University of Alberta

SEARCH TERM SELECTION AND DOCUMENT CLUSTERING FOR QUERY SUGGESTION

by

Xiaomin Zhang

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Spring 2011
Edmonton, Alberta

Examining Committee

Robert Holte, Computing Science

Sandra Zilles, Computer Science, University of Regina

Randy Goebel, Computing Science

Dangzhi Zhao, Library and Information Studies (External)

Abstract

In order to improve a user's query and help the user quickly satisfy his/her information need, most search engines provide query suggestions that are meant to be relevant alternatives to the user's query. This thesis builds on the query suggestion system and evaluation methodology described in Shen Jiang's Masters thesis (2008). Jiang's system constructs query suggestions by searching for lexical aliases of web documents and then applying query search to the lexical aliases. A lexical alias for a web document is a list of terms that return the web document in a top-ranked position. Query search is a search process that finds useful combinations of search terms. The main focus of this thesis is to supply alternatives for the components of Jiang's system. We suggest three term-scoring mechanisms and generalize Jiang's lexical alias search to be a general search for terms that are useful for constructing good query suggestions. We also replace Jiang's top-down query search by a bottom-up beam search method. We experimentally show that our query suggestion method improves Jiang's system by 30% for short queries and 90% for long queries using Jiang's evaluation method. In addition, we add new evidence supporting Jiang's conclusion that terms in the user's initial query terms are important to include in the query suggestions.

In addition, we explore the usefulness of document clustering in creating query suggestions. Our experimental results are the opposite of what we expected: query suggestion based on clustering does not perform nearly as well, in terms of the "coverage" scores we are using for evaluation, as our best method that is not based on document clustering.

Acknowledgements

I would like to thank my supervisors Dr. Robert C. Holte and Dr. Sandra Zilles first. For the project, they are my advisors and also teammates. They had the initial idea and guided me to quickly get on track at the beginning. As the study progressed, when there were many possible directions I could go, they first prevented me from going in too many directions and encouraged me to explore other possibilities later. Also, they provided me with lots of great ideas and insights. Though some of them did not appear in the thesis, I still gained a lot by studying and testing them. During the writing of my thesis, they've been so patient in their revision.

Then I'd like to thank Shen Jiang, Ying Xu, Hsiu-Chin Lin, Zhaochen Guo, Guohua Liu, Michel Masset. They've all helped me on this thesis work at different stages. Also Dr. Russell Greiner and Dr. Nathan Sturtevant have influenced me in some parts. I also really appreciated AICML in Department of Computing Science at University of Alberta gave me the opportunity to do this research. My roommates, though we are all different majors, are families to me in Edmonton.

At last, I want to thank my parents and my old friends all over the world. They support me all these years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Web search is one of the most popular and useful services on the Internet. Given the large number of web documents on the Internet, users find useful information via the search engine. For example, if a user wants to know something about “thermodynamics”, s/he may form the *query* “thermodynamics” and enter it in the search box of a search engine. The search engine returns millions or billions of web documents related to the query “thermodynamics” and the most *relevant* ones are ranked in the top. Each search result contains a title, a *snippet* and a URL address. A snippet consists of one or two sentences or phrases extracted by the search engine from the web document. By looking at the title and the snippet, a user decides whether to navigate to the corresponding web document or not.

Users may face some problems when using a search engine. Sometimes the search engine returns almost nothing for a query, sometimes the search engine returns a large number of irrelevant web documents. For both cases, the reason is usually not that there is no useful information for the user on the Internet, it is often because of an inappropriate query. The query entered in the search box is almost the only knowledge the search engine has to estimate the user’s requirement.

There are several possible reasons for an inappropriate query. On the lexical level, there are spelling errors [9], splitting a word which should not be split, merging words which should not be merged, acronym problems, etc. [14]. Considering spelling errors for example, when the query “machine learning” is typed as “machin learning”, the quality of the relevant results might be reduced greatly. For the acronym problems, the query “socs” might retrieve only a few web documents related to the International Symposium on Combinatorial Search (SoCS) mingled with documents not related to this subject. Besides lexical mistakes, users tend to form short queries consisting of only one or two words [3, 5]. Short queries are more likely to be ambiguous [7]. For example, if a user wants information about the jaguar car and enters the query “jaguar”, the returned results may contain web documents about the jaguar car, the panthera onca, an aircraft named jaguar, etc. Furthermore, the web documents on the Internet are created and maintained by different people. This leads to a *vocabulary mismatch* problem, i.e. the words in the user’s query may be different from the words in the relevant web documents, though these words refer to the same thing. For example, the

query “cat” may not be able to retrieve the web documents containing “feline”. On top of all these objective reasons, a user may simply not be able to form the correct words for his/her specific intent sometimes [54].

Therefore, based on the user’s original query, most search engines now supply query suggestions that may better represent the user’s search intent and help the user find useful web documents faster. For example, after a user types “volcanos in italy” on Google[1], the query suggestions are supplied at the bottom of the first page as shown in Figure 1.1. The procedure of creating these queries is called *query suggestion* [3, 31], *query refinement* [14, 16], *query expansion*, *query reformulation* or *query substitution*. Different research works may use different names or focus on different aspects, but they all share the purpose of generating new queries to improve the user’s original query and enhance the search usability.

Figure 1.1: The query suggestions for the query “volcanos in italy” on Google.com (Dec. 22nd, 2010).

There are many techniques to supply query suggestions for a query, such as methods based on global thesauri [5, 32, 56], local query suggestion methods [32, 56], methods utilizing search logs [3, 31, 54], etc. In addition to creating query suggestions that are relevant to the user’s query, most query suggestion methods also aim to return understandable and recognizable query suggestions for humans. However, a good query for humnas is not guaranteed to be a good query for the search engine. This thesis aims to provide good query suggestions for the search engine.

Since queries are often ambiguous, search results about different subjects often mix together. There is a considerable amount of research done on organizing search results, i.e. categorizing web documents for a query. These methods are usually referred to by the term *web document clustering* [12, 13, 15, 57, 58]. Though generating a query suggestion for each web document cluster seems promising, there appears to be no literature explicitly utilizing web document clustering for query suggestion. This thesis makes this attempt.

The following sections sketch the problem this thesis aims to solve, the general approach, the contributions and an outline of this thesis.

[1]The Google search engine is at `http://www.google.com/`.

1.1 Task Definition and Approach

Before introducing our work, the query suggestion method proposed by Jiang et al. [19] has to be described first, since we follow its basic assumptions, purpose and evaluation methodology.

The query suggestion method of Jiang et al. [19] is based on one crucial assumption. It assumes the user's query returns relevant web documents, but that the query is not good enough to return the relevant web documents in the top positions for the user to see. With this assumption, the general task of the system developed by Jiang et al. [19] is to create query suggestions that return relevant web documents which are likely to be missed by the user back to the top positions for the user to see.

Given the task, the relevant web documents that are likely to be missed by users have to be located first. Jiang et al. [18] conducted an experiment to explore the positions of relevant web documents in the search results of a query. Their work shows that there is a high probability that a relevant web document appears in the top 120 search results. However, most users only view the top 20 results (the first two pages of the search results). In other words, the work of Jiang et al. [18] shows that a web document between top 21-120 in the search results is very likely to be useful to users but may be missed because it is ranked too low (below the top 20).

With the assumption and the work on relevant web documents [18], Jiang et al. [19] suggested a purpose for their query suggestion method: creating query suggestions to return the web documents between 21-120 back in the top 20 results. These web documents are called *reference document*s for the user's query. In addition, they say a query *cover*s a web document if the query returns the web document in the top 20 results. Following the previous example (Figure 1.1), we show the reference document coverage of a query suggestion provided by Google.com for the query "volcanoes in italy" in Table 1.1. The left column shows the reference documents for the query volcanos in italy and the right column shows documents returned by the query suggestion major volcanoes in italy. The covered reference documents are emphasized in Table 1.1 (reference documents 59 and 88 are covered)."

There are some things to be noted about the purpose of both Jiang et al.'s query suggestion system and the query suggestion methods in this thesis. Covering all and only the reference documents is not the goal we pursue. Otherwise, a much easier approach would be to cluster the reference documents in groups, giving each group an understandable label, and returning the labels as links to the grouped documents directly to the user. The disadvantage of this approach would be that *only* the reference documents and no other documents would be presented to the user. What we are tackling in this thesis is how to provide query suggestions that allow the user to find relevant documents when they are not in the top 20 returned by the initial query. We need a way to estimate how many relevant documents a given query suggestion returns in its top 20. We follow Jiang et al. in using the number of reference documents covered to estimate this.

In order to evaluate their query suggestion method, Jiang et al. [19] introduced the MCC score

Original query: volcanos in italy	**Query suggestion: major volcanoes in italy**
1. List of volcanoes in Italy - Wikipedia, the free encyclopedia	1. CVO Website - Major Volcanoes in Italy - Map
...	2. CVO Menu - Italy Volcanoes and Volcanics
20. Stromboli volcano (Italy) in eruption — Flickr - Photo Sharing!	3. Volcanism of Italy - Wikipedia, the free encyclopedia
21. CVO Menu - INDEX to CVO Online Volcanoes	4. Mount Etna - Wikipedia, the free encyclopedia
22. Arenal Volcano Costa Rica overview	5. List of volcanoes in Italy - Wikipedia, the free encyclopedia
23. Answers.com - What Volcanos are in Italy	**6. major volcanoes of italy map and information page**
24. Undersea Volcano Threat For Italy/Undersea Volcanoes Of Asphalt In ...	7. Answers.com - Name 3 major volcanoes in italy
25. Google Maps / Google Earth - Volcanos & Italy & Vulcano	8. Answers.com - What is the most important Volcano in Italy
26. World Map of Volcanoes, Volcanoes Of The World	**9. Active Volcanoes: Stromboli, Italy**
27. World Volcanoes Map — Volcano Lookup — Kamchatka Volcanos — Japan ...	10. Volcanoes of Italy - Vesuvius, Campi Flegrei, Etna, Stromboli ...
...	11. Major Volcanoes - World Map, Map of the World
57. Most active volcanos in the world? - Yahoo! Answers	12. Volcanoes In Italy
58. Mt Etna Volcano, Italy - John Seach	13. World's Most Active Volcanoes - John Seach
59. major volcanoes of italy map and information page	14. Global Volcanism Program — Volcanic Activity Reports — Smithsonian ...
60. Volcanoes - Italiansrus.com	15. Italian Volcanoes — Italy
61. Vesuvius, the world's most closely watched volcano	16. World's most active volcanoes - Stromboli (Italy) - CSMonitor.com
...	17. Monte Vulture volcano, Italy
85. WebCam Central : WebCams by Category : Volcanos	18. Active Volcanoes in Europe
86. FIRE BELT AWARD	19. Exploring the Volcanoes of Italy — Expatify
87. Volcano Photos, Volcano Wallpapers, Pictures, Images – National ..	20. Names of Volcanoes in Italy — Directhit.com
88. Active Volcanoes: Stromboli, Italy	
89. top help: what is the top 5 most deadly volcanoes - Help.com	
90. Mt Etna, Sicily's Dominant Volcano, Italy - Video	
91. Volcanoes as emission sources of atmospheric mercury in the ...	
92. Volcanoes - definition of Volcanoes by the Free Online Dictionary ...	
...	
115. Top 7 Posh Hotels at Volcanic Sites	
116. Volcanos by peter francis (volcanos in antartica. how the heck are ...	
117. The Sirente crater, Italy: Impact versus mud volcano origins	
118. Amazon.com: DK Readers: Volcanoes and Other Natural Disasters ...	
119. EARTH CHANGES TV - VOLCANOS: Mt. Etna, Kilauea, Popocateptl ...	
120. Vacation and Travel Talk: Active Volcanos	

Table 1.1: The reference document coverage of the query suggestion provided by Google.com.

(Cumulative Coverage) and the MEC score (Expected Coverage). Suppose K is the number of query suggestions that are supplied. Then the MCC score equals the number of reference documents covered by these K query suggestions. The MEC score is the expected coverage of reference documents by one query suggestion. For example, suppose there are three query suggestions which cover reference documents $\{22, 46, 50\}$, $\{22, 50, 77, 98\}$ and $\{75, 77\}$ respectively. Then the set of the covered reference documents is $\{22, 46, 50, 75, 77, 98\}$, the MCC score is 6 and the MEC score is $\frac{|\{22,46,50\}|+|\{22,50,77,98\}|+|\{75,77\}|}{3} = 3$.

There is one thing to be noted about the evaluation method. We only used MCC and MEC to measure query suggestions with the purpose of evaluating them from the search engine side. However, there are more factors to be considered for query suggestions in practical use, such as the understandability of the query suggestion, and the distinction between query suggestions. Therefore, there are still steps to provide practical query suggestions after creating the ones by our method.

Based on the Query Suggestion by Query Search (QSQS) system of Jiang et al. [19], the first task of this thesis is to improve QSQS by retaining its control structure but replacing the methods in some phases of QSQS with our methods. There are four phases in QSQS, we mainly generalize the lexical alias search phase to a search term selection phase and suggest two alternatives for the query suggestion candidate search phase. An Improved Query Suggestion by Query Search (IQSQS) system is proposed.

Document clustering methods group documents into different clusters based on their similarities. Finding a query suggestion for each cluster seems promising. Therefore, another task of this thesis is to investigate whether document clustering methods help to find query suggestions with higher MCC and MEC scores. For this task, we implemented a Query Suggestion by Document Clustering (QSDC) system. The QSDC system modifies the control structure of QSQS, and inserts a document clustering module in the system.

1.2 Contributions of This Thesis

We proposed the IQSQS system to create query suggestions. For short queries (containing at most 2 terms), the best configuration of IQSQS (we name it IQSQS*) improves the MCC score by 29.2% and MEC by 32.6% compared to the QSQS system. For long queries (containing at least 3 terms), IQSQS* improves the MCC score by 82.1% and MEC by 99.6% compared to QSQS. In addition, we add evidence supporting the conclusion drawn by Jiang et al. [19] that terms in the user's query are important to retain in query suggestions in order to have high MCC and MEC scores.

In the lexical alias search phase of QSQS, a lexical alias for a reference document is selected as the basis of query search. A lexical alias consists of 5 to 10 terms that collectively cover a given reference document. During query search, some terms from the lexical alias are picked out and combined to form a query suggestion candidate. Since the main purpose of the lexical alias is not to cover the current reference document but to supply terms that can be used for forming query

suggestions, we suggest a search term selection phase to replace the lexical alias search phase. In particular, we suggest three scoring mechanisms, LA, OC and EOC, to evaluate terms. LA is evolved from lexical aliases to evaluate whether a term covers the current reference document, i.e. LA is a local coverage. OC and EOC measure how many reference documents, not only the current one, are covered by a term, i.e. the potential of global coverage. With these different scoring mechanisms, we proposed two search term selection methods: LAW and CSW. The experiment shows that, for short queries, CSW can improve the MCC score by 56.9% and MEC by 53.8% compared to LAW.

We also modified the query search phase of QSQS proposed by Jiang et al. [19]. Given a list of terms, query search forms queries by combining terms in different ways. We replace the top-down search method of Jiang et al. [19] by a bottom-up search method and enhance it by a beam search method. The experiment shows that, given the same set of terms, our query search methods perform equally well with Jiang et al.'s query search method.

In order to decrease the running time of evaluating terms and still select useful terms, we apply a pre-selection in the search term selection phase. We compared pre-selection by snippets and by frequency and found that these two methods are equally good. Considering other advantages snippets supply, we adopt snippets as our pre-selection method.

We also noticed the big improvement by always including the original query Q_0 in the query suggestions; the experiment in Chapter 3 demonstrates this. For short queries, adding Q_0 improved MCC by 92.3% and MEC by 94.5%. For long queries, adding Q_0 increased MCC by 32.4% and MEC by 36.8%.

QSQS and IQSQS process 100 reference documents one by one to create the final 10 query suggestions. A Greedy Query Suggestion method by Query Search (GQSQS) is developed by running 10 rounds to greedily find 10 best query suggestions. It runs faster than QSQS and IQSQS and its performance is generally between QSQS and IQSQS. For short queries, QSQS, GQSQS, and IQSQS roughly cover 50, 60, and 70 reference documents respectively. For long queries, QSQS, GQSQS, and IQSQS cover about 40, 70, and 80 reference documents respectively.

We evaluate document clustering methods for query suggestion by developing the QSDC system. The running time and the MCC/MEC score for QSDC are similarly to these of GQSQS. We measure the performances of nine different document clustering algorithms in QSDC, none of them outperforms the others. In order to evaluate whether document clustering helps or not, we carry out another experiment in which documents are randomly assigned to one of ten clusters with a uniform probability. By comparing the result, we conclude that the document clustering methods we used in this thesis do not improve query suggestion in terms of MCC and MEC.

1.3 Remark on Runtime Efficiency

We present our query suggestion methods in this thesis and evaluate them using MCC and MEC score. We do not consider computational time as an important issue, though we apply pre-selection

in the search term selection phase, pruning in the query search phase, and we make our program code efficient to decrease the running time. Since most of our computational time is spent on sending requests to the Google API, we believe that with large scale computing ability and well indexed web documents, the search engine could compute query suggestions with our methods quickly. In addition, the search engine may not need to pre-select terms before evaluating terms, because potentially all terms could be evaluated in a short time. Also, the query search phase could try more query suggestions from the search engine side. Therefore, the search engine could supply even better query suggestions in much shorter time than was required to obtain the results in this thesis.

1.4 Outline

This thesis proceeds as follows: Chapter 2 introduces the essential background knowledge for this work, such as content-based addresses, query search and Jiang et al.'s query suggestion method [19]. Then the Google API used for all experiments in this thesis is introduced. Instability of the Google API and its influence on the experimental analysis are explored.

Chapter 3 elaborates on the IQSQS system and then introduces the GQSQS system. An overview of the IQSQS system is presented and compared with the work of Jiang et al. [19]. Different phases of the IQSQS system are described in detail afterwards. The experiments demonstrate the improvements on the work of Jiang et al. [19].

The QSDC system is introduced in Chapter 4. Similar to Chapter 3, Chapter 4 also proceeds with a system overview and details of each phase in the QSDC system.

Related work about query suggestion methods, web document clustering methods and evaluation methods of document cluster labeling are all described in Chapter 5.

Chapter 6 summarizes the whole thesis and lists possible future directions for our work.

Chapter 2

Essential Background

As introduced in Chapter 1, our query suggestion system has the same purpose as that developed by Jiang et al. [19]: given an original query Q_0, supply K ($K = 10$) query suggestions to return most of the reference documents for Q_0 in the top 20 positions. The idea of finding query suggestions to return a specific set of web documents originates from the idea of *content-based addresses*. In this chapter, we will first describe the concept of content-based addresses, then several methods to find content-based addresses for web documents are introduced. The idea of *query search* that was proposed to search for a content-based address is illustrated after that. Based on content-based addresses and query search, the query suggestion method of Jiang et al. [19] is presented. The last section in this chapter introduces the Google API which we use to retrieve search results for queries in our experiments. The instability of the Google API is analyzed and its influence on our query suggestion systems is also investigated in the last section.

2.1 Content-based Addresses and Related Ideas

Martin and Holte [33] proposed the idea of content-based addresses. Later, a similar concept, called the *lexical signature*, was suggested by Phelps and Wilensky [39]. After that, Jiang et al. [19] applied a modified idea in their query suggestion system and named it the *lexical alias*. We introduce all of them in chronological order.

