information retrieval, clustering, categorization, visualization, database technology, machine learning and data mining. Data mining on the other hand deals with structured data [6].

2.2.1 General Text Mining Framework

A general text mining framework has been presented which consists of two components

- Text refining which is the process that transforms text into an intermediate form (IF). The IF can be document based where each entity represents a document such as clustering, or concept based where each entity represents a concept of interest. Mining concept based IF obtains patterns and relationships. However, document based IF can be transformed to concept based by extracting important information according to concerns of a certain domain.
- Knowledge distillation which concludes patterns and knowledge from the intermediate form.

2.2.2 Text Mining Products

Text mining products are also classified into two types of products.

- Document visualization, its general idea is to gather similar documents in clusters and show them in a certain graphical interface. Many products fall into this category.
- Text analysis and understanding, which is based on natural language processing and includes text analysis, categorization, information extraction and summarization.

2.2.3 Challenges

To reach the IF, semantic analysis need to be performed, which is computationally expensive and executes few words per second. It remains a challenge to come up with more efficient algorithms to perform semantic analysis. Moreover, there is a significant language component contributing in text mining, which makes reaching a language independent IF a hard task. Also, if the knowledge domain was known previously, parsing efficiency could be improved in the early stages to reach a more compressed IF.

2.3 Text Summarization and Document Clustering

The tool introduced in [7], performs two main tasks that utilize beneficial concepts: document clustering and text summarization. Some of these concepts influenced this work. The first concept, namely clustering, is the task of assigning a set of objects into groups so that objects of the same cluster are more similar. It is a common technique for statistical data analysis and is used in many fields including machine learning, pattern recognition and information retrieval.
Moreover, when mining text documents, some difficulties arise, handling synonyms (different words with same meaning) and hynonyms (words with same spelling but different meaning) is one of these difficulties. Also, there is a large number of words that exist in a text document, which makes it more difficult for any text mining algorithm. Therefore, pre-processing methods should be applied to reduce the number of words. The tool presents some solutions for this issue, such as case folding, removal of uninformative words, stemming and n-grams. Case folding is the process which alter all characters of a text document into the same case, either upper case or lower case. Uninformative words removal is withdrawing all the words which appear many times in a text document and do not contribute much in the document content where they appear. Stemming which is the process of converting each word to its stem, removing suffixes and verbal/plural inflections. Finally, N-grams are a part of a longer string, for example, `_DA` is a tri-gram for `DATA` where "`_`" represent a leading or a trailing space, n-grams do not require linguistic preparations and is not sensitive to grammatical errors, however, it is less effective than stemming and stop words removal in decreasing the number of words.
The tool also uses a summarization algorithm which is based on the most relevant sentences in a document. A sentence is relevant if it has a high average relevance of all words in the pre-processed sentence.

2.4 Web Development

2.4.1 PHP

PHP is a general purpose server-side scripting language used for web development. It is a recursive acronym for "Hypertext Preprocessor". PHP files contain PHP code which is embedded inside HTML "Hyper Text Markup Language". Moreover, the PHP code is surrounded by special start and end characters to jump into the PHP mode. The start characters are `<?php` and the end characters are `?>`. PHP also supports major operating systems and most web servers. Moreover, PHP supports lots of databases, which makes building a database-enabled website very easy. PHP is capable of outputting a lot of formats such as images, PDF files and even flash videos. Moreover, PHP includes many text processing capabilities. To use PHP for server-side scripting, three pre-requisites are needed:

- PHP parser.
- Web server.
- Web browser.

After running the web server with a connected PHP installation, one can view the output of a PHP program in the web browser through the web server [8].

2.4.2 JavaScript

JavaScript is a client-side interpreted object-based programming language and it is used and supported to a great extent. It is a client-side programming language since it runs on the client computer. Variables in JavaScript does not have types since JavaScript is a weakly typed programming language. One advantage of the weakly typed programming languages is that programmers do not need to specify variable types, which makes it easier on them, since the compiler or the interpreter does the type conversion. JavaScript code is embedded in HTML between special tags. The code begins with `<Script language = "JavaScript">` tag and end with `</Script>` tag. The client browser interprets and runs the JavaScript code [9].

2.4.3 API

API stands for Application Programmable Interface. An API is typically an interface which contains set of functions and methods that can be used in ones project to extend a certain application or an operating system and communicate with other software components [10].

2.4.4 JSON

JavaScript Object Notation (JSON) is a text format for representing structured data. Four primitive and two structured types can be represented by JSON. The primitive types are string, numbers, boolean and null. The structured types are arrays and objects. Many applications written in many programming language use JSON for data exchanging [11].

2.4.5 Atom

Atom is a format based on the XML language, which is the Extensible Markup Language. Atom is used for describing lists of related information known as “feeds”. The main use of Atom is to provide web feeds including web-logs and news headlines [12].

2.4.6 Highcharts

Highcharts is a library for charts written in JavaScript. Highcharts enable web developers to add interactive charts to their websites easily. Moreover, it is compatible with all modern browsers and it is free for all non-profit organizations. Highcharts has many features; numerous chart types, tool tip labels, dynamic, exporting and printing, text rotation for Labels and zooming. See figure 2.1 for an example [13].

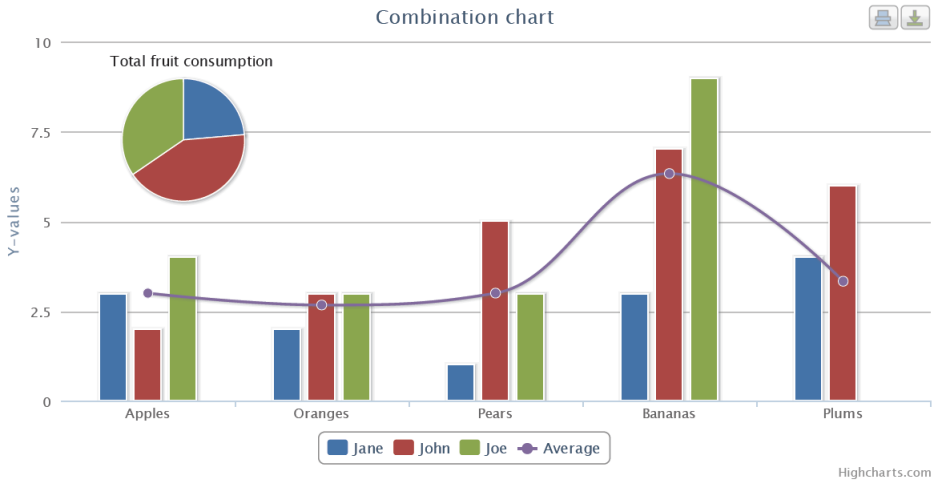

Figure 2.1: Highcharts combinational chart example, example from [1].

2.5 SWI-Prolog

SWI-Prolog is an open source implementation for Prolog. It is used for implementing in logic programming paradigm and experiment interactions with other programming paradigms. SWI-Prolog lacks optimization, since it focuses on two main objectives: portability(SWI is written in C and Prolog) and modifiability. It has many features such as multi-threading, Graphical user interface, interface with Java, IDE and libraries for constraint logic programming. SWI-Prolog has a specific library for CHR. The CHR system of SWI-Prolog is the K.U.Leuven CHR system, see chapter seven of [14]. SWI-prolog is available on Windows, Unix and MACOSX [14] [15].

Chapter 3

Approach

Accomplishing the aim to develop a summarization tool, starts with solving the three main issues stated in the introduction, which were

1. Searching for the best way to extract data from Google.
2. Performing linguistic analysis using CHR.
3. Improving the tool by trying different web development techniques as algorithms and visualizations.

In this chapter, a description of the approach taken to tackle these three problems can be found.