> **Content-based address (summary query)**: "a content-based alternative to a URL would be a list of key terms that could be used as a query to retrieve the target web page from a large search engine. We will call this query a *content-based address* or more specifically a *summary query*" [33].

Martin and Holte [33] proposed the idea of content-based addresses with two purposes. First, retrieving web documents whose URL addresses change, and second, retrieving web documents whose URL addresses change and whose contents are modified. Furthermore, they noticed that a content-based address also has some potential to return related web documents.

In order to find a content-based address for a web document, an initial query by concatenating the document's most frequent words and the title words is generated first. The initial query is often long and precise and can be used to retrieve a moved web document at the first position in the search results. Queries obtained from shortening and simplifying this initial query are evaluated as to whether they return the web document in the top 10 results or not. All the shortened queries that return the web document in the top 10 results are also collected as content-based addresses. These shortened queries can be used to retrieve a web document that is both moved and modified.

> **Lexical signature**: "it is to include in the hyperlink, along with the URL, some part of the document content. We call this content a lexical signature, as it is meant to identify the given page by its content" [39].

Phelps and Wilensky [39] suggested to include the lexical signature in the address of a web document with the URL to supply a robust hyperlink. With a robust-hyperlink-dereferencing, the lexical signature is ignored when the URL works. If the URL address fails, the lexical signature is issued to a search engine and the web document that matches the signature most closely is supplied to the user.

Phelps and Wilensky thought that rare terms were ideal to form a lexical signature. Therefore, they first utilized the data from search engines to determine the rare terms for a web document, then the term frequency (TF) of each rare term in a web document was calculated. Since most rare terms only occur once in a web document, they favored the rare terms by multiplying their TF scores with their inverse document frequency (IDF) scores (IDF was capped at 5). At the end, they selected the few best rare terms as the lexical signature for the web document.

> **Lexical alias**: "a lexical alias for a web document is a query for which the web document is ranked among the first K_l documents in the corresponding result list, where K_l is a fixed threshold" [17].

Jiang et al. [19] borrow the ideas of content-based addresses and lexical signatures, and the way they build up a lexical alias is similar to Martin and Holte's method [33]. Jiang et al. [19] concatenate the title words and the most frequent words to form a query. Once the query returns the web document in the top 20 results, the query is regarded as the lexical alias for the web document.

2.2 Query Search

Query search was proposed for finding content-based addresses for web documents [33]. Query search is a search process that is used to calculate the desired states (the desired queries) from the start states (a set of important terms). Martin and Holte [33] first proposed the idea of *query search* as quoted below, then Jiang et al. [19] applied query search in their query suggestion system.

> "QuerySearch is a system designed to search for a query that results in one or more particular documents being retrieved. There is a toolbox of possible search heuristics that can be applied. Basically, an initial query is simplified or extended in order to find a query that does a better job of finding the target documents" [33].

As we introduced in Section 2.1, Martin and Holte [33] first find the initial query and then compute the shortened queries. Correspondingly, there are two kinds of query search in their work: search for the initial query and search for the short and simplified queries. Query search for the initial query is simple (Algorithm 2.1 is the pseudocode): the first initial query candidate consists of the f most frequent terms[1] plus the title terms (line 5). If this candidate returns the web document as the first hit of the search results, this candidate is the initial query and this query search finishes (lines 6-8). Otherwise, f is increased by 1 and the same process is repeated (lines 4-10). This search ends when the initial query is found.

Algorithm 2.1 Query search for the initial query by Martin and Holte [33].

Input: a web document d.
Output: the initial query $initialQuery$ that returns d as the first hit of results.

1: initialize $initialQuery$ to be empty.
2: rank terms in d by frequency from highest to lowest, store sorted terms in a list $sortedTerms$.
3: initialize f to be a fixed number.
4: **for** $i = f$ **to** $sortedTerms.size()$ **do**
5: the initial query candidate $initialQueryCandidate$ = top i terms from $sortedTerms$ + title terms.
6: **if** d is returned as the first hit when $initialQueryCandidate$ is issued to a search engine **then**
7: $initialQuery \leftarrow initialQueryCandidate$
8: break loop.
9: **end if**
10: **end for**
11: **return** $initialQuery$

Query search for the short and simplified queries is comparatively complicated (Algorithm 2.2 shows the pseudocode). At the beginning, a query queue is created and is initialized to contain only the initial query (line 1). Then a loop occurs, in which one query q is taken out of the query queue (the head of the queue is taken out) and processed in an inner loop (line 3). The inner loop goes through every term in q (lines 5-11). In the inner loop, a term in q is simplified[2] and the corresponding simplified query q' is tested as to whether q' returns the web document in the top 10 results or not. If q' does, q' is inserted in the query queue and the inner loop moves on to the next term of q (lines 6-10). If all the terms in q have been simplified and no simplified query for q returns the web document in the top 10 results, q is regarded as unable to be simplified and is inserted in the result set (lines 12-14). In this process, testing a query involves issuing it to a search engine and

[1] A term in [33] could be a phrase.

[2] Martin and Holte [33] considered various simplifying possibilities. For example, suppose the term is "$word_1 + word_2 + word_3$" where the quotation marks indicate all words have to appear, the term might be simplified by removing the quotation marks, removing one word from the term, etc.

checking the returned results (line 7). After the outer loop finishes, a result set is returned and all queries in this set are the desired queries (line 16).

Algorithm 2.2 Query search for simplified queries by Martin and Holte [33].

Input: a web document d, the initial query $initialQuery$ for d.
Output: the simplified query set $simplifiedQuerySet$.

```
1:  insert initialQuery to an empty query queue queue.
2:  while queue ≠ NULL do
3:     query q ← queue.pop()
4:     flag canBeSimplified ← false
5:     for every term t in q do
6:        simplify t to form a simplified query q'
7:        if d is returned in the top 10 results when q' is issued to a search engine then
8:           insert q' into queue.
9:           canBeSimplified ← true
10:       end if
11:    end for
12:    if canBeSimplified = false then
13:       insert q into the result set simplifiedQuerySet.
14:    end if
15: end while
16: return simplifiedQuerySet
```

Jiang et al. [19] utilized query search to find the lexical alias and the query suggestion candidates by adapting the query search methods of Martin and Holte [33]. We will present the details of the work by Jiang et al. [19] in Section 2.3.

The discussion above shows the idea of query search: constructing different queries by combining different terms and evaluating these queries by issuing them to a search engine. Each query is regarded as a search node in a search tree, different combinations imply different search paths, and pruning is applied when a query does not meet some condition.

2.3 Query Suggestion by Query Search (QSQS)

Jiang et al. [19] proposed the Query Suggestion method by Query Search (QSQS). Their assumption and purpose have both been introduced in Chapter 1, an overview and the details of the QSQS system will be presented in this section.

The structure of the QSQS system is shown in Figure 2.1 [19]. For a user's original query Q_0, the *reference document collection* phase collects all the reference documents for Q_0 from the web, parses and processes them. Then for each reference document d, the *lexical alias search* phase analyzes d and finds a lexical alias for d. Different combinations of the terms in the lexical alias are evaluated to create a set of query suggestion candidates in the *query suggestion candidate search* phase. After processing all the reference documents, a greedy selection method is applied to the set of query suggestion candidates to finalize the query suggestions for Q_0 in the *query suggestion selection* phase. Details of each phase are introduced in the following subsections.

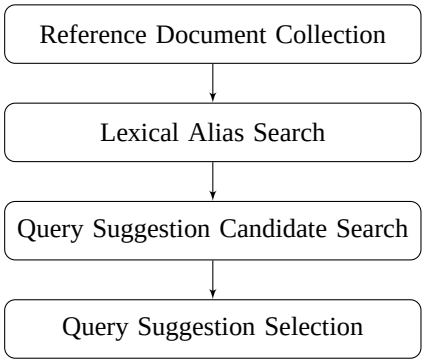

Figure 2.1: The structure of the QSQS system [19].

2.3.1 Reference Document Collection

Given the user's original query Q_0, the reference document collection phase retrieves raw reference documents using the Google API, applies HTML parsing, changes all capital letters to lower case, and removes stop words. Then every reference document is converted into a sequence of terms. A sample text and its processed result are given below as an example. Figure 2.2 shows the corresponding string vector which stores these terms in the program.

> **Sample text:** ITALY'S VOLCANOES: THE CRADLE OF VOLCANOLOGY, ITALY TRAVEL.
>
> **Processed text:** italy volcanoes cradle volcanology italy travel

italy	volcanoes	cradle	volcanology	italy	travel

Figure 2.2: The corresponding string vector for the text "italy volcanoes cradle volcanology italy travel".

2.3.2 Lexical Alias Search

Jiang et al. [19] think the title is often a summary of a reference document and it is given a higher weight than other parts of the reference document by the search engine. Similarly, the most frequent terms are important too, because the most frequent terms are likely to be related with the content of the reference document. Therefore, Jiang et al. [19] set the form of the lexical alias to be the title words plus the most frequent words.

For a reference document d, a sequence of seed terms consists of title terms and the most frequent terms (frequency of at least 3). If the same term appears more than once in the title, only the first occurrence is kept. For instance, if the title of document d is "gmail email from google gmail", and the most frequent terms in d are "access efficient spam mobile ...", then the sequence of seed terms for d is "gmail email from google access efficient spam mobile ...". Given a length l, a lexical alias

candidate is formed by taking the first l terms from the sequence of seed terms in order. Suppose $l = 5$, the lexical alias candidate from the previous example is "gmail email from google access". The length of the lexical alias candidate is set to be between 5 and 10 (l loops from 5 to 10). Once a lexical alias candidate is built up, it is tested whether it covers d. The loop stops and the current lexical alias candidate is returned as the lexical alias for d if it covers d. If none of the lexical alias candidates in the loop covers d, an empty string is returned. Following the example above, the first lexical alias candidate is "gmail email from google access". If it does not cover d, the next lexical alias candidate "google email from google access efficient" is tested. If the second one covers d, then "google email from google access efficient" is returned and the lexical alias search ends.

Algorithm 2.3 Lexical Alias Search by Jiang et al. [19].

Input: a reference document d (title and document body)
Output: a lexical alias $lexicalAlias$ for d

```
1:  if d = NULL then
2:      lexicalAlias ← NULL
3:  else
4:      initialize a list of terms termSequence and lexicalAlias to be empty.
5:      remove the duplicate words in the title.
6:      for i = 1 to title.length() do
7:          append the i-th word in the title to the end of termSequence.
8:      end for
9:      sort terms in the document body by frequency from highest to lowest, remove the terms which
        appear less than 3 times, store sorted terms in sortedTerms.
10:     for i = 1 to sortedTerms.length() do
11:         append the i-th word in sortedTerms to the end of termSequence.
12:     end for
13:     if termSequence.length() ≥ 5 then
14:         lexicalAliasCandidate ← NULL
15:         for i = 1 to 5 do
16:             append the i-th word in termSequence to the end of lexicalAliasCandidate.
17:         end for
18:         index ← 5
19:         while index ≤ 10 do
20:             if lexicalAliasCandidate covers d then
21:                 lexicalAlias ← lexicalAliasCandidate
22:                 break loop.
23:             else if index ≤ termSequence.length() then
24:                 append the index-th word in termSequence to the end of lexicalAliasCandidate.
25:                 increase index by 1.
26:             else
27:                 break loop.
28:             end if
29:         end while
30:     end if
31: end if
32: return lexicalAlias
```

Algorithm 2.3 shows the pseudocode for lexical alias search. Lines 1-2 return an empty string if the current reference document is empty. Lines 4-12 form the sequence of seed terms by adding the

title terms and the most frequent terms. Line 13 tests whether the sequence of seed terms contains less than 5 words. Lines 15-17 form the first lexical alias candidate by appending the first 5 terms from the sequence of seed terms. Line 20 evaluates whether a lexical alias candidate covers the current document. If it does, line 22 breaks the loop. Otherwise, lines 23-25 append one more word from the sequence of seed terms to the end of the lexical alias candidate and loop again. The loop ends when the length of the lexical alias candidate exceeds 10 or the terms in the sequence of seed terms have all been tried.

2.3.3 Query Suggestion Candidate Search

Given the lexical alias for a reference document d, Jiang et al. [19] proposed a depth-first query search method to try different combinations of terms in the lexical alias. Specifically, the lexical alias returned by the lexical alias search phase is considered to be the root query (start node) of the search tree. A child query (child node) is formed by deleting one term from its parent query (parent node) and the search goes one step deeper. If the child query covers the current document d, the subtree of this child query is expanded and the search goes on. Otherwise, the subtree of this child query is pruned and the search proceeds with its next sibling query. The process ends after searching over the entire space. During searching, all the queries that cover the current document d with lengths between 2 and 5 are collected into a query suggestion candidate set.

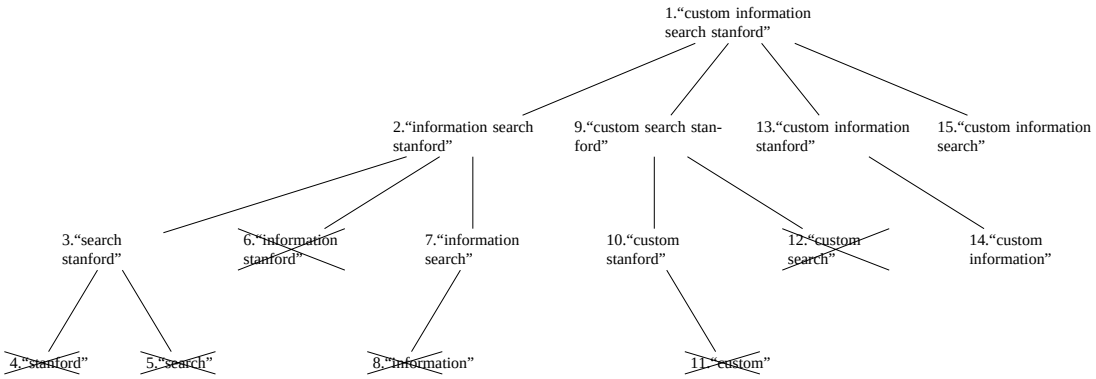

Figure 2.3: The query search tree for the list of search terms “custom information search stanford”, using the query search method of Jiang et al. [19].

For example, if the lexical alias is “custom information search stanford”, then the search tree using the query suggestion candidate search method by Jiang et al. [19] is like the one shown in Figure 2.3. The number on each node indicates the searching order. The root of the search tree is “custom information search stanford”, four child nodes are generated by deleting one word. Each node is tested as to whether it covers the current document d or not. The subtree of a node is expanded if it covers d and pruned if it does not. Nodes that are crossed out in Figure 2.3 are those which do not cover d, so the search does not continue below them. During the search, a table of visited queries is maintained to avoid duplicate searching. Because of this, for example, the right-

most node "custom information" on level 3 (suppose the root is on level 1) in Figure 2.3 is not expanded. At the end, all the nodes which are not crossed out and satisfy the length limit in the search tree are collected into a query suggestion candidate set.

2.3.4 Query Suggestion Selection

Given a set of query suggestion candidates, Jiang et al. [19] proposed a greedy selection method to determine the final query suggestions. The goal of the greedy selection is to maximize the MCC score. Query suggestion candidates which improve MCC the most are selected as the final query suggestions. Specifically, if K ($K = 10$) query suggestions are returned at the end, then there are K selection rounds in the greedy selection method. In each round, the query suggestion candidate which increases MCC the most is selected from the query suggestion candidate set, i.e. the system selects the query suggestion candidate which covers the most uncovered reference documents. If there is a tie, the contributions to MEC are compared and the bigger contributor is selected. After one round is processed, the query suggestion for this round is removed from the query suggestion candidate set, and the reference documents this query suggestion covers are marked as covered. After K rounds, the K query suggestions selected are supplied to the user.

Algorithm 2.4 Greedy Selection [19].

Input: the query suggestion candidate set QSC
Output: K ($K = 10$) final query suggestions QS

1: initialize QS and $coveredRefDoc$ to be empty.
2: **for** $round = 1$ **to** K **do**
3: initialize $mccContri$ and $mecContri$ to be 0, the query suggestion qs for this round and the reference document qs covers $qsCoveredRefDoc$ to be empty.
4: **for** $qsc \in QSC$ **do**
5: calculate the reference documents that qsc covers as $qscCoveredRefDoc$
6: **if** ($|qscCoveredRefDoc - coveredRefDoc| > mccContri$) **or** ($|qscCoveredRefDoc - coveredRefDoc| = mccContri$ **and** $|qscCoveredRefDoc| > mecContri$) **then**
7: $mccContri \leftarrow |qscCoveredRefDoc - coveredRefDoc|$
8: $mecContri \leftarrow |qscCoveredRefDoc|$
9: $qs \leftarrow qsc$
10: $qsCoveredRefDoc \leftarrow qscCoveredRefDoc$
11: **end if**
12: **end for**
13: insert qs into QS
14: remove qs from QSC
15: $coveredRefDoc \leftarrow (coveredRefDoc \bigcup qsCoveredRefDoc)$
16: **end for**
17: return QS

Algorithm 2.4 shows the pseudocode for the greedy selection method. Line 1 does initialization, and lines 2-16 process K rounds where lines 4-12 evaluate all the query suggestion candidates in the set and greedily select the best one.

2.4 The Google API

We use the Google API from the University Research Program for Google Search[3] to retrieve search results for queries in our experiments. Search results for queries can be retrieved by sending a GET request to the Google API over HTTP. Each Google API request consists of a base URL and several request parameters which are appended after the base URL as the URL-encoded query string arguments. The base URL is fixed as `http://research.google.com/university/search/service`. The request parameters indicate the desired search results. Table 2.1 gives the documentation for all the parameters. For instance, if we want to request the top 20 results for the query "google", we form a Google API request as "`http://research.google.com/university/search/service?clid=key-string&start=0&rsz=large&q=google`", where "key-string" is a secret key Google supplies to us. After the Google API receives this GET HTTP request, a response string is returned. By parsing the response string, we get the desired search results for a query.

Parameter	Description
clid	a secret key assigned by Google which must be included in every request to get access to the service from the Google API.
rsz	indicates the size of the desired search results, the only options available are "small" (10 results) and "large" (20 results).
start	indicates the position of the first search result returned, between 0 and 980 inclusively.
q	the URL-encoded search query.
lr	restricts the search to a particular language. For example, "&lr=lang_en" is for English, "&lr=lang_de" is for German.
snippets	adding "&snippets=true" will include snippets in the search results, otherwise, the results will contain no snippets.

Table 2.1: Google API request parameters (see Footnote 1).

2.4.1 The Instability of the Google API

We measure different query suggestion systems by MCC and MEC. We find that the score for the same query with the same method fluctuates irregularly over time. After looking into the problem, we trace this back to the instability of the Google API.

We say the Google API is unstable because if we request the top K_r ($K_r = 120$) search results for the same query twice, the Google API may supply us with two different sets of search results, even if the two copies of the query are issued within seconds of one another. There are two types of difference that occur in the search results. For the first type, the two sets of results contain the same web documents, only in different orders. For the second type, the two sets of results contain different web documents. The rest of this section explains why search results are unstable and analyzes the

[3]The Google API research documentation is at `http://research.google.com/university/search/docs.html`.

influence this instability has on our query suggestion systems and their evaluation.

We want the top K_r results for Q_0, but we cannot get them with one GET request, because the only options for the size of the requested results are "small" (10) and "large" (20). In order to get the top 120 results, we need to issue a request at least $120/20 = 6$ times. To be specific, we would need to request 20 results for Q_0 starting from 0 ("rsz=large", "q=Q_0", "start=0"), then request another 20 results for Q_0 starting from 20 ("rsz=large", "q=Q_0", "start=20"), and so on, until we get all K_r results for Q_0. We request 6 times consecutively when $K_r = 120$. If the Google API changes the search results between any of these requests, the results would be influenced.