3.1 Data Extraction

According to Google terms and conditions [16], the only legal way to extract data from Google is through its interface (API). However, sending a request to Google API and receiving the response can be done in different ways:

- Using programming languages.
- Using web page.

3.1.1 Programming Languages VS Web Page

Both of the previously mentioned ways will retrieve Google results. However, since the desired tool is a web tool, using a web page is more preferable. Moreover, using a web page has many advantages over other programming languages. For this work, we are interested in two of these advantages; the parser and Google Custom Search Engine.

Parser

Using a programming language will require implementing a parser since the received data is in form of Atom, JSON or plain text, which can be an overload. On the other hand the web page does not need a parser since the results are received as an object which contains the contents of each result, see object representation in 3.1.

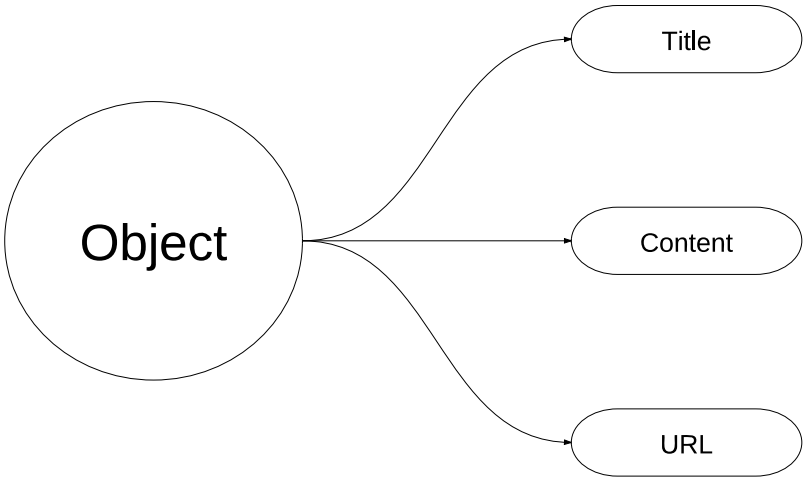

Figure 3.1: Drawing to show the object returned from Google, drawn by [2].

Google Custom Search Engine (CSE)

Google Custom Search API helps retrieving and displaying search results through Google Custom Search Engine, which is a customized search engine that has a control panel. It searches on specific websites in which the administrator specifies. However, the CSE can be adjusted through its control panel to search the whole web but these results are unlikely to match Google Web Search results. Google CSE control panel also allows its administrator to generate the code as a JavaScript code according to the desired features chosen from the control panel [17]. The CSE returns a maximum of 10 result pages. To create a CSE, first one must create a Google account, then go to `www..google.com/cse` and create a search engine specifying its features. Afterwards, the code for the search engine will be generated. Some of the control panel features are:

- View statistics about what people are searching for in the CSE.
- Collaboration by inviting another developers to contribute in the search engine.
- Enabling auto-completeness in the search bar.
- Expanding users search query by adding synonyms for the search query. For example, when the user searches for “car” results for “car” and “auto” are returned.
- Displaying returned results and enabling administrator to change layout and style.

3.2 Linguistic Analysis and Sequences

3.2.1 Frequent Words

Frequent words or important words are words which appear frequently in a text document. In order to realize these frequent words, one needs to count each word in this text document then filter these counts according to a certain threshold.

3.2.2 Sequences

A sequence is an ordered, non-empty set of items [18]. To detect sequences, three steps are presented:

1. Every N consecutive words represent N-word sequence. Hence, one must keep track of words' order and for every N consecutive words, N-word sequence should be generated.

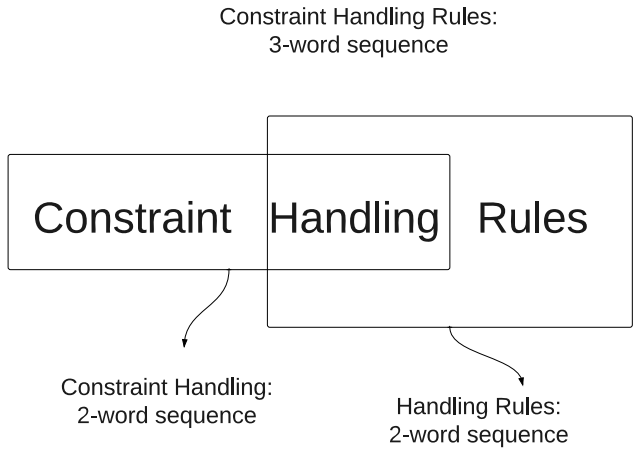

Figure 3.2: Example on sequences, drawn by [2].

2. After generating the N-word sequences, merging identical sequences into one sequence and increasing the appearance count of the generated merged sequence, is essential. Thus, disabling same sequence occurring more than once.

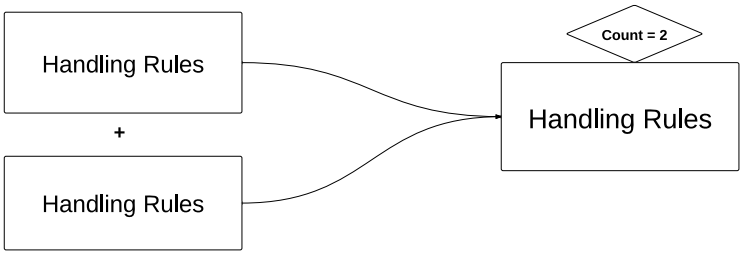

Figure 3.3: Merging two sequences into one, drawn by [2].

3. Finally, filtering sequences which have a small appearance count leaving only important sequences having large counts. These sequences are filtered according to a defined threshold number.

3.2.3 Words Gravity

Words gravity or relativity is the strength between the non sequential words, generating a sequence of these non sequential words. However, the words should be in the same sentence. Moreover, the gravity strength is set according to the distance between the chosen words. Afterwards, these sequences are summed up and filtered out as the previous sequences. The algorithm for the calculating strength is:

Algorithm 1 Gravity Algorithm

Given two words in the same sentence
For every $word_i$, $word_k$ and $k > i$, where k and i are the indices of the words
Gravity between $word_i$, $word_k = \left(\frac{1}{2}\right)^f$
where f = the number of words between $word_i, word_k$

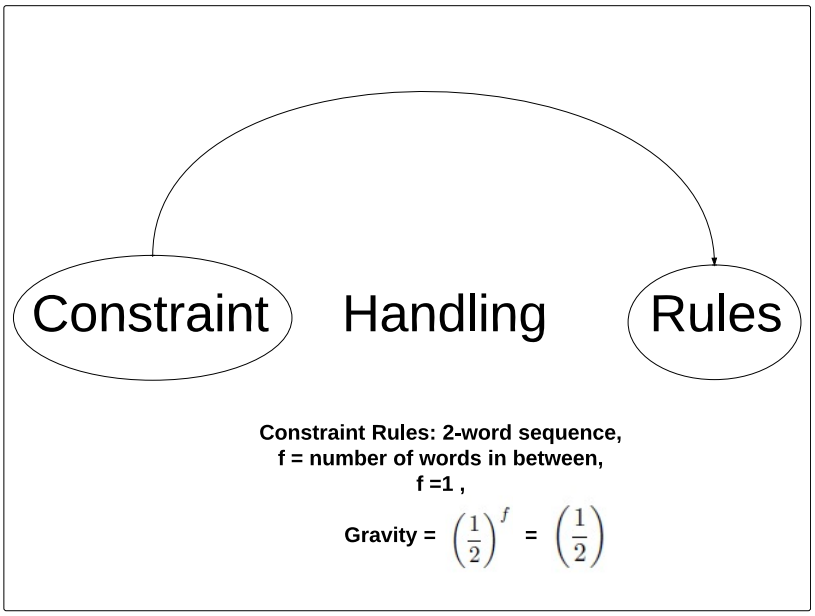

Figure 3.4: Example on gravity and how it is calculated, drawn by [2].