In order to investigate the instability of the Google API, we designed and carried out an experiment. We requested the top 120 results for the same query twice consecutively (6 requests to get the top 120 results, then 6 more to get the top 120 results again), and compared the two sets of results. The Google API was tracked for 24 hours starting from Sep. 13th 2010 and the experimental data is shown in Table 2.3. The time column in Table 2.3 refers to the time that the first of the 12 (=6+6) requests to the Google API was submitted. We tested seven different queries (shown in column "query") in the experiment. For each query, we compared its two sets of search results from the Google API. There are three types of results in Table 2.3. "Exactly the same" means the two sets of search results are exactly the same. "Type 1 difference" means the contents of the two sets of results are the same but the order of some documents is different. "Type 2 difference" means there are some documents that only appear in one set of results. The data in Table 2.3 shows two things. First, the instability may happen no matter what query we use and when we issue the query. Second, the results are relatively stable, since most of the results are "exactly the same" and most times only one or two results are different for "type 2 difference".

The instability of the Google API influences the evaluation of our query suggestion system. With two different sets of reference documents for the same query, the same query suggestion system might produce different query suggestions and different MCC and MEC scores. However, Table 2.3 indicates these scores are only slightly unstable. In order to judge the validity of evaluating our query suggestion system with the unstable Google API, we carried out an experiment. In this experiment, we applied three query suggestion systems, QSQS, QSDC and GQSQS[4], on a set of 50 short queries (see Appendix A). Every system is run two times (shown in column "time" in Table 2.2). The MCC and MEC scores are averaged on the 50 queries. For each system, its second MCC (MEC) score is compared with its first MCC (MEC) score, and the difference is given in parentheses in Table 2.2. For example, the second MCC score for QSQS is 0.14 less than its first MCC score. From the table, we can see that the MCC (MEC) scores for the same system on the same queries are very similar. By contrast, the MCC (MEC) scores for different systems on the same queries have noticeable differences. This experiment shows that even if the Google API is unstable, a query suggestion

[4]QSQS is the query suggestion system proposed by Jiang et al. [19] which has been introduced in Section 2.3; QSDC is a query suggestion by document clustering and will be introduced in Chapter 4; GQSQS is a greedy query suggestion method by query search and will be described in Section 3.7.

system always has very similar MCC and MEC scores. In addition, good methods and bad methods are still distinguishable with big differences between the MCC and MEC scores. Therefore, we can still use our evaluation method even though the Google API is unstable.

Method	MCC	MEC	Time
QSQS	56.08	7.122	Apr 14 2010
QSQS	55.94 (-0.14)	7.144 (+0.022)	Apr 18 2010
QSDC	67.46	10.448	Apr 15 2010
QSDC	67.18 (-0.28)	10.546 (+0.098)	Apr 20 2010
GQSQS	65.18	9.748	Apr 30 2010
GQSQS	64.84 (-0.34)	9.812 (+0.064)	May 1 2010

Table 2.2: The influence of the instability of the Google API on our query suggestion systems.

Because of the instability of the Google API, we consider two systems "equally good" if their MCC (MEC) scores are close even if not exactly the same. Good and bad systems are still comparable since their MCC differences are often bigger than 5 and their MEC differences are often bigger than 1.

Table 2.3: Data for evaluating the instability of the Google API

query	result	time
volcanos in italy	Type 2 difference: 1/120 results are different.	Mon Sep 13 22:00:00 2010
google	Exactly the same.	Mon Sep 13 22:00:05 2010
watermelon art	Type 2 difference: 1/120 results are different.	Mon Sep 13 22:00:09 2010
herbs	Exactly the same.	Mon Sep 13 22:00:14 2010
oreo cookie	Type 2 difference: 1/120 results are different.	Mon Sep 13 22:00:19 2010
venice hotels	Exactly the same.	Mon Sep 13 22:00:24 2010
refrigerator magnets	Type 2 difference: 1/120 results are different.	Mon Sep 13 22:00:29 2010
volcanos in italy	Type 1 difference: only different order.	Mon Sep 13 23:00:00 2010
google	Type 2 difference: 4/120 results are different.	Mon Sep 13 23:00:05 2010
watermelon art	Exactly the same.	Mon Sep 13 23:00:10 2010
herbs	Type 1 difference: only different order.	Mon Sep 13 23:00:14 2010
oreo cookie	Type 2 difference: 2/120 results are different.	Mon Sep 13 23:00:19 2010
venice hotels	Exactly the same.	Mon Sep 13 23:00:27 2010
refrigerator magnets	Type 2 difference: 3/120 results are different.	Mon Sep 13 23:00:31 2010
volcanos in italy	Exactly the same.	Tue Sep 14 00:00:00 2010
google	Exactly the same.	Tue Sep 14 00:00:04 2010
watermelon art	Exactly the same.	Tue Sep 14 00:00:08 2010
herbs	Exactly the same.	Tue Sep 14 00:00:12 2010
oreo cookie	Exactly the same.	Tue Sep 14 00:00:17 2010
venice hotels	Exactly the same.	Tue Sep 14 00:00:21 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 00:00:25 2010
volcanos in italy	Exactly the same.	Tue Sep 14 01:00:05 2010
google	Exactly the same.	Tue Sep 14 01:00:09 2010
watermelon art	Exactly the same.	Tue Sep 14 01:00:13 2010
herbs	Exactly the same.	Tue Sep 14 01:00:17 2010
oreo cookie	Exactly the same.	Tue Sep 14 01:00:21 2010
venice hotels	Exactly the same.	Tue Sep 14 01:00:25 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 01:00:29 2010

Continued on next page

Table 2.3 – continued from the previous page

query	result	time
volcanos in italy	Exactly the same.	Tue Sep 14 02:00:01 2010
google	Exactly the same.	Tue Sep 14 02:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 02:00:06 2010
herbs	Exactly the same.	Tue Sep 14 02:00:09 2010
oreo cookie	Exactly the same.	Tue Sep 14 02:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 02:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 02:00:17 2010
volcanos in italy	Exactly the same.	Tue Sep 14 03:00:01 2010
google	Exactly the same.	Tue Sep 14 03:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 03:00:06 2010
herbs	Exactly the same.	Tue Sep 14 03:00:09 2010
oreo cookie	Exactly the same.	Tue Sep 14 03:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 03:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 03:00:16 2010
volcanos in italy	Exactly the same.	Tue Sep 14 04:00:00 2010
google	Exactly the same.	Tue Sep 14 04:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 04:00:05 2010
herbs	Exactly the same.	Tue Sep 14 04:00:08 2010
oreo cookie	Exactly the same.	Tue Sep 14 04:00:10 2010
venice hotels	Exactly the same.	Tue Sep 14 04:00:13 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 04:00:16 2010
volcanos in italy	Exactly the same.	Tue Sep 14 05:00:00 2010
google	Exactly the same.	Tue Sep 14 05:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 05:00:05 2010
herbs	Exactly the same.	Tue Sep 14 05:00:08 2010
oreo cookie	Exactly the same.	Tue Sep 14 05:00:10 2010
venice hotels	Exactly the same.	Tue Sep 14 05:00:13 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 05:00:15 2010
volcanos in italy	Type 2 difference: 1/120 results are different.	Tue Sep 14 06:00:01 2010
google	Type 2 difference: 1/120 results are different.	Tue Sep 14 06:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 06:00:06 2010
herbs	Exactly the same.	Tue Sep 14 06:00:09 2010
oreo cookie	Exactly the same.	Tue Sep 14 06:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 06:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 06:00:17 2010
volcanos in italy	Exactly the same.	Tue Sep 14 07:00:01 2010
google	Exactly the same.	Tue Sep 14 07:00:04 2010
watermelon art	Exactly the same.	Tue Sep 14 07:00:06 2010
herbs	Exactly the same.	Tue Sep 14 07:00:09 2010
oreo cookie	Exactly the same.	Tue Sep 14 07:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 07:00:14 2010
refrigerator magnets	Type 1 difference: only different order.	Tue Sep 14 07:00:17 2010
volcanos in italy	Exactly the same.	Tue Sep 14 08:00:00 2010
google	Exactly the same.	Tue Sep 14 08:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 08:00:05 2010
herbs	Exactly the same.	Tue Sep 14 08:00:08 2010
oreo cookie	Exactly the same.	Tue Sep 14 08:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 08:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 08:00:18 2010
volcanos in italy	Exactly the same.	Tue Sep 14 09:00:00 2010
google	Exactly the same.	Tue Sep 14 09:00:04 2010

Continued on next page

Table 2.3 – continued from the previous page

query	result	time
watermelon art	Exactly the same.	Tue Sep 14 09:00:07 2010
herbs	Exactly the same.	Tue Sep 14 09:00:10 2010
oreo cookie	Exactly the same.	Tue Sep 14 09:00:13 2010
venice hotels	Exactly the same.	Tue Sep 14 09:00:17 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 09:00:22 2010
volcanos in italy	Exactly the same.	Tue Sep 14 10:00:01 2010
google	Exactly the same.	Tue Sep 14 10:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 10:00:06 2010
herbs	Exactly the same.	Tue Sep 14 10:00:10 2010
oreo cookie	Exactly the same.	Tue Sep 14 10:00:12 2010
venice hotels	Exactly the same.	Tue Sep 14 10:00:15 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 10:00:18 2010
volcanos in italy	Exactly the same.	Tue Sep 14 11:00:00 2010
google	Exactly the same.	Tue Sep 14 11:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 11:00:05 2010
herbs	Exactly the same.	Tue Sep 14 11:00:08 2010
oreo cookie	Exactly the same.	Tue Sep 14 11:00:10 2010
venice hotels	Exactly the same.	Tue Sep 14 11:00:13 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 11:00:16 2010
volcanos in italy	Exactly the same.	Tue Sep 14 12:00:00 2010
google	Exactly the same.	Tue Sep 14 12:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 12:00:06 2010
herbs	Exactly the same.	Tue Sep 14 12:00:10 2010
oreo cookie	Exactly the same.	Tue Sep 14 12:00:12 2010
venice hotels	Exactly the same.	Tue Sep 14 12:00:15 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 12:00:19 2010
volcanos in italy	Exactly the same.	Tue Sep 14 13:00:00 2010
google	Exactly the same.	Tue Sep 14 13:00:03 2010
watermelon art	Type 1 difference: only different order.	Tue Sep 14 13:00:06 2010
herbs	Exactly the same.	Tue Sep 14 13:00:10 2010
oreo cookie	Exactly the same.	Tue Sep 14 13:00:12 2010
venice hotels	Exactly the same.	Tue Sep 14 13:00:16 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 13:00:19 2010
volcanos in italy	Exactly the same.	Tue Sep 14 14:00:01 2010
google	Exactly the same.	Tue Sep 14 14:00:05 2010
watermelon art	Exactly the same.	Tue Sep 14 14:00:08 2010
herbs	Exactly the same.	Tue Sep 14 14:00:11 2010
oreo cookie	Exactly the same.	Tue Sep 14 14:00:14 2010
venice hotels	Exactly the same.	Tue Sep 14 14:00:17 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 14:00:22 2010
volcanos in italy	Exactly the same.	Tue Sep 14 15:00:00 2010
google	Exactly the same.	Tue Sep 14 15:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 15:00:05 2010
herbs	Exactly the same.	Tue Sep 14 15:00:08 2010
oreo cookie	Exactly the same.	Tue Sep 14 15:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 15:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 15:00:18 2010
volcanos in italy	Exactly the same.	Tue Sep 14 16:00:00 2010
google	Exactly the same.	Tue Sep 14 16:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 16:00:06 2010
herbs	Exactly the same.	Tue Sep 14 16:00:08 2010

Continued on next page

Table 2.3 – continued from the previous page

query	result	time
oreo cookie	Exactly the same.	Tue Sep 14 16:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 16:00:13 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 16:00:16 2010
volcanos in italy	Type 2 difference: 2/120 results are different.	Tue Sep 14 17:00:00 2010
google	Type 2 difference: 1/120 results are different.	Tue Sep 14 17:00:06 2010
watermelon art	Type 2 difference: 1/120 results are different.	Tue Sep 14 17:00:12 2010
herbs	Type 1 difference: only different order.	Tue Sep 14 17:00:17 2010
oreo cookie	Type 2 difference: 1/120 results are different.	Tue Sep 14 17:00:22 2010
venice hotels	Exactly the same.	Tue Sep 14 17:00:27 2010
refrigerator magnets	Type 2 difference: 1/120 results are different.	Tue Sep 14 17:00:31 2010
volcanos in italy	Exactly the same.	Tue Sep 14 18:00:01 2010
google	Exactly the same.	Tue Sep 14 18:00:04 2010
watermelon art	Type 1 difference: only different order.	Tue Sep 14 18:00:07 2010
herbs	Exactly the same.	Tue Sep 14 18:00:11 2010
oreo cookie	Exactly the same.	Tue Sep 14 18:00:14 2010
venice hotels	Type 2 difference: 8/120 results are different.	Tue Sep 14 18:00:23 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 18:00:26 2010
volcanos in italy	Exactly the same.	Tue Sep 14 19:00:00 2010
google	Exactly the same.	Tue Sep 14 19:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 19:00:06 2010
herbs	Exactly the same.	Tue Sep 14 19:00:09 2010
oreo cookie	Exactly the same.	Tue Sep 14 19:00:12 2010
venice hotels	Exactly the same.	Tue Sep 14 19:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 19:00:17 2010
volcanos in italy	Exactly the same.	Tue Sep 14 20:00:00 2010
google	Exactly the same.	Tue Sep 14 20:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 20:00:06 2010
herbs	Exactly the same.	Tue Sep 14 20:00:08 2010
oreo cookie	Exactly the same.	Tue Sep 14 20:00:11 2010
venice hotels	Exactly the same.	Tue Sep 14 20:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 20:00:17 2010
volcanos in italy	Exactly the same.	Tue Sep 14 21:00:00 2010
google	Exactly the same.	Tue Sep 14 21:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 21:00:06 2010
herbs	Exactly the same.	Tue Sep 14 21:00:09 2010
oreo cookie	Exactly the same.	Tue Sep 14 21:00:12 2010
venice hotels	Exactly the same.	Tue Sep 14 21:00:16 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 21:00:19 2010
volcanos in italy	Exactly the same.	Tue Sep 14 22:00:01 2010
google	Exactly the same.	Tue Sep 14 22:00:03 2010
watermelon art	Exactly the same.	Tue Sep 14 22:00:06 2010
herbs	Exactly the same.	Tue Sep 14 22:00:09 2010
oreo cookie	Exactly the same.	Tue Sep 14 22:00:12 2010
venice hotels	Exactly the same.	Tue Sep 14 22:00:14 2010
refrigerator magnets	Exactly the same.	Tue Sep 14 22:00:17 2010

Chapter 3

Improved and Greedy Query Suggestion by Query Search (IQSQS & GQSQS)

3.1 Introduction

We introduced the Query Suggestion method by Query Search (QSQS) proposed by Jiang et al. [19] in Chapter 2. In QSQS, a *lexical alias* for every reference document is calculated. The lexical alias contains terms that are the basis of the query search. Since terms in the lexical alias are combined to build different query suggestion candidates, we think their lexical alias search is simply a method to supply *search term*s. We use "search terms" to refer to terms selected from reference documents and that are important to form query suggestions with high MCC and MEC scores. Based on this observation, we propose an Improved Query Suggestion method by Query Search (IQSQS) whose main improvement is to replace the lexical alias search phase in QSQS by a *search term selection* phase. In addition to the improvements made to each phase of QSQS, we modify the control structure of QSQS in a greedy way and suggest another improved method, Greedy Query Suggestion by Query Search (GQSQS). The experimental results demonstrate the superiority of IQSQS and GQSQS over QSQS.

In this chapter, we will describe the system structure of the IQSQS system and then consider the alternative implementations of its various components. Experimental comparisons of the alternatives are then presented. GQSQS is described after IQSQS and a comparison of QSQS, IQSQS and GQSQS is given to end this chapter.

3.2 System Overview

The structure of the IQSQS system is shown in Figure 3.2. Figure 3.1 [19] presents the structure of the QSQS system for comparison. Algorithm 3.1 is the pseudocode of IQSQS. For the user's original query Q_0, the *reference document collection* phase collects all the reference documents for

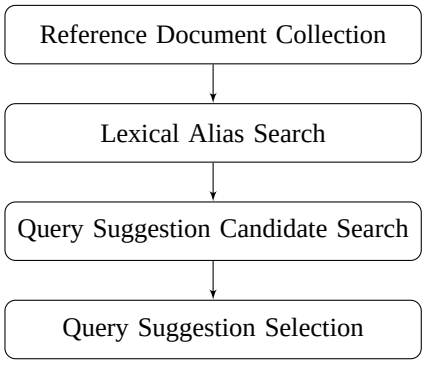

Figure 3.1: The structure of the QSQS system [19].

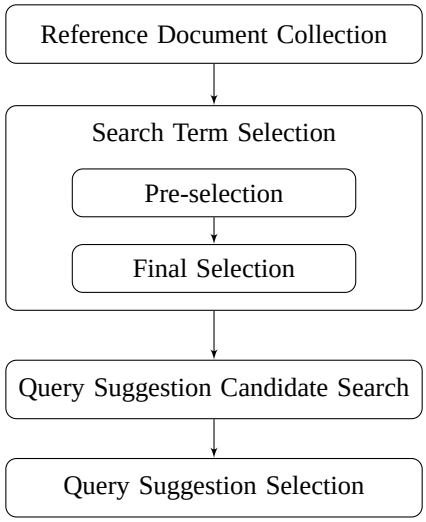

Figure 3.2: The structure of the IQSQS system.

Q_0 from the web, parses and processes them (line 2). For each reference document d, the *search term selection* phase contains a *pre-selection* step and a *final selection* step to generate a list of search terms for d (line 4). Different combinations of search terms are evaluated to create a set of query suggestion candidates in the *query suggestion candidate search* phase (line 5). After processing all the reference documents, a greedy selection method is applied to the query suggestion candidate set to finalize the query suggestions for Q_0 in the *query suggestion selection* phase (line 7).

Algorithm 3.1 Improved Query Suggestion by Query Search (IQSQS)

Input: the original query Q_0
Output: K ($K = 10$) query suggestions
1: initialize the query suggestion candidate set QSC to be empty.
2: collect all the reference documents D ($|D| = 100$).
3: **for** $i = 1$ **to** $|D|$ **do**
4: select a list of search terms ST_i for the ith reference document.
5: apply query search on ST_i and add query suggestion candidates for the ith reference document to QSC.
6: **end for**
7: greedily select K query suggestions from QSC and insert them into the query suggestion set QS.
8: **return** QS

3.3 Stemming

Stemming is often used to process texts in the *information retrieval* field. We do not apply stemming in the reference document collection phase in IQSQS. This section gives a simple introduction of stemming and explains why we do not use it.

In linguistic morphology, "the goal of stemming is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form" [32]. For instance, "stemming",

"stemmer", "stemmed" are all based on "stem". Terms with the same root usually have similar meanings, such as "connect", "connection", "connected" and "connecting" [32]. "The performance of the information retrieval system will be improved if these words are grouped together" [42], and stemming is often applied for this purpose [32, 42].