3.3 Web Development

By using one or more scripting language, several algorithms and visualizations are introduced to simplify viewing important words and sequences, and to cluster data into different groups according to their similarity.

3.3.1 Case Folding

Case folding is the process which alter all characters of a text document into the same case, either upper case or lower case. For example, the words “car”, “Car”, “cAr”, “caR”, “cAR”, “CaR”, “CAr”, “CAR” should be all altered to the word “car” if its a lower case folding or “CAR” if its an upper case folding [7]. Case folding is essential for pattern matching, because trying to match “Car” with “car” would not match despite the insignificant difference.

3.3.2 Stop Words Removal

Stop words are the words which appear many times in a text document and do not contribute much in the document content where they appear. The words “the”, “can”, “will” are examples for the stopping words. Removing this stopping words will decrease the document size which will in tern increase execution speed and efficiency. Moreover, the number of non informative patterns and counts will decrease significantly [7].

3.3.3 Charts and Highlighting

Viewing important words or sequences as bar charts, where each bar represents a word or a sequence, and their count is indicated by the bar’s length. Moreover, by selecting the bar, the word or sequence that corresponds to this bar will be highlighted in the original text.

3.3.4 Scored Results

Scored results is an algorithm to find the score of each result according to the number of important words which appeared in the result. The importance of a word is calculated according to how frequent it appeared in the search results. Afterwards, results are sorted according to this score in descending order to facilitate viewing more important results first. The algorithm to implement the scored results is as follows:

Algorithm 2 Scores Algorithm

Given That R holds the search results, S holds the result score which is initially equal zero, W holds the important words and C holds the important words’ counts.

For each $< R_i, S_i >$
  For each $< W_j, C_j >$
    If $(R_i.contains(W_j))$
      then $S_i \leftarrow S_i + C_j$
    End If
  End for
End for

3.3.5 Results Clustering

Classifying search results into small groups which differentiate between different connotations. Assume that we have the list of search results indices where each important word appeared in, e.g. the word “help” appeared in result `{64,63,40,4,12}`. The

numbers in the list indicates the indices of the search results. Then, after giving each pair of an important word (W) and set of result indices (R), a unique number representing a cluster (C), where W,R and C represent a tuple `<W,C,R>`. The algorithm starts by comparing pairs of tuples, through calculating the intersection between the set of result indices (R) in one tuple with all the other tuples; the pair with maximum intersection is chosen and is given the same group numbers (C). For instance, the tuple `<W1,1,{1,2,3,4,5}>` is in cluster group one and the tuple `<W2,2,{2,3,4,5,6}>` is in cluster group two, if they have maximum intersection then both of them will be assigned to group one, always switching cluster numbers with the smaller number to ensure correctness of the algorithm. If we used switching with larger group number failures could happen. For example, if there are three tuples `<X,C1,R1>`,`<Y,C2,R2>` and `<Z,C3,R3>` and they have group numbers `C1`, `C2` and `C3` respectively, where `C3>C2>C1` and we figure out that `R1` and `R2` have maximum intersection and should have the same group number, then their group numbers will be changed to `C2` since `C2` $>$ `C1`, therefore, the tuples will be changed to `<X,C2,R1>` and `<Y,C2,R2>`. Then, if `R2` has maximum intersection with `R3` and should also have the same group number, then their group numbers will be changed to `C3` to have `<Y,C3,R2>` and `<Z,C3,R3>`. This is wrong because they all should belong to the same group. Because if `X` has maximum intersection with `Y`, then they should have the same cluster number even if `Y` changed afterwards, same for `Y` and `Z`. However, switching with the smaller number will not cause this hazard because all of them will have group number `C1`. The algorithm is as follows:

Algorithm 3 Clustering Algorithm

Given that C is the cluster group, W is the important word, R is the set of search results indices.

All distinct C_i
Important word tuple $\leftarrow < W_i, C_i, R_i >$
  For each $< W_i, C_i, R_i >$
    get from all other tuples an important word tuple
    such that $< W_j, C_j, R_j >$ *has* $\mathbf{max}\,|R_i \cap R_j|$.
  End For
set C_i *and* C_j $\leftarrow \mathbf{min}(C_i, C_j)$

Chapter 4

Implementation

In this chapter, the tool implementation details are explained by applying the algorithms described in chapter 3 using the technologies presented in chapter 2. Moreover, the chapter includes a user guide section in addition to a tool evaluation and limitation section. Please note that, mapped arrays are arrays which have their elements corresponding to each other. Meaning, the first element of the first array has related data to the first element in the second array.

4.1 Implementation Outline

Initially search results are extracted using the data extraction code and then these search results are sent to PHP to remove the stopping words and case fold the text characters. Afterwards, PHP writes the text without stopping words and case folded in a text file, then calls SWI-Prolog using command prompt running a specific predicate. SWI-Prolog will run the Prolog predicate and the CHR rules returning the CHR output to PHP again. Eventually, PHP will perform some algorithms and visualizations to output the summary. This chapter discusses each step in more detail. Figure 4.1 gives an overview of the tool implementation architecture.

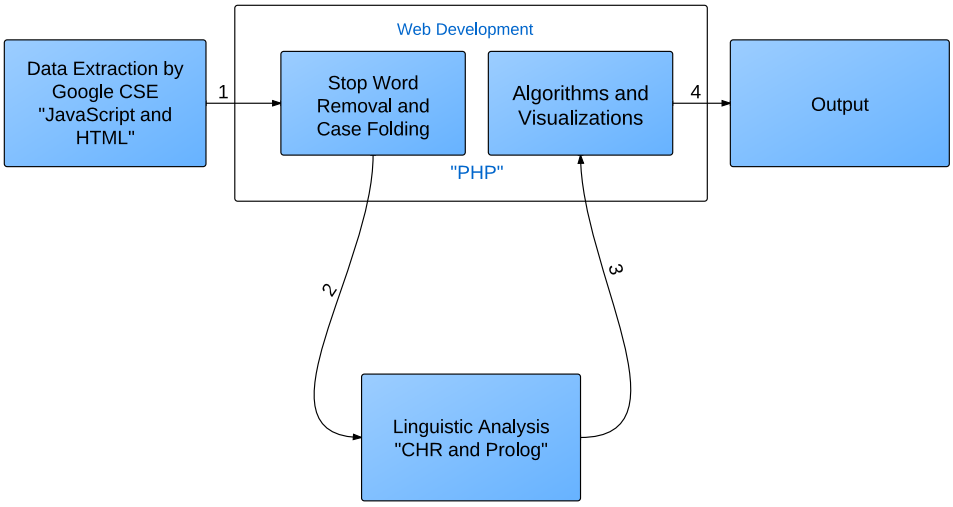

Figure 4.1: Code flow chart, drawn by [2].

4.2 Data Extraction Implementation

First, one needs to load the search engine and draw the search page, to enable the user to search a specific query returning one page of search results. Drawing the search page means having the search bar and a search button in the page as in figure 4.2, which is done automatically by Google CSE. However, to have a conclusive and informative summary one must have more than one result page, thus one must request more pages programmatically. Afterwards, some text manipulation is done and then the data is sent to PHP for data analysis and web development.

Figure 4.2: A figure showing the drawn Google search page.