"A stemming algorithm usually refers to a crude heuristic process that chops off the ends of words in the hope of reducing all the words with the same stem to a common form correctly most of the time, and often includes the removal of derivational affixes" [30, 32]. Stemming algorithms are often called stemmers. The most popular English stemmer is the Porter stemming algorithm [42] which is the defacto standard English stemmer. The Porter stemmer[1] has a group of pre-defined rules, different rules are applied in different contexts. We cite an example of Manning et al. [32] below to show the different rules and the way to apply them. In addition, "the Porter stemmer measures whether it is reasonable to remove the suffix. For instance, the word 'replacement' is reduced to 'replac', but 'cement' is not reduced to 'c'" [32]. Nevertheless, the Porter stemmer still cannot achieve perfection as defined by linguists. For example, it mistakenly stems "october" back to "octob".

Rule	**Example**
SSES $\rightarrow$ SS	caresses $\rightarrow$ caress
IES $\rightarrow$ I	ponies $\rightarrow$ poni
SS $\rightarrow$ SS	caress $\rightarrow$ caress
S $\rightarrow$	cats $\rightarrow$ cat

The Porter stemming algorithm may improve the MCC and MEC score of IQSQS, on the other hand it may lower the quality of query suggestions by making them incomprehensible for humans. We ran an experiment to compare the effects of stemming in IQSQS. The search term selection method applied in the experiment is LAW and the query suggestion candidate search method is QoAC, both will be introduced in the following sections. There are two sets of runs, stemming on (with stemming) and stemming off (without stemming). Each set was run on a group of 50 short queries and a group of 50 long queries (see Appendix A). The average MCC and MEC scores on each group of 50 queries are reported in Table 3.1.

Stemming option	**Short queries**	**Long queries**
stemming off	MCC=67.08 MEC=8.82	MCC=73.12 MEC=9.88
stemming on	MCC=63.72 MEC=8.25	MCC=70.78 MEC=9.63

Table 3.1: The average MCC and MEC scores on 50 queries for stemming on and stemming off in the IQSQS system.

From Table 3.1, IQSQS with stemming on has lower MCC and MEC scores on both short queries and long queries than with stemming off. Since different query data make MCC and MEC fluctuate, the sign test from statistics is adopted to ensure our conclusion. The MCC (MEC) scores of

[1]The Porter stemming algorithm is at `http://tartarus.org/~martin/PorterStemmer/`.

switching off and on stemming are considered two random variables, the values on 50 queries are 50 samples, the number of times that one is better than the other is counted, and the p-value (two-tailed) is reported in Table 3.2.

	Short queries	**Long queries**
MCC	p-value = 3.10E-07	p-value = 0.31
MEC	p-value = 9.85E-05	p-value = 0.68

Table 3.2: The p-values for sign tests to compare the effect of stemming in IQSQS.

We set $\alpha = 0.05$, then from Table 3.2, the result is significant that stemming on and off perform differently for short queries. However, for long queries, we fail to reject the null hypotheses and the data does not provide sufficient evidence to conclude that stemming on and off perform differently. To sum up, stemming off works better than stemming on. Additionally, stemming decreases the understandability of query suggestions. Therefore, we do not apply stemming in IQSQS[2].

3.4 Search Term Selection

Each reference document is converted into a sequence of terms after the reference document collection phase. For each reference document, a list of search terms T is selected and returned in the search term selection phase. The search term selection phase consists of two sub-phases: the pre-selection phase and the final selection phase. The pre-selection phase shrinks a whole reference document, which contains hundreds or thousands of terms, to a document with around 20 terms. Then the final selection is applied to select final search terms from these terms. In this section, we first introduce the pre-selection phase, then define two scoring mechanisms used in the final selection phase. The rest of this section describes two final selection methods: the Lexical Alias Word method (LAW) and the Coverage Score Word method (CSW).

3.4.1 Pre-selection

The final selection phase extracts search terms by scoring all the terms in a reference document and ranking them by their scores from highest to lowest. Because the scoring involves requesting to the Google API which is slow, the final selection will be too slow if there are hundreds or thousands of terms to evaluate. Therefore, a pre-selection phase is required to decrease the size of the reference document.

Most term selection methods can be applied here. For example, one could select the most frequent terms in a reference document or select the terms which appear close to terms in Q_0, etc. In the IQSQS system, we use the *snippet*s supplied by the search engine as the pre-selection method.

Given an original query Q_0 and a search result r, a snippet corresponding to Q_0 and r is a piece of text extracted from the web document of r by the search engine. The snippet helps the user

[2]In the next chapter, we propose a query suggestion method by document clustering, and we use stemming in the document clustering phase.

recognize the content of the website so that the user can decide whether s/he wants to navigate to it or not. Figure 3.3 shows one search result of the query "jaguar" by Google.com, the text under the title is the corresponding snippet.

Figure 3.3: One search result for the query "jaguar" from Google.com on Oct. 5th, 2010. The snippet is under the title "Jaguar - Wikipedia, the free encyclopedia", begins with "The jaguar (Panthera once) ...", and ends with "The jaguar is the ...".

There are three reasons why we use snippets in the IQSQS system. First, a snippet often contains around 20 terms which is a suitable size for the final selection phase. Second, snippets need no calculation on our part. Third, snippets supply terms that might form query suggestions with high MCC and MEC scores. We ran an experiment to compare the results of pre-selecting terms using frequency and using snippets in the IQSQS system. Similar with the experimental setting and method in Section 3.3, the average MCC and MEC scores on 50 short and 50 long queries (see Appendix A) are shown in Table 3.3, and the p-values of sign tests are shown in Table 3.4.

Pre-selection method	**Short queries**	**Long queries**
snippets	MCC=67.08 MEC=8.82	MCC=73.12 MEC=9.88
frequency	MCC=66.36 MEC=8.63	MCC=73.20 MEC=9.80

Table 3.3: The average MCC and MEC scores for pre-selecting using snippets and using frequency in IQSQS.

The performance of pre-selecting using snippets appears no different from that of pre-selecting using frequency. Sign tests are carried out by considering the MCC (MEC) scores of pre-selecting using snippets and using frequency as two random variables. The performances on 50 queries are 50 samples for each method. From Table 3.4, we fail to reject the null hypotheses. Together with the indistinguishable results in Table 3.3, we consider pre-selecting using snippets performs equally good with using frequency. Because of other advantages snippets have (given above), we use snippets as the pre-selection method in IQSQS.

	Short queries	**Long queries**
MCC	p-value=1.00	p-value=0.29
MEC	p-value=1.00	p-value=0.66

Table 3.4: The p-values for sign tests to compare pre-selecting using snippets and using frequency.

3.4.2 Final Selection

Given around 20 terms after pre-selection, the final selection phase extracts 10 terms and ranks them from highest to lowest. There are two scoring mechanisms we use in the final selection phase: the OC score (Overall Cover) and the LA score (Lexical Alias). For term t, the OC score is the number of reference documents that t covers when appended to Q_0, and the LA score is a binary value indicating whether or not t covers the current document[3] when appended to Q_0. For example, suppose Q_0 is "jaguar", the current reference document is d, and t is the term "car" in d. If the query "jaguar car" covers 10 reference documents including d, then $OC(t, D) = 10$, where D is the set of all reference documents, and $LA(t, d) = 1$.

There are two reasons why we use the original query Q_0 in OC and LA. First, we want to evaluate a term's potential of covering reference documents. However, a term is often unlikely to cover reference documents by itself. For instance, the term "car" is highly related to the query "jaguar", but the query "car" almost returns nothing about "jaguar" in the top results. Appending the term to the end of Q_0 helps out and is simple to do. Second, in addition to improving the MCC and MEC score, we also hope our query suggestions can convey more information about Q_0. For example, suppose Q_0 is "jaguar", we hope our query suggestions could be "jaguar car", "jaguar cat", "jaguar mac os", etc. which reflect the subcategories of the query "jaguar". Bonding Q_0 with each subcategory, such as "car" or "cat", helps creating specific query suggestions.

Lexical Alias Word (LAW)

Referring back to Chapter 2, Jiang et al. [19] restrict the form of a lexical alias to be the title words plus the most frequent words, but this form is not guaranteed to work the best. Furthermore, words that are neither title words nor the most frequent words might still be useful for query suggestions. Therefore, we propose the Lexical Alias Word (LAW) method which uses the LA score to evaluate all words. For a reference document d, a word in d is added to the search term set for d if its LA score is 1. After all the words in d are evaluated, the words in the search term set are ranked by their OC scores from highest to lowest. The first 10 search terms from the set are returned. If the search term set contains less than 10 words, all of them are returned. For example, for a reference document about "google custom search information technology services", suppose there are only four words "custom", "information", "search", and "stanford" with LA as 1. If their OC scores rank them as "custom information search stanford" from highest to lowest, then the list of search terms for this reference document is "custom information search stanford".

A special case is when there is no single word whose LA score is 1. Our program returns an empty word list and no query suggestion candidates will be constructed for this case. However, this case does not threaten our query suggestion method, because the probability that a reference

[3]Remember that IQSQS processes reference documents individually, a list of search terms is selected for each reference document.

document returns an empty list in our experiments is only around 3.7% (the data is based on 50 short queries and 50 long queries) and there are enough reference documents to generate query suggestions.

Algorithm 3.2 is the pseudocode of the LAW method. Lines 1-2 return an empty list if a reference document is empty. Line 4 does initialization. Line 5 collects all the words from a reference document into a word set. Lines 6-10 calculate the OC and LA scores for all the words and insert those whose LA score is 1 into a search term set. Lines 11-12 returns an empty list if the search term set is empty. Line 14 sorts the words in the search term set by their OC scores from highest to lowest. Lines 15-19 return the list of search terms.

Algorithm 3.2 Lexical Alias Word (LAW).

Input: a reference document d
Output: a list of search terms T

```
1:  if d = NULL then
2:      T ← NULL
3:  else
4:      initialize searchTermSet and wordSet to be empty.
5:      insert all the words in d into wordSet.
6:      for w ∈ wordSet do
7:          if LA(w) = 1 then
8:              add w to searchTermSet
9:          end if
10:     end for
11:     if searchTermSet is empty then
12:         return an empty list.
13:     else
14:         sort the words in searchTermSet by their OC scores from highest to lowest.
15:         if searchTermSet contains at least 10 terms then
16:             return the first 10 words in searchTermSet.
17:         else
18:             return all the words in searchTermSet.
19:         end if
20:     end if
21: end if
```

Coverage Score Word (CSW)

The lexical alias search method of Jiang et al. [19] and the LAW method both utilize the idea of lexical aliases and focus on covering the current reference document. For a reference document d, Jiang et al.'s method [19] looks for a lexical alias for d, and LAW returns search terms which cover d when appended to Q_0. However, we want to cover more reference documents rather than only one. Therefore, the Coverage Score Word method (CSW) is proposed to evaluate a word by its potential of covering more reference documents, i.e. the CSW method selects words by global coverage rather than local coverage.

Equation 3.1 is the coverage score function to measure the word's potential for global coverage.

$$CoverageScore(w, d) = \omega_1 \times OC(w, D) + \omega_2 \times LA(w, d) \tag{3.1}$$

The coverage score function is a linear combination of OC and LA. w is the word, d is the current reference document, D is the set of all reference documents, ω_1 and ω_2 are the weights ($\omega_1 = 0.25$ and $\omega_2 = 0.75$). For example, if the OC and LA score of word t in reference document d is 6 and 1 respectively, then the coverage score is

$$CoverageScore(t, d) = 0.25 \times 6 + 0.75 \times 1 = 2.25.$$

For each reference document, all the words in it are ranked by their coverage scores from highest to lowest, the top scoring 10 words are returned as the search terms. If there are fewer than 10 words, all of them are returned. The pseudocode of the CSW method is shown in Algorithm 3.3. Lines 1-2 return an empty search term list if d is empty. Line 4 does initialization. Lines 6-9 calculate the coverage score for every word in d. Line 10 sorts all the words by their coverage scores from highest to lowest. Lines 11-15 return the final search terms.

Algorithm 3.3 Coverage Score Word (CSW)

Input: a reference document d, weights ω_1 and ω_2.
Output: a list of search terms T

```
1:  if d = NULL then
2:      T ← NULL
3:  else
4:      initialize wordSet to be empty.
5:      insert all the words in d into wordSet.
6:      for w ∈ wordSet do
7:          calculate the OC and LA scores of w.
8:          CoverageScore(w) = ω1 × OC(w) + ω2 × LA(w)
9:      end for
10:     sort all the words in wordSet by their coverage scores from highest to lowest.
11:     if wordSet contains at least 10 words then
12:         return the first 10 words in wordSet.
13:     else
14:         return all the words in wordSet.
15:     end if
16: end if
```

3.5 Query Suggestion Candidate Search

For every reference document, the search term selection phase provides a list of search terms to the query suggestion candidate search phase, where query suggestions candidates are built up by combining search terms. There are many ways to combine a list of terms. Adding one term may make a query more specific and cover more reference documents related with the term. At the same time, it also makes a query lose generalization and therefore might lose some reference documents

which were covered before. Taking out one term can have similar effects. In this section, we first explain why we fix the order of search terms in the query suggestion candidates. Then different query suggestion candidate search methods are described and compared.

3.5.1 The Order of Search Terms

The search term selection phase selects a list of sorted search terms. The order of search terms in the list is important for our query search. There are two reasons for this. First, the order of terms in a query influences the corresponding search results. For instance, the query "jaguar car" and the query "car jaguar" return two different sets of search results. Second, the time cost of trying all permutations of search terms is too high (there are 3,628,800 different permutations for 10 search terms). Therefore, we keep the order of search terms in the query suggestion candidates the same as their order in the search term list. For example, given a sorted search term list $w_1 w_2 ... w_i ... w_j ... w_n$ ($i < j < n$), for query suggestion candidates containing both w_i and w_j, w_i always appears before w_j.

3.5.2 Adding Combination (AC)

In the IQSQS system, we propose a query search method named Adding Combination (AC). If we only consider the searching method, then AC is the same as the query search method of Jiang et al. [19] except that one is top-down and the other is bottom-up. But in IQSQS, AC is different from the query search method of Jiang et al. [19] in two ways. First, Jiang et al. [19] search queries from longest to shortest, and prune the sub-tree of a node if the node does not cover the current document d. AC searches all the queries without pruning. Second, the purpose of the method by Jiang et al. [19] is different from the purpose of AC. Jiang et al. [19] proposed the query search method in QSQS to process a lexical alias whose length was around 7, and they limited the length of the query suggestion candidate to be between 2 and 5. As opposed to that, AC is proposed to process a list of search terms whose length is around 10, and we limit the length of the query suggestion candidate to be between 1 and 3 excluding Q_0. If we replace AC with Jiang et al.'s [19] method, because it is top-down, the valid query nodes will all be at the bottom of the search tree. Moreover, since the length range $[1, 3]$ is not wide, the pruning in Jiang et al.'s method [19] may miss potential query suggestions. On the other hand, AC might take too much time to search all queries with lengths between 2 and 5. In short, Jiang et al.'s method [19] is more suitable for QSQS and AC is more suitable for IQSQS.

Figure 3.4 shows a search tree of AC when the search terms are "custom information search stanford". The dashed lines from the root node are not the searching paths, they only mean that each search tree below is from the root. The numbers on the nodes indicate the order of the search. The search for each tree stops after the last term "stanford" is added. A table of visited queries is used to avoid duplicate searching. All the queries within the length limit [1,3] are inserted into a query

suggestion candidate set.

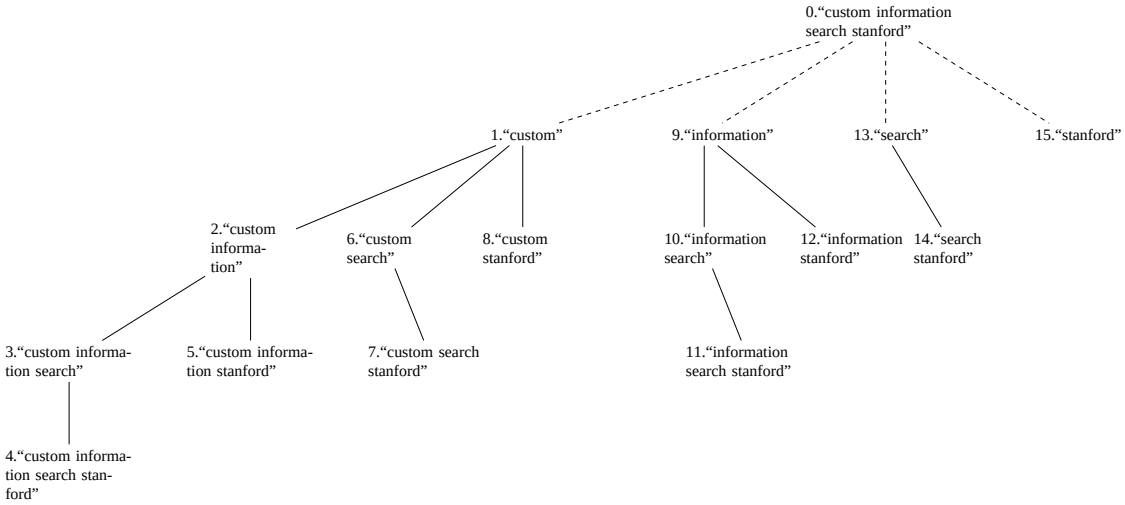

Figure 3.4: Query search tree for a list of search terms “custom information search stanford”, using the AC method.

3.5.3 Q_0 and Adding Combination (QoAC)

The terms in the original query Q_0 are special compared with the search terms selected from the reference document. In order to evaluate the effect of Q_0 in query suggestion candidates, we propose the Q_0 and Adding Combination (QoAC) method. QoAC differs from AC in that Q_0 is always added at the beginning of query suggestion candidates.

For example, suppose the list of search terms is “custom information search stanford” and Q_0 is “google”, then there will be four search trees with the root queries “google custom”, “google information”, “google search”, and “google stanford”. Under the node “google custom”, the child queries are “google custom information”, “google custom search”, and “google custom stanford”. The grandchild queries are “google custom information search”, “google custom information stanford”, etc. Except for Q_0, the search tree of QoAC is the same as that of AC.

3.5.4 Beam Search (BS)

Before we introduce the Beam Search (BS) method, the two terms *width* and *depth* for a search tree need to be defined. The depth of a search tree is the number of layers from the root node to the farthest leaf node. The width of a search tree is defined as the maximum number of nodes on the same layer. For example, in the search tree shown in Figure 3.4, the depth of the search tree is 5 and the width is 6.

The query search method of Jiang et al. [19] and AC both suffer from the complexity of combining many terms. Because of this, we limit the maximum length of the search term list to be 10. However, by ignoring the search terms after the top 10, we risk losing useful terms. BS is used

to balance the risk and the time cost. In a given amount of time, a fixed quantity of nodes can be searched. We let the BS method search a similar number of nodes with Jiang et al.'s method [19] and AC so that BS takes similar time to finish. Jiang et al.'s method [19] and AC search exhaustively (though Jiang et al. [19] apply pruning), they expand all nodes on one layer and search all the child nodes on the next layer. BS expands all nodes on one layer too, but it sorts the generated child nodes by a heuristic function (we use the OC score), retains the best W child nodes and prunes the others. W is the pre-defined width of the search tree[4]. When W is unlimited, the BS method becomes the AC method.