The following code snippet is generated by Google CSE used to load and draw the search page and initialize custom search and the custom search options, enabling the auto-complete in the search bar. Also, linking the search engine with the online control panel. The search bar is drawn in a division with id `cse`.

```
<script>
google.load('search', '1', {language : 'en',});
google.setOnLoadCallback(function() {
    var customSearchOptions = {};
```

```
    var customSearchControl = new google.search.CustomSearchControl(
    '002978428225665678344:-pyzpchfj_w', customSearchOptions);
    customSearchControl.setResultSetSize
    (google.search.Search.LARGE_RESULTSET);
    var options = new google.search.DrawOptions();
    options.setAutoComplete(true);
    customSearchControl.setAutoCompletionId('002978428225665678344:-
    pyzpchfj_w+qptype:1';
    customSearchControl.draw('cse', options);
    }, true);
</script>
<div id="cse" style="width: 100%;">Loading</div>
```

The next three lines are manually added to the previous script code for different reasons. The first line calls back the function `searchComplete` after the search results for each page are returned. The last two lines bind some global variables to the search objects for future manipulation

```
customSearchControl.setSearchCompleteCallback(this, searchComplete);
s = customSearchControl.getWebSearcher();
customSearch=customSearchControl;
```

Requesting more pages programmatically is done in this function `searchComplete`. The function simply saves the search results in an array of objects then checks; if the number of pages is less than 10 then add the search results to the array and go to the next page, else call a function `process` which sends the data to PHP. The new page is requested by calling a predefined function from the Google search object which was bounded previously. The variable page is initialized to one because page zero results are retrieved in the initial search call done by the user.

```
var page = 1;
var resultsCounter = 0;
function searchComplete()
{
  if(s.results && s.results.length > 0)
    {
      var results = s.results;
      for(var i = 0; i < s.results.length; i++)
      {
        allResultsArray[resultsCounter] = s.results[i];
        resultsCounter++;
      }
      if(page < 10)
      {
        s.gotoPage(page);
        page++;
      }
        else
        {
          process();
        }
    }
}
```

The `process` function loops over the results and removes some unwanted characters using a predefined JavaScript string function `replace`, then puts the results content and the corresponding links alternatingly in a string. Putting between the contents and links some special characters to differentiate between them. Finally, submitting a form which sends the data to a PHP page and also sends the search query to the page.

```
<script>
function process()
{
  for (var i = 0; i < allResultsArray.length; i++)
  {
    if(allResultsArray[i] != null)
    {
      var x = allResultsArray[i].content;
      x=x.replace(/<b>/gi,"");
      x=x.replace(new RegExp("","gi"), "");
      allResults =allResults + x + "******";
      allResults =allResults + allResultsArray[i].url+ "******";
    }
      else
      {
         break;
      }
  }
// Submitting the form and sending data to PHP.
document.getElementById('allResults').value=allResults;
document.getElementById('searchQuery').value=
customSearch.getInputQuery();
document.getElementById('form').submit();
}
</script>
<form id="form" method="post"
action="http://localhost/phpmyadmin/Clusters.php">
<input type="hidden" id="allResults" name="allResults">
<input type="hidden" id="searchQuery" name="searchQuery">
 </form>
```

4.3 Stop Words Removal and Case Folding (Web Development I)

4.3.1 Preparations

Firstly, the search results sent from the data extraction process have to be read. Then, one can split the data sent to have the results and the links in an array. Search results and links will be alternating in this array. Each search result has its link in the following element of an array. However, splitting the original text had to be done twice; one before removing the stop words and one after it, taking the links from the first split output and the search results from the second split output, because if the stop words were removed before splitting, links will be destroyed, and if it was done after splitting, then the results array had to be concatenated again to a string to perform the stop words

removal. Having the results and links alternating in an array, one can distribute them into two mapped arrays as well as case folding the results into a third mapped array to have the original search results and the case folded ones. The function used for splitting the results is a predefined function `explode`. The detailed implementation can be found in A.1.

4.3.2 Stop Words

Stop words removal is done by searching for these stop words and replacing them with an empty string. Using the predefined PHP function `preg_replace` which searches for a subject that matches a specific pattern and replaces it with a replacement. The function `implode` used to surround the word with `\b` and `/i`. This creates a regular expression. The `b` tags indicates a word boundaries. For example, the word "web" is matched and not a word partial as "webbing" or "website", the `i` indicates case insensitivity. The used PHP library was brought from an online website for code snippets, then some other words were added experimentally.

```
// The library is trimmed because its too big.
$commonWords = array('the','can','an','a',....);
$name0=preg_replace('/\b('.implode('|',$commonWords).')\b/i','',$name0);
```

4.3.3 Case Folding

Simply case folding is done by a predefined PHP string function `strtolower` which converts characters to lower case.

```
$results[$resultsCounter] = strtolower($splitted[$counter]);
```

4.4 Linguistic Analysis

Linguistic analysis is done using CHR and Prolog. When PHP calls SWI-prolog from the command prompt, a specific Prolog predicate starts executing. Afterwards, CHR rules are fired until there is no more rules to fire. Then an output is returned.

4.4.1 PHP Preparations

For PHP to call SWI-prolog, the text file which have the search results case folded has to be ready, because SWI-Prolog will read this file to perform the linguistic analysis on it.

Writting In a Text File

To write in a file, one must specify its path. Then to open it, `fopen` which is a predefined PHP function is used. Afterwards, iterating over the case folded results and writing them in order. A predefined PHP function `fwrite` is used to write the results in the file.

```
$myFile =
"C:\\Dokumente und Einstellungen\\Alexander\\Desktop\\text.txt";
$fh = fopen($myFile, 'w') or die("can't open file");
for ( $counter = 0; $counter < sizeof($results); $counter += 1)
{
  fwrite($fh, "$results[$counter]");
  results)
}
fclose($fh);
```

Calling SWI-Prolog

Calling SWI-Prolog is done via command prompt. PHP has a predefined function that executes a command and return the output of executing this command. The function is called `shell_exec`. It takes a command as an input. The command used to run SWI-Prolog consists of several parts; the SWI-Prolog path, path of the file to consult, name of the predicate to execute, size of the global and local stack and eventually "halt" which means close SWI-Prolog after returning the output.

```
$cmd = "C:\\Programme\\pl\\bin\\swipl.exe -f C:\\PrologTest.pl
-g test -L128m -G32g ,halt";
// Explaning the command by example:
// Path for SWI-Prolog (C:\\Programme\\pl\\bin\\swipl.exe)
// which file to consult (-f C:\\PrologTest.pl)
// which predicate to execute (-g test)
// increasing local and global stack ( -L128m -G32g).
// halt is to close SWI-Prolog.
$output = shell_exec($cmd);
```

4.4.2 Prolog and CHR Implementation

In order to detect sequences using CHR, one should have the text file represented as ordered constraints. Meaning, each word in the text will be represented as a constraint, in which each word have an identifier for its place in the sentence and an identifier for the sentence in the text as shown in figure 4.3. Afterwards, sequences can be detected easily using CHR. For this tool, two-word sequences, gravity and words count are used. However, sequences till the four-word sequences are implemented.

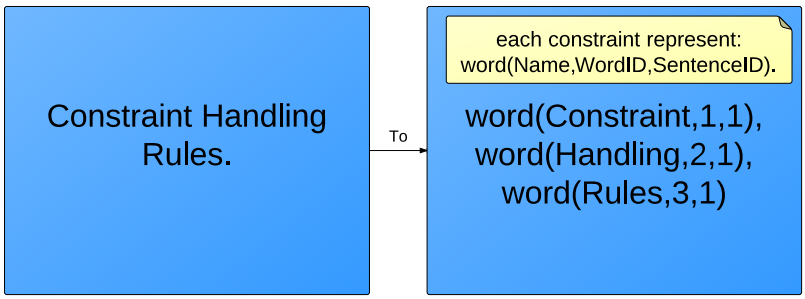

Figure 4.3: Changing the text to ordered constraints, drawn by [2].

Preparations

Before converting the text into ordered constraints, the data sent form PHP has to be read. Prolog has a predefined predicate called `open` which takes as an input the path of the file to be read and output the text file as a stream. This stream is used by another function called `read_file_to_codes` which takes the stream as an input and output a list which have all the characters in the file represented in ASCII codes. Afterwards, the function `removePunc` is invoked on this list to remove some unwanted characters. The function simply takes the list of characters as an input and output the same list without the unwanted characters. Preparations code in A.2.

From Text to Ordered Constraints

After having all the characters in the text file in one list, it can be divided into sentences to have each sentence as a sub-list. Moreover, each sentence can be divided into words to have each word as a sub-sub-list. This results in a list of sentences, where in tern each sentence list contains list of words

First dividing the list that contains all characters to sentences. The following predicate `cTs` simply iterates over the list of characters till it finds "." followed by a space, or a new line. Then the section that was iterated over will become a sub-list of the output list etc. The predicate outputs a list of sub-lists, each sub-list represents a sentence.

```
cTs([],[[]|[]]).
cTs([F,F2|L],[[F|R]|S]):-
    (F\=46;
    F\=10;
    F2\=32),
    cTs([F2|L],[R|S]).

cTs([L],[[L|R]|S]):-
    cTs([],[R|S]).


cTs([F,F2|L],[[]|S]):-
    ((F=46,
    F2=32);
    F=10),
    cTs(L,S).
```

Then dividing these sentences, each sentence will be divided into a list of words. This is done by the next predicate, which takes as an input a list of sub-lists, where each sub-list represents a sentence. The predicate basically iterates over each sentence, splitting it on space. Meaning, it iterates over each sub-list until it finds a space and sets the covered selection so far as a sub-sub-list of the output list (word). In the output list, sentences are the sub-lists and words are the sub-sub-lists. Also the predicate changes the characters from their ASCII codes to their original form using `char_code`, which is a predefined Prolog function that takes the ASCII code of the character as an input and returns its original form.

```
allcTc([],[]).
allcTc([H|T],[R|R2]):-
    cTc(H,R),
    allcTc(T,R2).

cTc([],[[]|[]]).
cTc([F|L],[[N|R]|S]):-
    F\=32,F\=46,
    char_code(T4,F),
    N=T4,
    cTc(L,[R|S]).

cTc([F|L],[[]|S]):-
    (F=32;F=46),
    cTc(L,S).
```

Having the text file as a list containing the sentences as sub-list and the words as sub-sub-list, ordered constraints can be simply done by creating a sentence identifier according to the position of the sentence in the list (text). Also, creating a word identifier according to the position of the word in the sub-list (sentence). These CHR rules take the output list and generate word constraint, where each constraint (`word/3`) contains the word itself and two identifiers; the identifier of the word in the sentence and the identifier of the sentence itself in the text.

```
comp([],_,_)        <=> true.
wordify([],_)       <=> true.
wordify([],_)       <=> true.
wordify([H|T],ID)   <=> comp(H,1,ID),ID2 is ID+1,wordify(T,ID2).
comp([H|T],ID ,ID3) <=> word(H,ID,ID3),ID2 is ID+1 ,comp(T,ID2,ID3).
```

Frequent Words Implementation

In the implementation of frequent words, sentence and word identifiers in the generated constraints were not used, since positions of the words in the text are not crucial. It is only important to know if the words appear or not. A word constraint (`word/3`) is a constraint having the word and the two identifiers, which were not used in the frequent words implementation. A count constraint (`count/2`) is a constraint having the word and its count value in the text. The CHR implementation of frequent words is simple and is based on two main conditions:

- If there is a word constraint, then simplify this word constraint to a new count constraint for this word, with its count value equal to one.
- If there are two count constraints for the same word, then simplify them to a new one, with its count value equal to the sum of both input count constraints values.

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

After all the counts have been calculated, filtering process takes place. In the current implementation of the tool, the threshold for filtering the counts is three. The next rule prints the words which has a count greater than three.

```
count(W,C) <=> C>3 | writeWord(W), writeln(' '), writeln(C).
```

Sequences and Word Gravity Implementation

Possessing the words as constraints, it is easy to detect sequences. For example, if you need to recognize a two-word sequence, one only needs to check if the two words are in the same sentence by checking the sentence identifier and also check if the two words are consecutive by checking the words identifier, and the same goes to the three-word sequence etc. Also, for detecting the non sequential word patterns, one only needs to check if the two words are in the same sentence and that the second word comes after the first one.

The following CHR rule detect two-word sequences and gravity at the same time. The rule takes two constraints representing two words with their identifiers as an input, and if they are a sequential or non sequential word patterns then the rule generates a sequence constraint (`sequence/3`) of the two words with a calculated gravity. If the two words are exactly following each other then when applying the gravity rule presented in the approach in algorithm 1, the calculated gravity will be one, otherwise the gravity will be powers of half according to how far the words are from each other. Also, a limitation is applied which is, the distance between two non sequential words should not exceed five words to generate a non sequential pattern. This rule generates sequences constraints (`sequence/3`) where each one consists of the two words contributing in the sequence and the gravity of the sequences.

```
word(W,I,S), word(W1,I1,S) ==>
           I1>I , R is I1-I-1 , R<5
           | R2 is 0.5 ** R , sequence(W,W1,R2).
```

Also the rule to generate three-word sequences is as follows:

```
word(W,I,S), word(W1,I1,S), word(W2,I2,S) ==>

           I1 is I+1, I2 is I1+1, sequence3(W,W1,W2,1).
```

Now that the sequences have been generated, identical sequences should be summed to calculate a sequence strength. This is done by checking on the two words in the sequences constraints. If the sequences are identical then the two sequences are simplified to new sequence containing the same two words but with their gravity summed to magnify the strength. Also, same for the three-words sequences to the N-words sequences. The CHR rule for summing sequences is the same idea as the rule that sum the important words' counts introduced in 4.4.2.

After summing the sequences, the filtering process starts in order to have only the important sequences. The sequences are filtered according to an experimental threshold, which is taken to be three in this tool. The CHR rule simply states that if there is a sequence which has strength less than three then filter it out.

```
sequence(W,W1,Count) <=> Count < 3 | true.
```

In order for the CHR output to be written in command prompt and consequently to be read in PHP, the CHR output has to be written i.e. printed. Printing out was implemented using a CHR rule and a Prolog predicate. The rule calls the Prolog predicate, giving it as input the words to be written. Also the rule writes the strength of the sequence on a new line.

Overlapping Sequences

Another CHR rule was implemented yet not used in the tool, is the overlapping sequences. Its idea is, merging every two overlapping sequences together to generate a new sequence. For instance, the sequences `Constraint Handling` and `Handling Rules` are overlapping, since the first sequence ends with `Handling` and the second sequence starts with `Handling`. If two sequence fulfil this condition a new sequence is generated combining them both, in the previous case the resulting sequence would be `Constraint Handling Rules`. The overlapping sequences are generated after all sequences are summed and filtered out, to ensure that there are no redundant sequences and only important overlapping sequences are generated.

```
sequence(W,W2,_), sequence(W2,W3,_) ==> sumsequence3(W,W2,W3).
```

4.4.3 PHP Handling Ouptut

After the SWI-Prolog terminates and the output is written on the command window, the written output is stored in a PHP array. Then, this PHP array is divided into four arrays; an array for the two-word sequences mapped with an array that holds the strengths of the two-word sequences, an array for the counts mapped with an array that holds the strengths of the counts array.