We think that the BS method may work better than the method of Jiang et al. [19] and AC, because BS evaluates more search terms. The only situation in which BS might miss a good query suggestion candidate is like this: suppose query $t_1 + t_2$ is pruned for bad performance, but query $t_1 + t_2 + \cdots + t_i$ ($i > 2$) is a really good query; it will not appear in the search tree of BS. However, this kind of situation is unlikely to happen. Because if adding t_i could turn a bad query suggestion candidate into a good one, then in this case terms t_1 and t_2 are probably not important, and BS would still be expected to find good query suggestion candidates containing term t_i.

3.5.5 Q_0 and Beam Search (QoBS)

The relation between QoBS and BS is the same as the relation between QoAC and AC. The original query Q_0 is always added in the beginning of query suggestion candidates.

3.6 Experiment to Find the Best Configuration of IQSQS

We have introduced the LAW and CSW search term selection methods, and the AC, QoAC, BS and QoBS query search methods. In order to evaluate these methods and find out the best match for our query suggestion objective, an experiment is carried out.

As discussed before, we fix the following settings in our experiment. Reference documents are processed with stemming off in the reference document collection phase, and terms are pre-selected using snippets in the search term selection phase. The experiment evaluates each combination of search term selection methods (LAW and CSW) and query search methods (AC, QoAC, BS and QoBS). The average MCC and MEC scores for each combination of methods are shown in Table 3.5. The test queries in this experiment are 50 short queries[5] (see Appendix A).

It is clear from Table 3.5 that LAW is inferior to CSW. For both AC and BS, CSW improves MCC by around 20 and MEC by more than 2 compared to LAW. The increase in MCC indicates that CSW helps query suggestions cover around 20 more reference documents than LAW does when the initial query is not automatically included in the query suggestions. The increase in MEC implies that each query suggestion covers 2 more reference documents on average in these cases. Sign tests

[4] In our implementation, we start with 20 terms and limit the width to be 15.

[5] We run on short queries to find the best configuration and run the best configuration on long queries which is reported in Section 3.8.

	LAW	CSW
AC	MCC=37.82 MEC=4.83	MCC=56.66 MEC=7.26
QoAC	MCC=67.08 MEC=8.82	MCC=70.88 MEC=8.99
BS	MCC=34.36 MEC=4.35	MCC=53.90 MEC=6.69
QoBS	MCC=66.06 MEC=8.46	MCC=70.82 MEC=9.15

Table 3.5: The average MCC and MEC score for each combination of methods in IQSQS.

are used to compare CSW and LAW. With four different search term selection methods (AC, QoAC, BS, and QoBS), the MCC and MEC scores of CSW and LAW are compared and the p-value for each test is calculated. All the results are shown in Table 3.6.

	MCC	MEC
AC	p-value = 1.78E-15	p-value = 1.78E-15
QoAC	p-value = 3.80E-03	p-value = 0.12
BS	p-value = 9.06E-14	p-value = 2.27E-12
QoBS	p-value = 6.98E-11	p-value = 5.61E-06

Table 3.6: The p-values for sign tests to compare CSW and LAW.

In Table 3.6, only 1 out of 8 tests fails to reject the null hypothesis. Therefore, we conclude that CSW performs better than LAW and constantly use it as the search term selection method in all the following experiments. LAW selects a term based on whether it can cover one reference document. As opposed to that, CSW selects terms based on their potential of covering more reference documents. LAW focuses on local coverage and hence may miss terms that are important for global coverage.

For QoAC and QoBS, CSW still brings an improvement to LAW, though amounting to a much smaller increase compared to the cases of AC and BS. We attribute the decreased improvement to the original query Q_0 being included in QoAC and QoBS for two reasons. First, though AC and BS do not contain Q_0 automatically, most of their final query suggestions still contain terms in Q_0. The data in our experiment shows that more than 90% of the final query suggestions for AC and BS contain at least one term in Q_0. This data shows the superiority of terms in Q_0 over other terms from reference documents. Though the lower scores of AC and BS also indicate that it is not enough to contain terms in Q_0 to achieve higher scores. Second, as we attempt to cover reference documents D of Q_0, queries with the form Q_0 plus other terms tend to specify Q_0 and cover a portion of D. As opposed to that, queries of other forms, even including terms in Q_0, are likely to cover relevant documents which might not be from D. For example, suppose Q_0 is "volcanos in italy", the query "volcanos in italy information" is likely to cover more reference documents than the query "italy volcanos information" and the query "information italy volcanos". We compared the query suggestions "volcanos in italy watched" and "watched volcano italy" (the original query is "volcanos in italy") and found that "volcanos in italy watched" covers four more reference documents than "watched volcano italy", though they are very similar from humans' perspective.

From Table 3.5, another clear conclusion is that QoAC and QoBS are superior to AC and BS. This implies that adding the original query Q_0 helps to improve the MCC and MEC scores. The conclusion is consistent with our analysis in the previous paragraph and also strengthens the same conclusion drawn by Jiang et al. [19]. Additionally, sign tests are carried out to compare methods with Q_0 (QoAC and QoBS) to those without Q_0 (AC and BS). The p-values reported in Table 3.7 supply sufficient evidence to conclude that methods with Q_0 work differently from methods without Q_0. Therefore, for all the following experiments, we adopt query search methods with Q_0.

	MCC	**MEC**
QoAC vs AC	p-value = 3.48E-13	p-value = 9.02E-05
QoBS vs BS	p-value = 1.78E-15	p-value = 2.27E-12

Table 3.7: The p-values for sign tests to compare methods with Q_0 and those without Q_0.

	MCC	**MEC**
QoBS-CSW vs QoAC-CSW	p-value = 1.00	p-value = 0.89

Table 3.8: The p-values for the sign test to compare CSW-QoBS and CSW-QoAC.

There are two combinations of methods left to be compared: CSW-QoBS and CSW-QoAC. Their average MCC and MEC scores are close in Table 3.5. The p-values of the sign test reported in Table 3.8 could not reject the null hypotheses that CSW-QoBS and CSW-QoAC perform similarly either. Therefore, we ran CSW-QoBS and CSW-QoAC on 50 long queries and reported the scores and the p-values in Table 3.9 and 3.10. The results on long queries demonstrate the superiority of CSW-QoBS over CSW-QoAC. Therefore, we conclude that CSW-QoBS is better than CSW-QoAC.

	QoBS	**QoAC**
CSW	MCC=76.33 MEC=10.51	MCC=73.83 MEC=9.70

Table 3.9: The average MCC and MEC score for CSW-QoBS and CSW-QoAC on long queries.

From the discussion above, our general conclusion is that CSW is better than LAW to select search terms and methods including Q_0 are better than those without it. The best configurations for IQSQS is CSW-QoBS and is referred to as IQSQS*.

3.7 Greedy Query Suggestion by Query Search (GQSQS)

The QSQS system by Jiang et al. [19] processes all reference documents individually to accumulate a set of query suggestion candidates and, at the very end, selects the best K as the final query suggestions. The IQSQS system made improvements on each phase of the QSQS system, but did not change the order in which the phases were executed. We now consider a method, Greedy Query Suggestion by Query Search (GQSQS), that changes the control structure of QSQS [19].

Algorithm 3.4 is the pseudocode of GQSQS. After the reference documents are collected, there are K rounds to greedily find the best K query suggestions. In each round, the search term selection

	MCC	MEC
QoBS-CSW vs QoAC-CSW	p-value = 4.02E-4	p-value = 6.60E-5

Table 3.10: The p-values for the sign test to compare CSW-QoBS and CSW-QoAC on long queries.

Algorithm 3.4 Greedy Query Suggestion by Query Search (GQSQS)

Input: the original query Q_0
Output: K ($K = 10$) query suggestions

1: initialize the query suggestion set QS to be empty.
2: collect all the reference documents D ($|D| = 100$) for Q_0.
3: **for** $i = 1$ **to** K **do**
4: set the best query suggestion qs_i to be empty.
5: select a list of search terms ST_i.
6: apply query search on ST_i, compare every query suggestion candidate with qs_i and update qs_i greedily.
7: insert qs_i into the query suggestion set QS
8: **end for**
9: **return** QS

phase evaluates all terms in all reference documents and returns the top scoring ones. Query search is applied on search terms produced by the search term selection phase. The one that contributes the most to MCC (breaking ties by the contributions to MEC) is added to the final query suggestion set.

In the search term selection phase, a modified coverage score function is used, as shown in Equation 3.2.

$$CoverageScore(t, S_{k-1}) = OC(t, D) + EOC(t, S_{k-1}) \tag{3.2}$$

Here k is the current round, S_{k-1} is the set of final query suggestions selected from the previous $k-1$ rounds, and D is the set of all reference documents for Q_0. The OC score has been introduced in Equation 3.1. The EOC (Extra Overall Cover) score of a term equals the number of uncovered reference documents that the term covers when appended to Q_0. Different from QSQS and IQSQS, GQSQS determines one query suggestion after one round. Therefore, when we process round k ($1 \leq k \leq 10$), $k-1$ query suggestions have been generated and a portion of reference documents have been covered by these query suggestions. We strive to cover as many reference documents as possible, the EOC score is brought in to serve this purpose.

An experiment to evaluate GQSQS was carried out. The query data, the experimental setting, and the experiment method are the same as those in previous experiments. The experiment result is described in the next section.

3.8 Comparison of QSQS, IQSQS* and GQSQS

The average MCC and MEC scores for QSQS, IQSQS* and GQSQS are reported in Table 3.11. From Table 3.11, IQSQS* and GQSQS are clearly superior to QSQS on both short queries and long queries. Sign tests for comparing IQSQS* with QSQS and GQSQS with QSQS were conducted, the p-values are shown in Table 3.12 and 3.13. For all the cases, it is significant that IQSQS* (or

GQSQS) performs differently from QSQS.

System	Short Query	Long Query
QSQS	MCC=54.80 MEC=6.89	MCC=42.86 MEC=5.34
IQSQS*	MCC=70.82 MEC=9.15	MCC=78.05 MEC=10.66
GQSQS	MCC=63.88 MEC=9.73	MCC=68.82 MEC=11.08

Table 3.11: The average MCC and MEC scores of each system.

	Short queries	Long queries
MCC	p-value = 3.55E-15	p-value = 2.91E-11
MEC	p-value = 7.60E-09	p-value = 1.46E-11

Table 3.12: The p-values for sign tests to compare IQSQS* and QSQS.

	Short queries	Long queries
MCC	p-value = 1.83E-06	p-value = 2.46E-10
MEC	p-value = 7.92E-09	p-value = 4.07E-09

Table 3.13: The p-values for sign tests to compare GQSQS and QSQS.

In Table 3.11, IQSQS* is better than GQSQS on the MCC score, while GQSQS works better than IQSQS* on the MEC score. Again, the differences between GQSQS and IQSQS* are tested by the sign test, and the p-values are reported in Table 3.14.

All the results From Table 3.14 are significant. Therefore, we conclude that IQSQS* performs better than GQSQS on the MCC score, and GQSQS works better than IQSQS* on the MEC score. IQSQS* selects a list of search terms for each reference document (100 reference documents) and applies query search for each list of search terms. As opposed to that, GQSQS selects a list of search terms for one round (10 rounds) and then applies query search. Therefore, IQSQS* tries far more query suggestion candidates than GQSQS does. We expect IQSQS* to perform better on both MCC and MEC. The reason why IQSQS* has a lower MEC score than GQSQS may be the coverage score functions. The coverage score function IQSQS* uses is a combination of OC and LA where LA is given a higher weight. The one GQSQS uses combines OC and EOC with the same weight. The exact reason needs a deeper investigation to determine.

In Table 3.11, we also notice that QSQS produces better query suggestions for short queries than for long queries, but IQSQS* and GQSQS perform better on long queries than on short ones. The query suggestions generated by IQSQS* and GQSQS contain Q_0 and selected search terms, whereas QSQS does not utilize Q_0 directly. We have shown that terms in Q_0 are more likely to be selected in query suggestions. Since QSQS limits the length of the lexical alias to be between 5 and 10, when there are more terms in Q_0 (i.e. a long query), there will be less chance for terms from reference documents. This may hurt the quality of query suggestions from QSQS, therefore, QSQS has lower MCC and MEC scores on long queries.

	Short queries	Long queries
MCC	p-value = 4.62E-10	p-value = 2.33E-10
MEC	p-value = 4.53E-03	p-value = 4.55E-03

Table 3.14: The p-values for sign tests to compare IQSQS* and GQSQS.

3.9 Summary

This chapter introduces two variations, IQSQS and GQSQS, on the QSQS system by Jiang et al. [19]. Both improve the performance of QSQS substantially.

In each phase of IQSQS, there are usually several options or methods available. We conduct different experiments to determine the best configuration for IQSQS. In the reference document collection phase, we find that the application of stemming hurts the system performance so we switch off stemming in IQSQS. When we pre-select search terms, using snippets works as well as using frequency, but we chose the snippet-based method because of its other advantages. After the two new search term selection methods LAW and CSW and the four query search methods AC, QoAC, BS, and QoBS are introduced, an experiment was carried out to find the best combination of methods for IQSQS. In the end, CSW with QoBS has the highest MCC and MEC score, and we refer to it as IQSQS*.

GQSQS changes the control structure of QSQS and adopts all the best options determined in IQSQS. With a modified coverage score function and the greedy strategy, GQSQS gets better MCC and MEC scores than QSQS but lower scores than IQSQS with less effort.

Chapter 4

Does Document Clustering Help?

Queries are usually ambiguous, therefore search results of the query usually mix web documents about different aspects of the query together. For example, for the query "jaguar", the web documents about "jaguar car", "jaguar cat", "jaguar mac os" etc. are all mixed together in the corresponding search results. Because of this, the user often has to look for the desired web documents among all the search results. One possible solution is to use web document clustering to organize search results [12, 13, 15, 57, 58]. A general survey of these methods is in Chapter 5. Document clustering groups documents into different clusters by comparing their similarity so that documents in one cluster are similar to each other and documents in different clusters are dis-similar to each other. Though there is a large body of research on web document clustering, we seem to be the first to apply web document clustering to query suggestion.

4.1 Introduction

We think web document clustering methods are promising for query suggestion, assuming web document clustering methods really group search results about different topics into different clusters. If query suggestions for Q_0 could be created based on each cluster, they might represent different topics of Q_0, which may help users shorten their search time and satisfy their needs more directly. For instance, if the query suggestions for the query "jaguar" are "jaguar car", "jaguar cat" etc., the user who wants the jaguar car models and issues the query "jaguar" might click the query suggestion "jaguar car" directly.

We propose the Query Suggestion method by Document Clustering (QSDC) by plugging in different document clustering methods to GQSQS and aim for two goals. First, we would like to create query suggestions that can cover most of the reference documents. Second, we would like to create a query suggestion for each cluster that can represent the content of the cluster and also cover the reference documents in this cluster.

4.2 Query Suggestion by Document Clustering (QSDC)

The QSDC system follows the structure of the GQSQS system, but inserts a *reference document clustering* phase. Algorithm 4.1 is the pseudocode for QSDC. Reference documents for Q_0 are first collected (line 2), then a document clustering method is applied to cluster them into K ($K = 10$) clusters (line 3). For each cluster, a modified coverage score function is used to evaluate all terms in the cluster and the top scoring terms are extracted as the search terms for this cluster (line 6). The *query suggestion candidate search* phase generates different query suggestion candidates by combining search terms in different ways (line 7). The query suggestion for this cluster is then found greedily (line 7). After processing all K clusters, K query suggestions are returned (line 10).

Algorithm 4.1 Query Suggestion by Document Clustering (QSDC)

Input: the original query Q_0
Output: K ($K = 10$) query suggestions

1: initialize the query suggestion set QS to be empty.
2: collect all the reference documents D ($|D| = 100$) for Q_0.
3: cluster D into K clusters.
4: **for** $i = 1$ **to** K **do**
5: set the best query suggestion qs_i for cluster i to be empty.
6: select a list of search terms ST_i for cluster i.
7: apply query search on ST_i, compare every query suggestion candidate with qs_i and update qs_i greedily.
8: insert qs_i into the query suggestion set QS
9: **end for**
10: **return** QS

The following sections will introduce the pre-processing of reference documents for document clustering, different document clustering methods, and a CC score (Cluster Cover) in the modified coverage score function.

4.2.1 Reference Document Collection

After reference documents are processed as described in the IQSQS system, there are two more steps in the QSDC system, applying stemming[1] and converting the set of reference documents into a weighted matrix.

For each reference document, its snippet is extracted and stemming is applied to it (we utilized the Porter stemmer[2]). Then the set of reference documents is converted into a weighted matrix. Each entry of the matrix corresponds to a reference document d and a term t, the value of the entry reflects the importance of t to d. Usually, the TFIDF weighting method is used. Suppose the set of reference documents is D, then for document d and term t,

$$TFIDF_{d,t} = TF_{d,t} \times IDF_t.$$

[1] We only apply stemming in the document clustering phase.
[2] The Porter stemming algorithm is available at `http://tartarus.org/~martin/PorterStemmer/`.

$TF_{d,t}$ is defined as

$$TF_{d,t} = \frac{N(d,t)}{|d|}$$

Here, $N(d,t)$ is the number of the occurrences of them t in document d, $|d|$ is the number of occurrences of all the terms in d. $IDF(t)$ is the inverse document frequency of term t,

$$IDF(t) = log\frac{|D|}{|\{d_i : t \in d_i, d_i \in D\}|}.$$

After the TFIDF weight for each entry is calculated, the final entry value is its TFIDF weight normalized with respect to all the entries of the corresponding reference document, i.e.

$$entry(d,t) = \frac{TFIDF_{d,t}}{S_d}$$

$$S_d = \sqrt{\frac{\sum_{t=1}^{|T|}(TFIDF_{d,t})^2}{|T|}}.$$

Here, T is the set of all the terms in the reference documents.

4.2.2 Reference Document Clustering

After the set of reference documents is represented by a weighted matrix, document clustering methods such as K-means, agglomerative hierarchical clustering, spectral clustering, etc. can be applied. For the rest of the thesis, we use the term *centroid* to refer to the average center of a set of documents, i.e. a centroid of a set of documents may itself be not contained in that set of documents. The QSDC system tests the following document clustering methods:

- K-Means: we utilize a library developed by Kanungo et al. [21]. Four versions of K-means, namely, Lloyd's [22], Swap [22], Hybrid [22] and EZ-Hybrid [22], are supplied in this library. The implementation of Lloyd's algorithm in the library is the typical K-means clustering method. Because Lloyd's algorithm may get stuck in local minimal solutions, Swap is proposed to perform swaps between the current centroids and a set of candidate centroids. A swap is accepted if it decreases the average distortion (the mean squared distance from each data point to its nearest centroid). EZ-Hybrid is a simple hybrid algorithm of Swap and Lloyd's. EZ-Hybrid performs one swap after several iterations of Lloyd's. Hybrid combines Swap and Llyod's in a more complex way. Hybrid performs several swaps and then several iterations of Lloyd's. In addition, Hybrid utilizes an approach similar to simulated annealing to avoid getting trapped in local minimal solutions.

- Agglomerative Hierarchical Clustering (AHC): we implemented four versions of AHC ourselves: single-linkage, complete-linkage, average-linkage and centroid. They will be introduced in the section on agglomerative hierarchical clustering.

- Spectral Clustering: we use an implementation of the normalized spectral clustering method [37] developed in the XVDM system[3].

The following sections introduce these three clustering algorithms. For K-means, we select Lloyd's algorithm to introduce because it is the typical K-means method and many variants of K-means algorithms are based on it. For AHC, we introduce all the four versions used in our experiment. There are different variants of spectral clustering, different mathematical derivations lead to different spectral clustering algorithms. We select the unnormalized spectral clustering algorithm to introduce.