4.5 Algorithms and Visualizations(Web Development II)

4.5.1 Charts and Highlighting Implementation (JavaScript)

Having the counts and sequences represented in PHP arrays, they can be converted into JavaScript arrays to use Highcharts. The following code is a JavaScript code that does a simple conversion from a PHP array to a JavaScript array. However, conversions of arrays which hold numbers needs to parsed.

```
var words = <?php echo json_encode($counts); ?>;
var numbers = <?php echo json_encode($countsValues) ; ?>;
for(var i = 0 ; i< numbers.length;i++)
{
  numbers[i]= parseFloat(numbers[i]);
}
```

Two bar charts were implemented in the tool; one visualizes the important words as bars, where the bar length indicates how many times did the important words appear. The second one is as the previous but visualizes two-word sequences. Bar charts were implemented using an example downloaded from Highcharts [19] and using the converted arrays described above. However, some minor changes were made to the example code; changing the size of the chart according to the array size, and changing the on-click event handler to highlight the selected word or sequence. The click event handler calls a JavaScript function called `highlight`. This function highlights the selected word in the original text. Also, there is another function that highlights the sequences.

```
plotOptions:
{
   column: {
  pointPadding: 0.2,
  borderWidth: 0,
  cursor: 'pointer',
  point: {events: {click: function() {
  highlight(this.category);}}}},
  dataLabels:{
  enabled: true}
}

function highlight(text)
{
  var inputText = document.getElementById("inputText")
  var strLen = text.length;
  var re = new RegExp('( '+text+')','gi');
  inputText.innerHTML = inputText.innerHTML.replace(re,'<span
  style="background-color:yellow;">$1<\/span>');
}
```

The charts output is shown in figures 4.4, 4.5 and 4.6 when running the search query `Galaxy`. Where, figure 4.4 shows the bar chart of important words' counts for the query `Galaxy`. Moreover, the word `galaxies` is selected to be highlighted within the search results. Also, it is hovered over to show its exact count. The chart is sorted in descending order; from the higher counts to the lower ones.

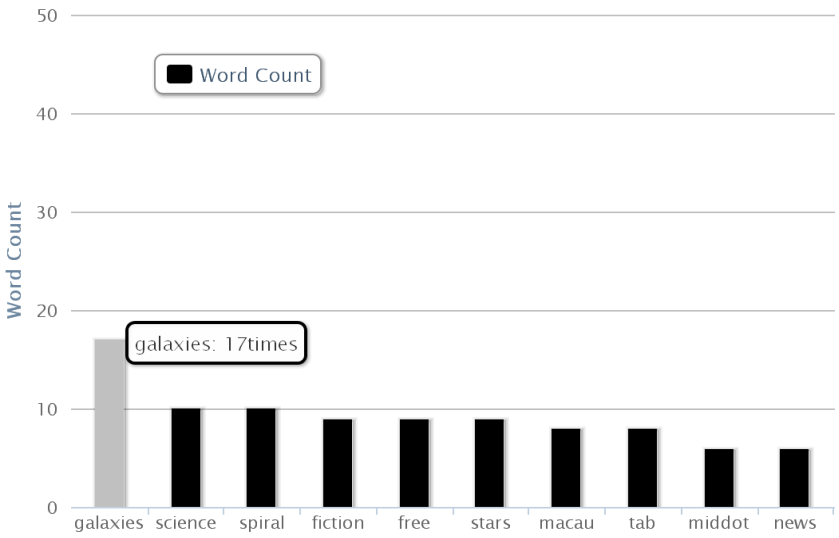

Figure 4.4: Important words' counts chart.

The second figure 4.5 also shows a bar chart of the two-word sequences for the same query. The chart is also sorted as the previously mentioned chart. The word `science fiction` is selected to be highlighted within the search results. Some of the counts might have decimals because of the gravity calculation.

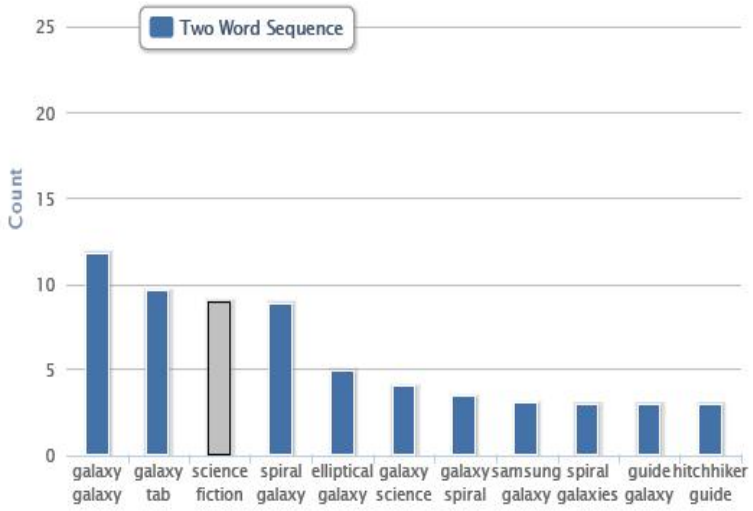

Figure 4.5: Two-word sequences chart.

The last figure 4.6 shows the search results, having the selected words in both of the previous charts highlighted.

galaxy (plural galaxies). (rare) Milky ; apparent band concentrated stars [edit] Derived terms. terms derived galaxy (noun) ...
Galaxy Suites amp; Spa Imerovigli Santorini offers Greek traditional modern luxury suites spectacular views sunset, volcano Oia.
13, 2009 Galaxies large systems stars interstellar matter, typically million trillion stars, masses locally owned operated Austin cafe, Galaxy Cafe combines high quality homemade food convenience amp; moderate pricing "fast casual" dining.
information growth galaxies. includes images, news, glossary.
Science fiction LED watch design Japan. 3 LED colors unusual time.

Figure 4.6: Search results having the selected words highlighted.

4.5.2 Scored Results Implementation

Scored results are implemented according to the algorithm illustrated in the approach in algorithm 2. The following code is an implementation of the presented algorithm, where `$resultsUS` is an array which contains the search results. Moreover, the array `$counts` contains the important words found in the text mapped with `$countCounter`, which contains `$counts` array strengths. The code loops over the search results and checks if it has any important word, then it adds the count of the important word to the result score. The scores of all the results are stored in the array `$rating` which is mapped with `$resultsUS`. Matching the important words to the result is done using a predefined function called `preg_match_all` which returns true if there is a match or false otherwise. Also, the function returns the pattern matches found and store them in an array, which

is `$matches` array in this case. The function takes as an input a pattern, a subject to match this pattern on and a flag. The flag used is `PREG_PATTERN_ORDER` which orders the first element of the array `$matches` to have all the full pattern matches, and the second element to have the matched sub-patterns. Using this array, one can easily get how many full pattern matches are found by getting the size of the first element of the array `$matches`.

```
 $rating = array();
 $ratingCounter=0;
for ( $counter = 0; $counter < sizeof($resultsUC); $counter += 1)
{
  $varC =0;
  for ( $counter2 = 0; $counter2 < sizeof($counts); $counter2 += 1)
  {
    $var =  "~" . substr_replace($counts[$counter2] ,"",-1) . "~i" ;
    if(preg_match_all($var,$resultsUC[$counter],$matches,
    PREG_PATTERN_ORDER))
    {
      $varC= $varC + $countsCounter[$counter2];
    }
  }
  $rating[$ratingCounter]=$varC;
  $ratingCounter+=1;
}
```

The following line of code is added to remove an extra trailing space in each important word for pattern matching, because matching the word “Google” with “Google ” will return false.

```
    $var =  "~" . substr_replace($counts[$counter2] ,"",-1) . "~i" ;
```

After calculating all the results score in the array `$rating`, the array is sorted according to the highest score to have the highest scores result at the top of the array and the least scores at the end. Insertion sort is used for this sorting.

The output of the scored results is shown in figures 4.7, 4.8, when running `Galaxy` as a search query, where figure 4.7 shows some of the top scored results and figure 4.8 shows some of the least scored results. Moreover, the important words that affected the score of the results are shown in larger font.