K-means Clustering

Given the number of clusters K, K-means clustering partitions a set of documents into K clusters with the objective of minimizing the average squared distance of documents in a cluster from their cluster centroids as defined in Equation 4.1 [32].

$$\vec{\mu}(C_i) = \frac{1}{|C_i|} \sum_{\vec{d} \in C_i} \vec{d} \tag{4.1}$$

$\vec{\mu}$ is the centroid of a cluster C_i, $\vec{d}$ is a document represented as a vector of term weights and $|C_i|$ is the size of the cluster C_i [32]. The objective of K-means is shown in Equation 4.2 [21].

$$\arg\min_{C} \sum_{i=1}^{K} \sum_{\vec{d_j} \in C_i} \|\vec{d_j} - \vec{\mu_i}\|^2 \tag{4.2}$$

$\|\vec{d_j} - \vec{\mu_i}\|$ is the distance between $\vec{d_j}$ and $\vec{\mu_i}$, Euclidean distance and cosine distance are commonly used. There is no efficient solution to this problem [21], so a variety of heuristic algorithms is used.

One of the most common heuristic algorithms for K-means clustering is based on a simple iterative mode to find a local minimum, which is often called *Lloyd's algorithm* (the idea is from Lloyd [29]). The first step of Lloyd's algorithm is to randomly select K documents as the initial cluster centroids. Then there are two steps iteratively repeated until a stopping condition is satisfied. Step one re-assigns each document to its closest cluster centroid. Step two re-computes each cluster centroid from all the documents belonging to it. There are several conditions that can be used as termination conditions: *a*) when the number of iterations exceeds a pre-defined value; *b*) when the document re-assignment does not change between two consecutive iterations; *c*) when the cluster centroids remain the same between two consecutive iterations; *d*) when the average squared distance falls below some threshold; and so on [32]. Figure 4.1 [32] shows an example of the process of Lloyd's algorithm.

Lloyd's algorithm converges because the average squared distance decreases or remains the same in each iteration. First, re-assigning documents cannot increase it because every document is as-

[3]The XVDM system is a high dimensional visual data mining system developed by the department of Computer Science, Free University of Bozen-Bolzano, Italy. The website of the spectral clustering implementation is at `http://projects.js-development.com/spectral-clustering`.

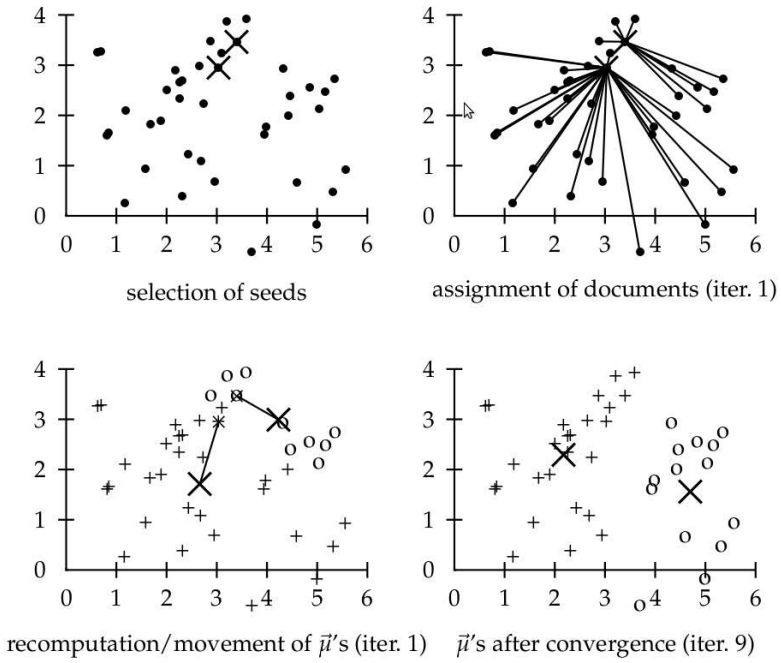

Figure 4.1: An example of the K-means algorithm (Lloyd's algorithm) when $K = 2$. Lloyd's algorithm first randomly selects two cluster centroids, then iteratively assigns the documents to the cluster centroids and re-computes the cluster centroids. After nine iterations, the cluster centroids have converged (the figure is taken from [32]).

signed to its closest centroid. Second, re-computing the centroids cannnot increase it because the new centroid of a cluster C_i minimizes $\sum_{\vec{d_j} \in C_i} \|\vec{d_j} - \vec{\mu_i}\|^2$. Since there is a finite set of partitions, the decreasing must reach a local minimum point at some stage.

Agglomerative Hierarchical Clustering

K-means clustering produces *flat* clusters whereas *hierarchical clustering* constructs a hierarchy of clusters by combining or dividing clusters iteratively. The combining version (bottom-up) is called *agglomerative hierarchical*, the dividing version (top-down) is called *divisive hierarchical*. We introduce agglomerative hierarchical clustering here because it is more popular [32, 48].

The agglomerative hierarchical clustering algorithm treats every document as a singleton cluster at the beginning, then iteratively finds two clusters that are the most similar and merges them into a new cluster. A dendrogram is constructed at the end. An example of the agglomerative hierarchical algorithm is shown in Figure 4.2[4], where $A, B, C, \ldots, G$ denote the document singleton clusters.

The agglomerative hierarchical clustering algorithm chooses two clusters that are the most similar every time. Different *linkage criteria* are used to evaluate the similarity (or distance) between two clusters. Before we introduce different linkage criteria, the *metrics* to measure the similarities between two documents need to be illustrated first. There are several metrics used to compute the

[4]The figure is from `http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/Hybrid_Hierarchical_Clustering`.

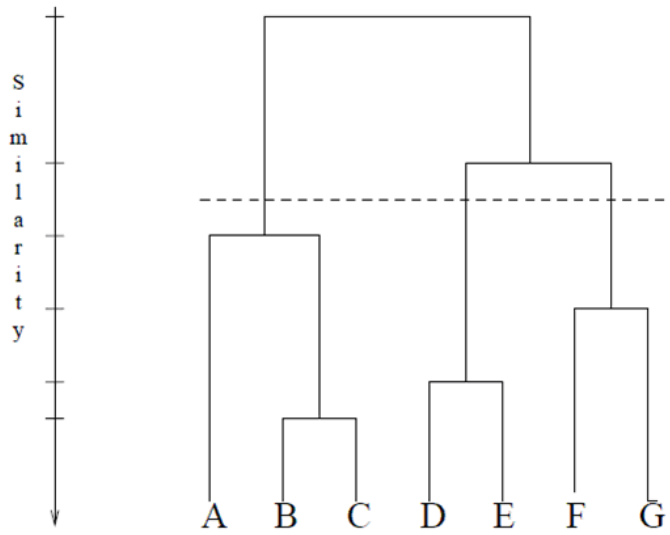

Figure 4.2: An example of the agglomerative hierarchical clustering algorithm (see footnote 4). A dendrogram is obtained by applying merging iteratively based on the similarity of the clusters. The dashed line means the cutting point of the hierarchy (described later).

similarity between documents. The *squared Euclidean distance* (Equation 4.3)

$$dist(\vec{d_i}, \vec{d_j}) = \sum_k (\vec{d_{ik}} - \vec{d_{jk}})^2 \tag{4.3}$$

and the *cosine similarity* (Equation 4.4)

$$dist(\vec{d_i}, \vec{d_j}) = cos^{-1} \frac{\vec{d_i} \cdot \vec{d_j}}{\|\vec{d_i}\|\|\vec{d_j}\|} \tag{4.4}$$

are the most popular ones.

With the metrics to evaluate the distance between two documents, there are four linkage criteria to measure the distance between two clusters. The four linkage criteria group the agglomerative hierarchical clustering methods into four types: *single-linkage* clustering, *complete-linkage* clustering, *average-linkage* clustering and *centroid* clustering. The single-linkage clustering method regards the minimum distance between a pair of documents in two clusters as the distance between the two clusters,

$$linkage(C_p, C_q) = min\{dist(d_i, d_j) : d_i \in C_p, d_j \in C_q\}$$

In contrast, the complete-linkage clustering method uses the maximum distance between documents in two clusters as the distance between the two clusters,

$$linkage(C_p, C_q) = max\{dist(d_i, d_j) : d_i \in C_p, d_j \in C_q\}$$

The average-linkage clustering method measures the distance between two clusters by the average distance of documents in the two clusters, expressed as

$$linkage(C_p, C_q) = \frac{1}{|C_p||C_q|} \sum_{d_i \in C_p} \sum_{d_j \in C_q} dist(d_i, d_j)$$

The centroid-linkage clustering method uses the distance between the centroids of the two clusters as the distance between these two clusters. Different distance metrics and linkage criteria lead to different clusterings [32].

Sometimes we do not need a dendrogram, instead, we need clusters. In this case, we just need to cut the hierarchy of the clusters. There are several cutting methods, for example, cut at a fixed similarity level like the dashed line shown in Figure 4.2, cut when the distance of two clusters are above some threshold, stop merging when there are K clusters if K clusters are needed, and so on [32]. We follow the last method in our implementation.

Spectral Clustering

Different from the K-means clustering algorithm and the agglomerative hierarchical clustering algorithm, the *spectral* clustering algorithm translates the document clustering problem into a graph partition problem, and utilizes the spectrum of the corresponding matrix to reduce the dimensionality for clustering. Based on different graph Laplacian matrices, there are different spectral clustering algorithms, such as unnormalized spectral clustering utilizing the unnormalized graph Laplacian, normalized spectral clustering utilizing the normalized graph Laplacian, etc. [52]. We select the *unnormalized spectral clustering* algorithm to describe here.

Given a set of documents $d_1, d_2, \ldots, d_n$ to cluster, if we know the similarity s_{ij} ($s_{ij} \geq 0$) between all pairs of documents d_i and d_j, a similarity graph $G = (V, E)$ can be constructed. In G, a vertex v_i represents a document d_i, v_i and v_j are connected if $s_{ij} > 0$ (or above some threshold), and the edge is weighted by s_{ij}, i.e. $\omega_{ij} = s_{ij}$. G is an undirected ($\omega_{ij} = \omega_{ji}$) and non-negative weighted graph. Suppose A is a subset of the set of vertices ($A \subset V$), $I_A = (f_1, \ldots, f_n)^T \in \mathbb{R}^n$ is an indicator vector where an entry f_i equals 1 if $v_i \in A$ and f_i equals 0 otherwise. Clustering is to partition the graph into different parts so that the vertices in the same part connect with higher weights and the vertices in different parts connect with lower weights (or don't connect) [52].

The *graph Laplacian matrices* are very important to the spectral clustering algorithm. The *unnormalized graph Laplacian* is defined by Equation 4.5,

$$L = D - W \tag{4.5}$$

D is a diagonal matrix with entries $D_{ii} = \sum_{j=1}^{n} \omega_{ij}$, and W is the weight matrix with $W = (\omega_{ij})_{i,j=1,\ldots,n}$. L is symmetric and positive semi-definite[5], and for every vector $f \in \mathbb{R}^n$

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} \omega_{ij} (f_i - f_j)^2 \tag{4.6}$$

With these, we can conclude another property related to the *unnormalized spectral clustering* algorithm (spectral clustering for unnormalized graph Laplacian) [36, 52]. Let G be an undirected

[5]A positive semidefinite matrix is a self-adjoint matrix with all of its eigenvalues nonnegative.

graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L equals the number of connected components $I_{A_1}, \ldots, I_{A_k}$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $I_{A_1}, \ldots, I_{A_k}$ of those components. This property states that, in a perfect scenario, i.e. graph G can be partitioned into k connected components, the multiplicity of the eigenvalue 0 of L equals k, and the eigenvectors of eigenvalue 0 are the connected component indicators [52].

With this in mind, the unnormalized spectral clustering algorithm is as follows, where $S \in \mathbb{R}^{n \times n}$ is the given similarity matrix and K is the number of clusters to be constructed.

1. Construct a similarity graph with W as the adjacency matrix.
2. Compute the unnormalized Laplacian L.
3. Compute the smallest K eigenvectors $v_1, \ldots, v_K$ of L.
4. Let $V \in \mathbb{R}^{n \times K}$ be the matrix containing the vectors $v_1, \ldots, v_K$ as columns.
5. For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^K$ be the vector corresponding to the i-th row of V.
6. Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^K$ into clusters $C_1, \ldots, C_K$ with the K-means clustering algorithm.

The output is a collection of clusters $A_1, \ldots, A_K$ with $A_i = \{v_j | y_j \in C_i\}$ [37, 52]. For the practical use, the result of the spectral clustering algorithm is influenced by the similarity measure mechanisms, the number of clusters, and which graph Laplacian is used.

4.2.3 Search Term Selection

A new score named Cluster Cover (CC) is brought into the coverage score function of QSDC. For a cluster c_k, the CC score of a term t is the number of reference documents in c_k that t covers when appended to Q_0. For instance, if the query $Q_0 + t$ covers 10 reference documents, among which 6 documents belong to cluster c_k, then $CC(t, c_k) = 6$. The coverage score function used in the QSDC system is given in Equation 4.7.

$$CoverageScore(t, S_{k-1}) = OC(t, D) + CC(t, c_k) + EOC(t, S_{k-1}) \tag{4.7}$$

S_{k-1} is the set of final query suggestions selected from previous clusters, and D is the set of reference documents. The OC score (Overall Cover) and the EOC score (Extra Overall Cover) have been introduced in Chapter 3. Referring to the example above, $OC(t, D) = 10$. If 8 documents out of these 10 are newly covered, $EOC(t, S_{k-1}) = 8$. The coverage score for the term t is

$$CoverageScore(t, S_{k-1}) = OC(t, D) + CC(t, c_k) + EOC(t, S_{k-1}) = 10 + 6 + 8 = 24.$$

4.2.4 Comparison of Different Document Clustering Methods

An experiment is carried out to compare the results of different document clustering methods. Following the test queries (50 short queries and 50 long queries, as Appendix A) and experiment method applied in Chapter 3, the average MCC and MEC score of each document clustering method is reported in Table 4.1 (AHC refers to the agglomerative hierarchical clustering). From the data, there appears to be no difference between these document clustering methods. Wondering whether document clustering changes anything, we calculated the MCC and MEC score for a random document clustering method as reported in the last row of Table 4.1. The random clustering method randomly assigns a document to one of the ten clusters with a uniform probability. Comparing the MCC and MEC scores of using document clustering methods and random clustering, we find that document clustering methods contribute nothing to improve MCC and MEC. Sign tests for comparing document clustering methods and the random clustering method are conducted too. Table 4.2 shows the p-values on short queries, and Table 4.3 reports the p-values on long queries.

Clustering Method	Short Queries	Long Queries
K-means (Lloyd)	$MCC = 64.04$ $MEC = 9.56$	$MCC = 68.39$ $MEC = 11.08$
K-means (Swap)	$MCC = 63.08$ $MEC = 9.53$	$MCC = 68.22$ $MEC = 10.99$
K-means (EZ-Hybrid)	$MCC = 63.42$ $MEC = 9.65$	$MCC = 68.04$ $MEC = 11.13$
K-means (Hybrid)	$MCC = 63.96$ $MEC = 9.63$	$MCC = 68.30$ $MEC = 11.08$
AHC (single-linkage)	$MCC = 63.12$ $MEC = 9.66$	$MCC = 67.22$ $MEC = 11.07$
AHC (complete-linkage)	$MCC = 63.00$ $MEC = 9.66$	$MCC = 68.00$ $MEC = 11.14$
AHC (average-linkage)	$MCC = 61.72$ $MEC = 9.47$	$MCC = 67.30$ $MEC = 10.96$
AHC (centroid)	$MCC = 63.16$ $MEC = 9.57$	$MCC = 67.65$ $MEC = 11.01$
Spectral	$MCC = 63.16$ $MEC = 9.49$	$MCC = 67.61$ $MEC = 11.02$
Random	$MCC = 63.39$ $MEC = 9.77$	$MCC = 67.91$ $MEC = 11.11$

Table 4.1: The average MCC and MEC scores of different document clustering methods in the QSDC system.

	MCC	MEC
K-means (Lloyd)	p-value = 0.19	p-value = 1.00
K-means (Swap)	p-value = 0.82	p-value = 0.66
K-means (EZ-Hybrid)	p-value = 0.21	p-value = 0.50
K-means (Hybrid)	p-value = 0.19	p-value = 0.66
AHC (single-linkage)	p-value = 0.68	p-value = 0.65
AHC (complete-linkage)	p-value = 1.00	p-value = 1.00
AHC (average-linkage)	p-value = 0.65	p-value = 0.12
AHC (centroid)	p-value = 0.50	p-value = 0.36
Spectral	p-value = 0.65	p-value = 0.68

Table 4.2: The p-values for sign tests to compare different document clustering methods with randomly assigning documents on short queries.

From Table 4.2 and 4.3, there is only one case (AHC Single-linkage on MCC) in which the p-value equals α (we set $\alpha = 0.05$). The others all fail to reject the null hypotheses, and hence show that the data is not sufficient to conclude there is a difference between those document clustering

	MCC	MEC
K-means (Lloyd)	p-value = 0.66	p-value = 0.83
K-means (Swap)	p-value = 0.83	p-value = 0.38
K-means (EZ-Hybrid)	p-value = 0.82	p-value = 1.00
K-means (Hybrid)	p-value = 1.00	p-value = 0.52
AHC (single-linkage)	p-value = 0.05	p-value = 0.38
AHC (complete-linkage)	p-value = 0.38	p-value = 0.65
AHC (average-linkage)	p-value = 0.82	p-value = 0.13
AHC (centroid)	p-value = 0.66	p-value = 0.38
Spectral	p-value = 0.50	p-value = 0.60

Table 4.3: The p-values for sign tests to compare different document clustering methods with randomly assigning documents on long queries.

methods and the random clustering method. Therefore, we think document clustering methods, at least the ones we tested, contribute nothing in our query suggestion system. However, we need to notice that we evaluate different methods by their MCC and MEC scores instead of the qualities of the query suggestions they create, which is how document clustering methods are usually evaluated.

4.3 Comparison of QSDC with QSQS, IQSQS and GQSQS

We summarize the performances of all systems in Table 4.4. Since there is no difference between different document clustering methods, we simply select K-means (Lloyd's) to represent QSDC in Table 4.4. We have compared QSQS, IQSQS* and GQSQS in Chapter 3, and the values for these systems in Table 4.4 are copied from Table 3.9. From the average MCC and MEC scores in Table 4.4, QSDC performs similarly to GQSQS. A sign test to compare QSDC and GQSQS is carried out and the p-values are shown in Table 4.5. None of them succeeds in rejecting the null hypotheses, therefore, we conclude that QSDC and GQSQS perform equally well.

System	Short Query	Long Query
QSQS	MCC=54.80 MEC=6.89	MCC=42.86 MEC=5.34
IQSQS*	MCC=70.82 MEC=9.15	MCC=78.05 MEC=10.66
GQSQS	MCC=63.88 MEC=9.73	MCC=68.82MEC=11.08
QSDC	MCC=64.04 MEC=9.56	MCC=68.39 MEC=11.08

Table 4.4: The average MCC and MEC scores of each system.

	Short queries	Long queries
MCC	p-value = 1.00	p-value = 1.00
MEC	p-value = 0.40	p-value = 0.82

Table 4.5: The p-values for sign tests to compare QSDC and GQSQS.

QSDC has the same control structure as GQSQS. However, in each round, QSDC processes one cluster of reference documents, while GQSQS processes all the uncovered reference documents. Therefore, QSDC processes fewer reference documents than GQSQS in each round and creates

equally good query suggestions.

4.4 Summary

This chapter applies document clustering to query suggestion and introduces a query suggestion method by document clustering (QSDC). In QSDC, the set of reference documents is converted into a weighted matrix after pre-processing; stemming is applied before clustering. Four versions of the K-means algorithm, four versions of the agglomerative hierarchical clustering algorithm and one spectral clustering algorithm are tested. In order to attain the purpose that the query suggestion for a cluster should cover the documents in this cluster, we also propose a Cluster Cover score (CC) that evaluates the cluster coverage of a term in the coverage score function.

An experiment was carried out to compare the performances of different document clustering methods. In terms of MCC and MEC scores, none of the document clustering methods works better than a random clustering method. Therefore, we think that the document clustering methods we tested do not help much in our query suggestion system. In addition, QSDC's performance is the same as GQSQS's.

Chapter 5

Related Work

There are many methods to create query suggestions. We classify them into three groups and introduce each one. In addition to query suggestion, we survey *web document clustering* and its use for organizing search results and helping supply query suggestions. Query suggestion evaluation methods are discussed at the end of this chapter.

5.1 Query Suggestion Methods

Many query suggestion methods extract words from a public global thesaurus that are relevant or similar to the words in the user's query and then use these words to replace or expand the words in the user's query. These methods are grouped together as methods based on *global thesauri* [5, 32, 56]. Since global thesauri may not be specific enough for one user's search intent, there is a different approach based on the documents relevant to the user's original query. These relevant documents are usually from the initial search results of the user's query, so these methods are called *local* methods [32, 56]. Similar to methods based on global thesauri, most of the local methods extract words that are relevant or similar to the words in the user's query. Recently, research [3, 31, 54] has begun to utilize search engine *logs* accumulated everyday and contributed by all the users to help query suggestion. We call these the methods based on *search logs*, and sub-classify them into three groups: *probabilistic* methods, *methods based on semantic relations* [3, 5] (these methods aim to find query suggestions that are semantically related to the user's query), and methods based on *graph models*. In addition to the methods above, some literature explores other information resources, such as the web document's snippet, the *anchor text* [23], or the user's personal information repository [7] (the personal collection of text documents, emails, cached web pages, etc).

5.1.1 Methods Based on Global Thesauri

Global means the thesauri and information are independent of the user's original query [32, 56]. The methods based on global thesauri analyze this global knowledge and extract relevant or similar terms to replace or expand the terms in the user's query. There are generally two types of thesauri in

these methods: existing and manually maintained thesauri, and automatically constructed thesauri. The problems the methods based on global thesauri need to solve are generally how to construct the thesaurus and how to extract relevant words from the thesaurus.

There are some existing thesauri for synonyms of different concepts, such as WordNet[1]. Voorhees [53] utilizes the word relations encoded in WordNet to expand the user's query. For each word *w* in the user's query, the words from the synonym set of *w* can be used to supply query suggestions.

Jing and Croft [20] consider every noun as a *concept* and build a word similarity thesaurus by linking different concepts that co-occur in a specified *window* range together. For example, suppose the window range is three sentences, then for one concept *c*, all the concepts that appear within three sentences from *c* are connected with it. A word similarity thesaurus is constructed in this way to help create query suggestions.

Qiu et al. [43] construct a term similarity thesaurus too. They use the *vector space model* to represent a term as a vector, term $t = (w_{doc_0}, w_{doc_1}, \ldots, w_{doc_n})$ where w_{doc_i} is the weight of the term in document i. A commonly used weighting method is TFIDF. With the vector model, the similarities between different terms are calculated and a term similarity thesaurus is built. Some research [53, 56] calculates the similarity with terms in the user's query to extract terms. Qiu et al. [43] measure the similarity with the user's query rather than terms to extract terms. A probabilistic model expressed in Equation 5.1 is used to measure the similarity between a term and a query.

$$Sim(q,t) = \sum_{t_i \in q} \omega_i \times Sim(t_i, t). \tag{5.1}$$

Here, $Sim(q,t)$ is the similarity between query q and term t. t_i is a term in q. $Sim(t_i,t)$ is the similarity between term t_i and t. ω_i is the weight of t_i in q.

5.1.2 Local Methods

For local query suggestion methods, usually the *relevant documents* from the initial search results of the user's original query are determined first, then relevant terms from these relevant documents are extracted to help create query suggestions. Based on how to determine the relevant documents, there are roughly three types in the literature [32]: *relevance feedback* [44, 46], *pseudo-relevance feedback* [55], and *indirect-relevance feedback* [10].

Relevance feedback [44, 46] involves an interactive procedure between the user and the search engine. After the search engine returns the search results for the user's query, the user is required to mark which web documents are relevant. Since the user indicates the relevant documents clearly, all the web documents are classified into two sets: relevant and irrelevant. Terms used for query suggestions are extracted from the relevant documents with the purpose of maximizing a function such as Equation 5.2 [32].

$$\vec{q}_{optimal} = \arg\max_{\vec{q}} [sim(\vec{q}, C_r) - sim(\vec{q}, C_{nr})] \tag{5.2}$$

[1]A lexical database for English, its website is at `http://wordnet.princeton.edu/`.

Here, $\vec{q}$ is a query vector, C_r is the set of relevant documents, C_{nr} is the set of irrelevant documents. The function $sim(\vec{q}, C_r)$ ($sim(\vec{q}, C_{nr})$) measures the similarity between $\vec{q}$ and C_r (C_{nr}). When sim is defined as cosine similarity, the optimal query vector $\vec{q}_{optimal}$ is (from [32]):

$$\vec{q}_{optimal} = \frac{1}{|C_r|} \sum_{\vec{d}_i \in C_r} \vec{d}_i - \frac{1}{|C_{nr}|} \sum_{\vec{d}_j \in C_{nr}} \vec{d}_j, \tag{5.3}$$

Here, $\vec{d}_k$ is a document vector, $|C_i|$ is the number of documents in set C_i. $\vec{q}_{optimal}$ equals the vector difference between the centroids of the relevant and irrelevant documents.

Some research [5] shows that users are usually reluctant to mark relevant documents. Therefore, *pseudo-relevance feedback* (also called blind-relevance feedback) [35, 55], in which the involvement of users is removed, are proposed. Pseudo-relevance feedback often assumes the top *T* results of the initial results as relevant. Based on these pseudo-relevant documents, the same methods of finding query suggestions for relevance feedback can be used in the pseudo-relevance feedback methods.

The third category of local query suggestion methods is called indirect-relevance feedback. Other information such as the *query session* [10], the *clickthrough data* [10], etc. are used to determine the relevant documents. Some indirect-relevance feedback methods will be introduced in the next section.

5.1.3 Methods Utilizing Search Logs

Recently, more and more work on query suggestion [3, 9, 16, 34] has been utilizing *search logs*. Search logs record the search histories of users on the search engine, i.e. the search logs record all the interactions between the user and the search engine. Query session data is extracted from search logs. A query session records the process of a user searching for a piece of specific information, and consists of one or more queries and several clicked search results. For example, suppose a user wants to watch the music videos of rock band “Coldplay”, enters “coldplay” in Google, refines the query to be “coldplay youtube”, and finally clicks on one result entitled “YouTube - The Best of Coldplay (PART ONE)”. Then the query session contains the first query “coldplay”, the second query “coldplay youtube”, and the URL of the clicked result.

Various kinds of information can be inferred from search logs. For example, similar queries from different users, the queries in one query session, the relations between the queries and the clicked documents, the last one or last few clicked documents in one query session (they are usually called *landing pages* or *landing documents*), and so on. We introduce different query suggestion methods utilizing search logs by categorizing them into probabilistic methods, methods based on semantic relations and methods based on graph models.

Probabilistic Methods

Cui et al. [10] use query logs to extract the probabilistic correlations between the terms in queries and terms in web documents. They assume the clicked documents in one query session are relevant

to the queries in the same query session. Their main idea is: if a set of documents is often clicked for similar queries, the terms in these documents are strongly related to the terms in these queries. By processing all the query logs, Cui et al. [10] calculate the probabilistic correlations between the terms in queries and the terms in documents. With the correlations, query suggestions are generated.

Cucerzan et al. [9] consider two queries relevant for each other if they return the same or similar web documents. Therefore, they link queries together if they share the same landing pages in search logs. Then, the relevant queries to the user's query are returned as the query suggestions.

Methods Based on Semantic Relations

Many query suggestion methods based on semantic relations [3, 45, 54] utilize search logs. Therefore, we introduce these methods in this section.

Cao et al. [3] proposed a context-aware query suggestion method. They first cluster the queries from query logs into different clusters, each cluster represents a concept. Then the most frequent concept sequences are calculated. A concept sequence captures the process from the user entering the first query to the user clicking the last landing page. Suppose a query session consists of queries $q_0, q_1, ..., q_m$, after representing each query with its corresponding concept, the concept sequence is $c_0, c_1, ..., c_n$. m might not equal n because different queries with the same concept are merged. After the popular concept sequences are calculated, a concept sequence suffix tree is constructed for faster processing. The concept sequence suffix tree is the query suggestion model. The user's search intent is captured by matching his/her original query to the concept suffix tree and finding out the concept the user's next query might belong to. The most popular queries in the next concept are returned as the query suggestions.

Wang et al. [54] use search logs to calculate the relations at the level of terms rather than queries. They define two types of term relation: *quasi-synonyms* and *contexture terms*. Quasi-synonyms means two words are synonymous, such as "car" and "automobile", or two words are syntactically substitutable in the similar contexts, for example, "yahoo" and "google" are quasi-synonymous when used as representatives for a search engine. Two words have a contexture relation if they are closely related in some specific context. For example, the relation between "car" and "rental" or the relation between "car" and "price". The authors build probabilistic models to calculate the relations between terms and substitute the terms in the user's query to supply query suggestions.

Sadikov et al. [45] stated that many relevant queries to the user's query could be extracted from query logs. Because only 5 to 10 of them could be suggested to the user, they proposed a query clustering method to cluster all the relevant queries with the purpose of representing distinct information needs with a limited number of query suggestions.

Graph Model

Because a query session is the path of a user's searching process, a query session can be converted into a graph. The general method is to treat a query as a graph node. The first query and all the following queries in one query session can be connected by directed edges. In addition, if the clicked documents are considered as graph nodes too, a query node q and a document node d can be connected if d is one of the clicked documents for q. Several works [2, 31, 45] have been done based on graph models.

Boldi et al. [2] generate query suggestions based on short random walks on a *query-flow* graph. A query-flow graph represents the query behaviors aggregated from search logs. In a query-flow graph, two connected queries may indicate these two queries are in the same query session, any path in the graph may be a complete search experience. Several kinds of information are attached in the query-flow graph. For example, the edge weight between two queries indicates the possibility that a real search goes from the first query to the second query, and so on.

Ma et al. [31] build two bipartite graphs, one consisting of user nodes and query nodes, the other one consisting of query nodes and document nodes. They extract a latent feature space for queries from the graphs and construct a query similarity graph based on these features. The most similar queries are supplied to the user.

Sadikov et al. [45] convert a query clustering problem into a graph clustering problem, and return the query clusters as the query suggestions.

5.2 Web Document Clustering

Much research is done on *web document clustering* with the purpose of organizing search results [12, 13, 15, 57, 58]. There appears to be no literature explicitly utilizing web document clustering for query suggestion. However, clustering web documents is a procedure of mining contents and relations between web documents. After a set of web documents is grouped into different clusters, the content of each cluster expressed by a *cluster label* could be used for query suggestion. For example, for the search results of the query "jaguar", a web document clustering method may cluster these documents into the documents related with the jaguar cat, the jaguar car, etc. The corresponding cluster labels may be "jaguar cat", "jaguar car", etc. In addition, there have been commercial search engines based on web document clustering, such as Vivisimo and Yippy[2]. Figure 5.1 shows the search results for the query "jaguar" from Yippy. The left column in Figure 5.1 shows the categories of the web documents returned for the query "jaguar".

Most of the works on web document clustering aim to produce good clusters and understandable cluster labels, none of them focuses on query suggestion. Furthermore, cluster labels that are understandable to humans are not guaranteed to be good queries for the search engine. Therefore,

[2]Vivisimo: `http://vivisimo.com/`. Yippy: `http://search.yippy.com/`.

Figure 5.1: The clustered search results of the query “jaguar” from Yippy (Jul. 15th, 2010). In addition to the search results in the center of the web page, 10 main clusters of the search results are shown in the left column. If a user needs the information about the jaguar cat, s/he may simply click the “Animal, Cat” label for related web documents

even though a system might produce perfect clusters and cluster labels, these labels may not be able to work as query suggestions directly. For example, in Figure 5.1, the label “Animal, Cat” is very likely to work terribly if we issue it for the web documents about “jaguar cat”. On the contrary, our work in Chapter 4 applies web document clustering methods to serve query suggestion. We first cluster the *reference document*s for a query, then label each cluster using an optimization equation to guarantee that the label can return many useful documents when used as a query suggestion.

There are generally two steps for web document clustering in the literature: *clustering* and *cluster labeling*. Both are introduced below.

5.2.1 Clustering

The purpose of document clustering is to group different documents into different clusters so that documents in the same cluster are similar to each other and documents in different clusters are dissimilar to each other [1]. In Chapter 4, we have introduced K-means, the agglomerative hierarchical clustering method and the spectral clustering method. In this section, we will describe several web documents clustering methods proposed recently. One famous web document clustering method, *Suffix Tree Clustering* [57], is introduced as a representative. There are also some methods first

extracting phrases from a set of web documents and then assigning each document to a phrase to accomplish clustering. We call these *phrase-centered document clustering* methods.

Suffix Tree Clustering Method

In the web context, Zamir et al. [57] proposed a document clustering method named *Suffix Tree Clustering* (STC). STC considers a document as a sentence string rather than a bag of words to maintain the information between words. There are two phases for STC: identifying the base clusters and combining the base clusters.

In the phase of identifying the base clusters, one suffix tree for all the documents is constructed. A suffix tree for a string S is a *compact trie* containing all the suffixes of the string S. A suffix tree for more than one string is a compact trie containing all the suffixes of all the strings. For example, if there are three documents reading as "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too", the suffix tree for these documents is shown in Figure 5.2 (from [57]). In Figure 5.2, the labels on the edges are phrases, the label of a node is the concatenated phrase from the root to the node. Every node represents a common phrase between different documents. For the nodes that represent the suffixes, the positions of the suffixes are also attached (in the boxes in Figure 5.2). Zamir et al. [57] say a node contains a document if the phrase of the node appears in the document. After building the suffix tree, all the nodes in the suffix tree, except those containing only one document, are the base clusters. Following the previous example, the nodes a, b, c, d, e, f in Figure 5.2 are the base clusters. Table 5.1 clearly shows the base clusters [57].

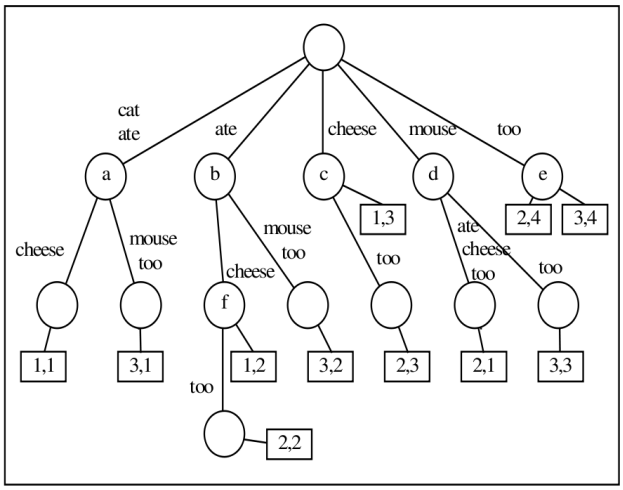

Figure 5.2: The suffix tree of strings "cat ate cheese", "mouse ate cheese too", and "cat ate mouse too" (from [57]).

In the phase of combining base clusters, Zamir et al. [57] use a fairly direct way to combine the base clusters. They combine two base clusters if these two clusters share sufficiently many documents. Specifically, for base clusters A and B, $|A|$ means the number of documents in A,

Node	Phrase	Documents
a	cat ate	1,3
b	ate	1,2,3
c	cheese	1,2
d	mouse	2,3
e	too	2,3
f	ate cheese	1,2

Table 5.1: The base clusters from Figure 5.2 (from [57]).

$|A \cap B|$ means the number of documents contained in both A and B. A and B will be combined if and only if $|A \cap B|/|A| > 0.5$ and $|A \cap B|/|B| > 0.5$. After combining the base clusters, each combined component is considered a final cluster. The documents contained by the nodes in a cluster are grouped together as a cluster. The clustering result for the previous example is shown in Figure 5.3 [57].

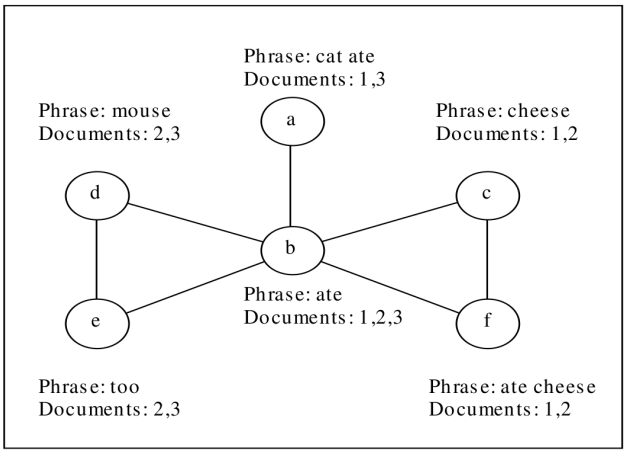

Figure 5.3: The clustering result for Figure 5.2 and Table 5.1. There is only one combined component, therefore, only one cluster is returned which contains all the documents (from [57]).

Phrase-Centered Document Clustering Methods

There are some web document clustering methods adopting the "phrase-centered" approach [6, 12, 26, 27, 59]. These methods first extract phrases that will eventually be used as cluster labels, and then assign documents to the phrases to form different clusters.

Ferragina et al. [12] proposed a snippet clustering system. They first retrieve the snippets and enrich them using an *anchor text knowledge base*[3]. If there is an anchor text about a URL in the anchor text knowledge base, this anchor text will be appended to the snippet text of the URL. After enriching the snippets, the *gapped sentences* will be selected. Ferragina et al. use a gapped sentence to distinguish from a *contiguous sentence*. For example, when using the gapped sentence, the phrases "John Fitzgerald Kennedy", "John F. Kennedy", and "John Kennedy" all equal the phrase

[3]The anchor text knowledge base was built by the authors from more than 200 millions web pages. An anchor text is usually a descriptive text attached with the hyperlink.