Figure 4.7: Snapshot showing top scored search results.

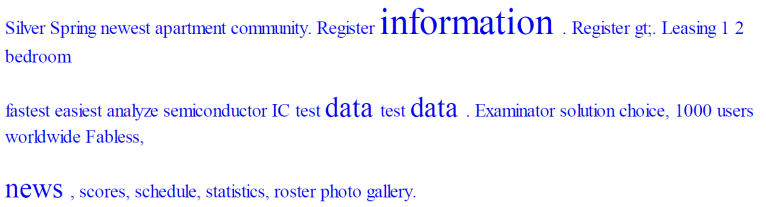

Figure 4.8: Snapshot showing the least scored search results.

4.5.3 Results Clustering Implementation

As presented in the approach in 3.3.5, before applying the clustering algorithm, determining for each important word the set of numbers which represent the indices of the results list in which each important word appeared in, should be done first. For instance, word "help" appeared in results `{63,62,54,7,1}`. However, before determining these sets, the search query is removed from the list of important words. Because most probably the search query would appear in every search result, which may result in only one cluster group and this is undesirable. The following code generates the set of result indices R for each important word W, it iterates over the array `$counts` which holds the important words. The code holds an important word and creates an array for it, then checks, in which search results did it appear in. If an important word appeared in a search result, then the index of the search result is pushed in the array of the important word. The output of this algorithm is an array of arrays, where each sub-array holds the set of numbers indicating the indices of the search results for a certain important word. The resulting two-dimensional array `$Cluster` is mapped with `$counts`, where each important word in `$counts` has its result set as a sub-array in the same index in `$Cluster`, presenting the pair $< W, R >$

```
for ( $counter2 = 0; $counter2 < sizeof($counts); $counter2 += 1)
{
  $Cluster[$counter2]=array();
  for ( $counter = 0; $counter < sizeof($resultsUC); $counter += 1)
  {
```

```
    $var =  "~" . " " . $counts[$counter2] . " " . "~i" ;
    if(preg_match_all($var,$resultsUC[$counter],$matches
       ,PREG_PATTERN_ORDER))
      {
        array_push($Cluster[$counter2], $counter);
      }
  }
}
```

The code is an implementation to algorithm 3 presented in the approach. where `$Cluster` is a list of sub-lists, each sub-list indicate the set of numbers representing the indices of the search results an important word appeared in. However, a small part was added to the implementation which is, if there are no intersections between a sub-list and all the other sub-lists then a new unique group is created for it. Intersection calculation is done using the PHP predefined predicate `array_intersect`, the function simply check if two arrays intersect and returns the intersection count. The output array is `$clusterGroups`, it is an array of numbers mapped with `$Cluster` and `$counts`. The numbers in `$clusterGroups` represents the unique cluster numbers for each important word in `$counts`, this mapping presents the tuple $< W, C, R >$. The implementation code in A.3.

The figures 4.9, 4.10 show the output of the cluster algorithm running `Galaxy` as a search query. The figures show two out of seven clusters for the word `Galaxy`.

large-milky-stars-galaxies-

information, facts, photos, news, videos, galaxies National Geographic.

common type galaxy called quot;spiral galaxy.quot; surprisingly, spiral galaxies spirals, long arms winding bright bulge

interactive java applet model galaxy collisions applet study galaxies collide merge gravitationally ...

Oct 20, 2005 Astronomers classify galaxies major categories. Spiral galaxies flat disks bulges centers beautiful spiral ...

Chance Alignment Galaxies Mimics Cosmic Collision Dark Matter Map Galaxy Cluster Abell 1689 Eighth Anniversary Image Hubble Smash

Figure 4.9: The first cluster for the query “Galaxy”.

tab-samsung-home-

MLS Regular Season, LA Galaxy, 1 3, Real Salt Lake, Home Depot CCL CONCACAF, LA Galaxy, 1 2, Toronto FC, Home Depot Center, RECAP ...

Meet Samsung Galaxy Tab family including Galaxy Tab 10.1, Galaxy Tab 8.9 Galaxy Tab 7.0 Plusall WiFionly connected versions

blazing fast processor America Largest 4G Network, Samsung Galaxy II lets watch surf faster home Internet.

Verizon Galaxy Tab Verizon 13.58ounce, 7inch tablet powered Android 2.2 OS slips easily pocket bag. Verizon Samsung

Figure 4.10: The second cluster for the query "Galaxy".

4.6 User Guide

This section describes how to install this tool. The implemented tool includes an HTML file, a Prolog file and four PHP files. The following are some guidelines to install the tool:

- First installing PHP, in this tool Wampserver is used to install PHP, more information in [20]. After installation, put the PHP files in this path `C:\wamp\apps\phpmyadmin3.4.5`. The following are some notes regarding the path:
  - The path may differ according to the Wampserver installation path and the PHP version.
  - The path may completely change if another tool other than Wampserver is used.
- Download and install SWI-prolog from [15]. The installation has to be in a path which includes no spaces, because the command prompt cannot access a path with spaces. For example, `C:\Programe Files\` is not valid because there is a space between `program` and `Files`.
- The Prolog file has to also be put in a path with the same specifications as SWI-Prolog.
- Create a text file in any place so that PHP can write in it and Prolog can read it.
- Modify the paths in the HTML file, Prolog file and the PHP files according to the installation paths of PHP, SWI-Prolog and the text file.

4.7 Tool Evaluation

In this section, we conduct some general feature comparisons to other online tools trying to evaluate this tool, and introduce its limitations.

4.7.1 Basic Comparison

After searching for online tools for text mining, plenty of online tools were found in this context; summarizing tools, clustering tools and also analyzing tools, examples for these tools are [21], [22] and [23] respectively. Each one of the previously mentioned tools has a lot of features and capabilities. However, a tool which combines all the basic features of summarizing, analyzing and clustering was not found. Moreover, the most common way found for representing important words and sequences was the tag clouds, also no tool visualizing them as charts was found. Also, this is the first tool to use CHR in text mining and classification of Google search results.

4.7.2 General Limitations

The tool is relatively computationally slow, it takes around `22.03` seconds to output the result for nearly `100` search results. Counts of important words, sequences and gravity mainly consumes majority of the time taken.

4.7.3 SWI-Prolog Limitation

According to the SWI-Prolog manual regarding the local stack size [14]. The maximum local stack size that could be obtained on 32-bit windows using 32-bit SWI-Prolog is $128MB$ (mega bytes), which is very small in size for modern applications while having modern hardware. This affects the amount of input text to the tool, since the tool transforms the text file into a list which in tern needs a large stack according to how big is the text file. However, on a 64-bit windows with a 64-bit SWI-Prolog, the maximum local stack that can be obtained is 2^{32} times higher, utilizing and exceeding nowadays hardware capabilities. Increasing the stack size has a very slight effect on decreasing the execution time of small amount of input data. However, this slight effect increases as the input text increases.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The main goal of the thesis was to implement a web text mining tool for Google search results, that uses CHR as a base for its implementation. It is the first tool to use CHR for text mining purposes, and combines it with algorithms and visualizations to summarize Google search results. The algorithms and visualizations which are the web development part of this tool, were implemented using PHP. Three main issues had to be dealt with in order to implement this tool. First, searching for the best way to extract Google search results. Second, performing linguistic analysis by detecting sequences and calculating important words' counts. Last but not least, improving the tool by adding some algorithms and visualization. Handling the first issue, Google search results were extracted using Google Custom Search Engine, which is a customized search engine that possesses an online control panel. The control panel allows its administrator to generate implementation code according to the desired features. It introduces collaboration, viewing statistics of the engine, auto-completeness in search bar and other features. Afterwards, text transformation was performed on the returned search results, converting the text within the search results into ordered constraints in CHR. Then, CHR was used to analyze Google search results by calculating important words' counts besides generating two-word sequences and gravity, where gravity is the strength between non sequential patterns. Implementing sequences in CHR is effortless, any N-word sequence can be generated in a single rule, and to leave the important sequences only, another two rules were added. However, as N increases, computational time increases. Finally, some algorithms to improve the tool were added, and a combination of some basic but important features was created. The combination includes interactive charts which were used to represent important words' counts and sequences as bar charts, ranking the search results according to the important words it has. Eventually, the search results were classified to realize a comprehensive and clustered summary of results.

5.2 Future Work

There are many future expansions that can be done for this tool to enhance its performance and improve its capabilities. Some of the these enhancements are

- *Speed enhancement:* Speeding CHR execution through prallelizing the CHR rules. Strong parallelism can be applied, running overlapping parts in parallel. However,

confluence and monotonicity features has to be proven for these rules. Strong parallelism, confluence and monotonicity properties are discussed in part two of the CHR book [5]. Parallelizing CHR rules would enhance the speed of execution significantly.

- *More linguistic analysis and better algorithms and visualizations:* If the parallelism proved significant speed increase, three- and four-word sequences and gravity can be generated and added to the tool. Moreover, these three- and four-word sequences can be also used to enhance charts and scored results; viewing two-, three- and four-word sequences as bar charts in addition to important words' counts, instead of only viewing important words' counts and two-word sequences. Moreover, scored results also could be enhanced by modifying the algorithm to include two, three, four-word sequences in the calculation, to have a better scoring algorithm.

- *Language independence:* The tool can be advanced to be a language independent tool. The stop words library is the only restriction that makes this tool language dependant. If other libraries for different languages are added to the tool, a language independent tool can be achieved.

- *Iterative clustering and tree maps:* Improve the clustering algorithm to have iterative clustering, where each group can be itself clustered into more groups, resulting in more specified and detailed clustered data. Moreover, the clusters visualization can be improved by using tree maps, an example of such tree maps is shown in [24].

Appendix A

Implementation Code

A.1 Preparations for Stop Words and Case Folding Implementation

```
if (isset($_POST['allResults']))
{
    // Getting the results from google search page.
    $name0 = $_POST['allResults'];
}
if (isset($_POST['searchQuery']))
{
    // Getting the search query from google search page.
    $name1 = $_POST['searchQuery'];
}

$name1 =explode(" ",$name1);
// The library is trimmed because its too big.
$commonWords = array('the','can','an','a',....);
$splitted2= explode("******",$name0);
$name0=preg_replace('/\b('.implode('|',$commonWords).')\b/i','',$name0);
$search  = array('   ','  ');
$name0 = str_replace($search,' ', $name0);
$splitted= explode("******",$name0);
$results=array();
$urls=array();
$resultsCounter=0;
$urlsCounter=0;
$resultsUC = array();
for ( $counter = 0; $counter &lt; sizeof($splitted)-1 ; $counter += 2)
{
    $resultsUC[$resultsCounter] =($splitted[$counter]);
    $results[$resultsCounter] = strtolower($splitted[$counter]);
    $urls[$urlsCounter]=$splitted2[$counter+1];
    $resultsCounter+=1;
    $urlsCounter+=1;
}
```

A.2 Preparations for CHR

```
test:-
  readFile('C:/Dokumente und Einstellungen/Alexander/Desktop/text.txt').

readFile(X):-
  open(X, read, Stream),
  read_stream_to_codes(Stream,Y,[]),
  removePunc(Y,Y2).

removePunc([],[]).
removePunc([F|T],[F|T2]):-
  F\=44,F\=63,F\=59,F\=34,F\=58,F\=40,F\=41,F\=93,F\=91,F\=124,
  removePunc(T,T2).

removePunc([F|T],R):-
  (F=44;F=63;F=59;F=34;F=58;F=40;F=41;F=93;F=91;F=124),
  removePunc(T,R).
```

A.3 Clustering Implementation

```
$groupsCounter=200;
for ( $counter = 0; $counter < sizeof($Cluster);$counter+=1)
{
  $var = $Cluster[$counter];
  $flag =0;
  $maxSoFar = -1;
  $indexSoFar = -1;
  for($counter2 = 0; $counter2 <sizeof($Cluster)
      && $counter2 != $counter; $counter2+=1)
  {
    $var2 = $Cluster[$counter2];
    $result = array_intersect($var, $var2);
    $size =sizeof($result);
    if($size>0)
    {
      $flag=1;
      if($maxSoFar < $size)
      {
        $indexSoFar=$counter2;
        $maxSoFar= $size;
      }
           }
  }
    if($flag==0)
    {
      $groupsCounter++;
      $clusterGroups[$counter]=$groupsCounter;
      $clusterArraySize[$counter]= count($Cluster[$counter]);
    }
      else
```

```
      {
        if($clusterGroups[$counter] > $clusterGroups[$indexSoFar])
        {
          $clusterGroups[$counter]=$clusterGroups[$indexSoFar];
          $clusterArraySize[$counter]=$clusterArraySize[$indexSoFar];
        }
          else
          {
            $clusterGroups[$indexSoFar]=$clusterGroups[$counter];
            $clusterArraySize[$indexSoFar]=$clusterArraySize[$counter];
          }
      }
}
```

Bibliography

[1] "Highcharts example." http://www.highcharts.com/demo/combo, June 2012.

[2] "Lucid charts an online tool for drawing flow charts." https://www.lucidchart.com/, June 2012.

[3] "Alexa the web information company." http://www.alexa.com/siteinfo/google.com, March 2012.

[4] "Extensive list of the major internet search engines." http://www.listofsearchengines.info, March 2012.

[5] T. Frühwirth, *Constraint Handling Rules*. Cambridge University Press, Aug. 2009.

[6] A. hwee Tan, "Text mining: The state of the art and the challenges," in *In Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, pp. 65–70, 1999.

[7] J. Larocca Neto, A. D. Santos, C. A. A. Kaestner, and A. A. Freitas, "Document clustering and text summarization," in *Proceedings of the 4th International Conference Practical Applications of Knowledge Discovery and Data Mining (PADD-2000)* (N. Mackin, ed.), (London), pp. 41–55, The Practical Application Company, January 2000.

[8] "Php manual." http://www.php.net/manual/en/intro-whatis.php, June 2012.

[9] D. Flanagan, *JavaScript: The Definitive Guide*. O'Reilly Media, Inc., 2006.

[10] "Apidef." http://msdn.microsoft.com/en-us/library/aa141380(v=office.10).aspx, June 2012.

[11] D. Crockford, "The application/json media type for javascript object notation (json)," tech. rep., July 2006.

[12] M. Nottingham and R. Sayre, "The atom syndication format." Internet RFC 4287, December 2005.

[13] "Highcharts." http://www.highcharts.com/products/highcharts, June 2012.

[14] J. Wielemaker, "Swi-prolog 6.0 reference manual," March 2012.

[15] "Swi-prolog official website." http://www.swi-prolog.org/index.txt, June 2012.

[16] "Google terms and conditions." http://www.google.com/intl/en/policies/terms/, June 2012.

[17] "Google cse." http://www.google.com/cse/, June 2012.

[18] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, (Washington, DC, USA), pp. 3–14, IEEE Computer Society, 1995.

[19] "Highcharts downloaded example." http://www.highcharts.com/download, June 2012.

[20] "Wampserver, a windows web development environment." http://www.wampserver.com/en/s, March 2012.

[21] "Summarization tool." http://www.sensebot.net/, June 2012.

[22] "Clustering tool." http://search.carrot2.org/stable/search, June 2012.

[23] "Analysis tool." http://textalyser.net/index.php?lang=en#analysis, June 2012.

[24] "News map." http://newsmap.jp/, June 2012.