"John Kennedy". These phrases are different when using the contiguous sentences. Therefore, the gapped sentence is more flexible than the contiguous sentence. In order to extract the gapped sentences, they utilize the web directory Open Directory Project (ODP)[4] to calculate the ranks for all the words. $TF(w)$ evaluates the frequency of a word w in ODP rather than in a document, and is defined as

$$TF(w) = 1 + log\#(w)$$

$\#(w)$ is the frequency of w in ODP. $IDF(w)$ evaluates the generalization of w in all the categories of ODP rather than in all documents. The authors define the rank of a word w with respect to a category C_i in ODP as

$$rank(w, C_i) = TF(w) \times IDF(w) \times b(w, C_i) \times ns(C_i)$$

Here, $b(w, C_i)$ is a boosting factor for w if it appears in special positions of C_i, such as titles or descriptions. $ns(C_i)$ is a boosting factor for category C_i by considering the depth of C_i in the ODP hierarchy, a deeper category is given higher score because a deeper category is regarded more specific. The rank of a pair of words (w_h, w_k) is defined as

$$rank(w_h, w_k) = max_{C_i}\{\prod_{r=h,k} rank(w_r, C_i)\}$$

After the ranks based on ODP for all the words are calculated, all pairs of words, i.e. two-word phrases, within a fixed *window* are extracted. For example, for the sentence "Google API stable night", if the window distance is two words, then the extracted pairs of words will be "Google API", "API stable", and "stable night". After obtaining all pairs of words, the rank for every pair of words is calculated and the word pairs whose scores are below a threshold are discarded. The remaining pairs of words are incrementally merged to form longer gapped sentences using the same method, until no merge is possible or the sentence contains eight words. At the end, all the gapped sentences are the candidate labels.

Ferragina et al. then build a hierarchical clustering with all the candidate labels as the leaf cluster labels. A web document d is assigned to a cluster c if d contains the label of c. The candidate labels are the *primary* labels for the leaf clusters. In order to form the parent clusters, the *secondary* labels for the leaf clusters are extracted. If a candidate label l appears in more than 80% of the documents in cluster c, then l is a secondary label for c. The leaf clusters that share the same candidate labels in their primary or secondary labels are combined together to form a parent cluster p. The primary label for p is the common candidate label shared by its child clusters. Similarly, the secondary labels for p are candidate labels which occur in more than 80% of the documents in p. In this way, a hierarchy of at most three levels is built. The label for each cluster is the primary label.

Chen et al. [6] proposed a web document clustering method based on *word sense communities* which are groups of keywords that co-appear frequently in the search results for a query. Chen et

[4]The ODP is a hierarchical structure of more than 3,500,000 websites in more than 460,000 categories maintained by humans. The website is at `http://www.dmoz.org/`.

al. [6] convert the document clustering problem into a problem of finding the community structure on the network of extracted keywords. Specifically, all the nouns are regarded as keywords and are extracted from the web documents using Minipar[5]. Each keyword is a graph node, two nodes are connected with an edge if the corresponding keywords co-appear in the same document. After constructing the graph, an existing algorithm for finding the community structure in the graph is applied. The objective function aims to form clusters with fewer edges between different clusters and more edges within the same clusters. In this way, the clustering is finished and each cluster contains several keywords. After clustering, the documents are assigned to different clusters by their TFIDF scores. An overall TFIDF score of document d for cluster c is defined in Equation 5.3.

$$TFIDF_{d,c} = \sum_{f \in c} TFIDF_{d,f} \tag{5.4}$$

Here, f is a keyword in c. With this score, document d is assigned to the cluster which has the highest TFIDF score.

5.2.2 Cluster Labeling Methods

After clustering documents, a cluster label consisting of words is needed for humans to understand the content of the cluster. Generating cluster labels is called *cluster labeling*. A cluster label is generally representative of its cluster and discriminative with other clusters. Manning et al. [32] classify cluster labeling methods into *differential cluster labeling* and *cluster internal labeling*. Differential cluster labeling methods select a word for a cluster label by comparing its distribution in this cluster to its distribution in other clusters. The representative methods are *Mutual Information*, χ^2 *Test*, etc. These methods select label words to distinguish one cluster from others. Cluster internal labeling methods extract label words for a cluster solely based on this cluster. The straightforward methods are based on frequency or the centroid of the cluster [32]. We introduce these popular cluster labeling methods and some recent works which utilize external resources such as Wikipedia[6] to label clusters. Several advanced methods which combine different basic cluster labeling methods together are described too.

Frequency-Based Methods

Frequency-based methods select the most frequent words in the cluster. The frequency can be easily calculated in the vector space model. For example, if the document is represented as a vector, each entry corresponds to a distinct term and equals the number of occurrences of the term in the document. The frequency of a term can be defined as the number of its occurrences in all the documents. An intuitive method based on frequency [32] selects the words which occur the most

[5]Minipar is a broad-coverage parser for the English language, the website is at `http://webdocs.cs.ualberta.ca/~lindek/minipar.htm`.

[6]Wikipedia: `http://www.wikipedia.org/`.

in all the documents in a cluster. Frequency-based methods are simple and straightforward, but the words for labels selected by these methods may be neither representative nor distinctive.

Mutual Information

Mutual information (MI) measures the mutual dependence of two *random variables*, i.e. MI evaluates the information of one random variable that can be inferred when given the other random variable. Equation 5.5 [8] defines the MI of two discrete random variables X and Y.

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log(\frac{p(x,y)}{p_1(x)p_2(y)}) \tag{5.5}$$

Here, $I(X;Y)$ is the mutual information of X and Y, $p(x,y)$ is the joint probability distribution function of X and Y, and $p_1(x)$ and $p_2(y)$ are the marginal probability distribution functions of X and Y respectively. With Equation 5.5, when X and Y are independent, MI of X and Y is 0, i.e. knowing one of them implies nothing about the other one. When X equals Y, MI of X and Y becomes the entropy of X (or Y).

In the context of cluster labeling, MI measures the mutual dependence of random variables U and C. U takes the value 1 if the document contains term t, 0 otherwise; C takes the value 1 if cluster c contains the document, 0 otherwise. MI evaluates the relations between whether a document contains term t or not and whether cluster c contains the document or not. The MI equation is shown in Equation 5.6 [32].

$$I(U;C) = \frac{N_{11}}{N} log \frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N} log \frac{NN_{01}}{N_{0.}N_{.1}} + \frac{N_{10}}{N} log \frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N} log \frac{NN_{00}}{N_{0.}N_{.0}} \tag{5.6}$$

Here, each N value represents the number of documents that contain term t or not (indicated by the first subscript) and belong to cluster c or not (indicated by the second subscript) at the same time. N_{11} is the number of documents which contain t and belong to c. $N_{1.}$ is the number of documents which contain t. $N_{.1}$ is the number of documents which belong to c, etc. N is the total number of the documents. The MI score in this context measures how much a term can contribute to make a document belong to a cluster.

Pearson's χ^2 Test

Pearson's χ^2 (shortened as χ^2 test) is usually used to evaluate two things: whether the observed distribution fits with an expected distribution, and whether paired observations on two variables are independent of each other. In the cluster labeling field, the χ^2 test is often used for measuring the independence of two variables. Suppose $O_{i,j}$ is the observed number of times of the two variables taking the value i and j respectively, $E_{i,j}$ is the expected number of times that these two variables take the value i and j respectively. Then the equation of $E_{i,j}$ is shown in Equation 5.7 [40].

$$E_{i,j} = \frac{\sum_{k=1}^{|c|} O_{i,k} \sum_{k=1}^{|r|} O_{k,j}}{N} \tag{5.7}$$

Here, c is the set of all the values that the first variable can take. r is the set of all the values that the second variable can take. N is the number of all the occurrences or observations. With $E_{i,j}$, the χ^2 test is calculated by Equation 5.8 [40].

$$\chi^2 = \sum_{i=1}^{|r|}\sum_{j=1}^{|c|}\frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}} \tag{5.8}$$

In the cluster labeling context, the χ^2 test measures the independence between the occurrence of a term and the occurrence of a cluster. Let the Ns denote the numbers of the documents containing term t or not (indicated by the first subscript) and belonging to cluster c or not (indicated by the second subscript) as defined above. The calculation for the expected frequency is shown in Equation 5.9 [32].

$$E_{i,j} = \frac{\sum_{k\in\{0,1\}} N_{i,k} \sum_{k\in\{0,1\}} N_{k,j}}{N} \tag{5.9}$$

Equation 5.10 [32] is the corresponding χ^2 equation.

$$\chi^2 = \frac{(N_{11} + N_{10} + N_{01} + N_{00}) \times (N_{11}N_{00} - N_{10}N_{01})^2}{(N_{11} + N_{01}) \times (N_{11} + N_{10}) \times (N_{10} + N_{00}) \times (N_{01} + N_{00})} \tag{5.10}$$

Jensen-Shannon Divergence

Jensen-Shannon Divergence (JSD) is used to measure the similarity between two probability distributions. Suppose P and Q are two probability distributions, then the JSD of P and Q is defined by Equation 5.11 [28].

$$JSD(P||Q) = \frac{1}{2}D(P||M) + \frac{1}{2}D(Q||M), \text{where } M = \frac{1}{2}(P + Q) \tag{5.11}$$

JSD is a symmetrized version of *Kullback-Leibler divergence* (Equation 5.12 [25]) which measures the extra information needed to infer P when given Q.

$$D(P||Q) = \sum_i P(i) log \frac{P(i)}{Q(i)} \tag{5.12}$$

In the cluster labeling context, Carmel et al. [4] used JSD between the term distribution in the documents belonging to a cluster and the term distribution in all the documents. In every cluster, a term is scored with its JSD contribution, the highest scored terms are selected as the label terms. In particular, the probability distribution is calculated based on the occurrences of terms. For example, assume cluster C has 10 terms and the set of all the documents D contains 100 terms (if one term occurs twice, count two). Suppose term t occurs 2 times in C and 20 times in D, i.e. term t has the same distribution in C as in D. $P_C(t) = 2/10$, $P_D(t) = 20/100$, and $M(t) = \frac{1}{2}(P_C(t) + P_D(t)) = P_C(t) = P_D(t)$. The JSD contribution of t is

$$JSD_{contribution}(t) = \frac{2}{10} log(\frac{2/10}{2/10}) + \frac{20}{100} log(\frac{20/100}{20/100}) = 0.$$

A JSD contribution of 0 is interpreted to mean that term t does not contribute anything to distinguish the documents in cluster C from all other documents in the document set. Therefore, a 0 JSD contribution term is not suitable to be a label word for cluster C.

Centroid and Title

Suppose a document is represented by a vector in which each entry equals a term weight in the document. The centroid of a set of documents is the vector averaged on all the document vectors in the set. Stein et al. [47] first calculate the centroid for the documents in a cluster, then use the terms with top weights in the centroid as the label for the cluster. Cutting et al. [11] calculate the centroid first, then find the document that is closest to the centroid, and use the title of the document as the cluster label. The benefit of using document titles is that a title is usually more understandable than a list of extracted terms. The methods utilizing the centroid and the title are efficient for selecting label terms that are important to the cluster, but they often fail to supply discriminative label terms.

Methods Based on External Resources

External resources such as ODP[7], Wikipedia, etc. can be used to help labeling clusters. A large amount of web documents are clustered and labeled manually in ODP. A document can be compared with the existing clusters in ODP and be labeled with the manual label for the most similar cluster. Wikipedia has a hierarchical structure with a label for each cluster and linking information between similar or relevant clusters.

Advanced Cluster Labeling Methods

The method by Carmel et al. [4] creates better cluster labels by combining different cluster labeling methods or modifying basic cluster labeling methods. There are also some works [6, 13, 38, 49, 50, 57] proposing their own cluster labeling methods to tackle their specific problems. We group all these methods as advanced cluster labeling methods and introduce some here.

Carmel et al. [4] combined mutual information, JSD, Wikipedia etc. to label clusters. For a cluster, they extract the important phrases (single terms and n-grams) from the documents in this cluster and also from the related documents from Wikipedia. All these extracted phrases are the label candidates. A combined scoring method is applied to select the top scored phrases which are returned as the final cluster labels for the cluster.

Specifically, the JSD score for every phrase in the cluster is calculated. The top scored phrases are selected and put into a cluster label candidate set denoted by $T(c)$, where c represents the cluster. A query q is formed from $T(c)$ and issued against Wikipedia[8]. A list of documents $D(q)$ in Wikipedia is retrieved by query q and considered as the set of relevant documents with cluster c. The titles and category labels associated with the documents in $D(q)$ are extracted as the label candidates for cluster c too, denoted as $L(c)$. The MI (Mutual Information) score and SP (Score Propagation) score for each label candidate in $T(c)$ and $L(c)$ are calculated. The MI and SP scores

[7] An open directory project at `http://www.dmoz.org/`.

[8] A search index is retrieved from the Wikipedia Dump: `http://en.wikipedia.org/wiki/Wikipedia:Database_download`.

are aggregated and all the label candidates are ranked. The top scored candidates are the final labels. Details about the MI and SP score are as follows.

MI is used to measure the sum of the pointwise mutual information between a label candidate and all the label candidates from $T(C)$, where $C = \{c_1, c_2, \ldots, c_K\}$ is the set of clusters (K is the number of clusters). In the equation for the MI score (Equation 5.13 [4]),

$$MI(l, T(C)) = \sum_{t \in T(C)} PMI(l, t|corpus) \times \omega(t) \tag{5.13}$$

l is the candidate label, $l \in L(c) \cup T(c)$. PMI is the pointwise mutual information.

$$PMI(l, t|corpus) = log(\frac{Pr(l, t|corpus)}{Pr(l|corpus) \times Pr(t|corpus)}) \tag{5.14}$$

$$Pr(x|corpus) = \frac{\#(x|corpus)}{\#(corpus)} \tag{5.15}$$

$\omega(t)$ is the relative importance of term $t \in T(C)$. $corpus$ is an external textual source[9]. $\#(x|corpus)$ denotes the number of occurrences of term x in $corpus$. $\#(corpus)$ is the number of all the terms in $corpus$.

SP measures the label candidates extracted from Wikipedia, i.e. label candidates from $L(c)$. SP gives a label candidate higher score if the Wikipedia documents associated with it rank in the top positions in $D(q)$, i.e. the Wikipedia documents returned by query q. The SP score of a label candidate is measured as the averaged weight of words in it. Equation 5.16 [4] shows the SP function,

$$SP(l|D(q)) = \frac{1}{n(l)} \sum_{w \in l} \omega(w) \tag{5.16}$$

l is a label candidate for cluster c, $l \in L(c)$. $n(l)$ is the number of distinct words in l. For example, if l is "twitter over capacity twitter", then $n(l)$ is 3. $\omega(w)$ is the weight of word w, which is accumulated by all the weights of label candidates in $L(C)$ that contain the word w.

$$\omega(w) = \sum_{l \in L(C), w \in l} \omega(l) \tag{5.17}$$

For example, if there are two label candidates including "twitter" as ω("twitter tee") $= 0.2$ and ω("twitter client") $= 0.3$ in the cluster, then ω("twitter") $= 0.2 + 0.3 = 0.5$. $\omega(l)$ is the weight of the label candidate and is measured by the scores of the Wikipedia documents associated with l.

$$\omega(l) = \sum_{d \in D(q), l \in d} \frac{score(d)}{n(d)} \tag{5.18}$$

d is the Wikipedia document associated with l. $score(d)$ is the score of document d. Several scoring mechanisms can be applied here. For example, $score(d) = rank^{-1}(d)$. $n(d)$ is the number of label candidates associated with d.

[9]The authors used Google n-grams collection to estimate the term frequency in a very large web collection.

Equation 5.19 defines the aggregated MI and SP score for each label candidate.

$$score(l|C) = \beta_1 \times MI(l|C) + \beta_2 \times SP(l|C) \tag{5.19}$$

The β_i are weights.

Geraci et al. [13] label clusters by three steps: local candidate selection, global candidate selection, and final labels.

Local candidate selection: for a cluster, initialize scores for all the words to be 0. For each word, increase its score by 3 if it appears in a document title in the cluster; increase its score by 1 if it occurs in a snippet in the cluster. The top 10 words are selected as the local candidates.

Global candidate selection: for each cluster, a modified mutual information method (Equation 5.20 [13])

$$MI'(t,c) = P(t,c)log\frac{P(t,c)}{P(t)P(c)} + P(\bar{t},\bar{c})log\frac{P(\bar{t},\bar{c})}{P(\bar{t})P(\bar{c})} \tag{5.20}$$

is applied to select 3 terms from 10 local candidates. MI' measures the mutual dependence of term t and cluster c. In Equation 5.20, $P(t)$ is the probability of a document containing t. $P(c)$ is the probability of a document belonging to c. $P(t,c)$ is the probability of a document belonging to c and containing t at the same time. $P(\bar{t})$ is the probability that a document does not contain t. $P(\bar{c})$ is the probability that a document does not belong to c. $P(\bar{t},\bar{c})$ is the probability that a document does not belong to c and does not contain t. Comparing Equation 5.20 with the standard mutual information function (Equation 5.4), the modified mutual information takes the positive correlation and removes the negative correlation, i.e. the authors are only interested in the co-occurrence or co-absence of a term and a cluster.

Final labels: all the contiguous substrings of the snippets and document titles in the cluster are extracted. All the substrings are scored and the shortest one with the highest score is the final label. A cumulative method is used to score a substring. To score a substring, a candidate word in the substring adds a higher score (its MI score), a word in the query adds a lower score, other words decrease the score. In this way, the final label contains more candidate words and few other words.

Krishna et al. [24] proposed a concept distinctiveness and document coverage method to label clusters. They first utilize a tool developed by IBM T. J. Watson Research Lab to extract adjectives and noun phrases (phrases can be a single term or multiple terms). All the phrases are added to a phrase set. In addition, the constituent words and the sub-phrases for the phrases in the set are also extracted and added to the set.

There are two important concepts for their labeling methods: document coverage and sibling node distinctiveness. A document is said to be covered by a hierarchical cluster structure if at least one of the top level clusters contains the document. Krishna et al. [24] aim to make the final hierarchical cluster structure cover more documents. Sibling node distinctiveness means that the sibling clusters at the same level should be distinctive from each other. They select phrases based on these two concepts.

If there are K clusters on the top level, a greedy method is applied to select phrases. Denote by S_{k-1} the set of phrases which have already been selected. U_{k-1} is the set of the remaining phrases. Denoting a phrase with c_j, then the selected phrase c_k is

$$c_k = \arg\max_{c_j \in U_{k-1}} g(S_{k-1}, c_j) \tag{5.21}$$

g is the objective function

$$g(S_{k-1}, c_j) = \omega_1 g_c(S_{k-1}, c_j) + \omega_2 g_d(S_{k-1}, c_j), \tag{5.22}$$

where g_c and g_d measure the document coverage and sibling node distinctiveness respectively. ω_1 and ω_2 are the weights. $g_c(S_{k-1}, c_j)$ measures the increase in the document coverage if phrase c_j is added to the existing phrases set S_{k-1}.

$$g_c(S_{k-1}, c_j) = |d(c_j)| - |d(c_j) \cap d(S_{k-1})| \tag{5.23}$$

$d(c_j)$ is the set of documents c_j covers. $d(S_{k-1})$ is the set of documents that all the selected phrases cover. $g_d(S_{k-1}, c_j)$ measures the increase of the total number of topics when phrase c_j is added to S_{k-1}

$$g_d(S_{k-1}, c_j) = |t(c_j)| - |t(c_j) \cap t(S_{k-1})|. \tag{5.24}$$

$t(c_j)$ is the set of topics introduced by c_j. The documents covered by c_j may also be covered by other phrases, these other phrases plus c_j form the set $t(c_j)$. $t(S_{k-1})$ represents the set analogously defined for the phrases in S_{k-1}. In the end, the phrases in S_{k-1} are the final cluster labels.

5.3 Evaluation Methods for Cluster Labeling

Most of the research on web document clustering focuses on clustering or cluster labeling, less effort has been spent on evaluating the quality of the cluster labels. Some research [4, 12, 51, 59] used the "descriptor ranking" evaluation methods proposed by Treeratpituk et al. [50], which include Match@N (Match in the top N results), P@N (Precision in the top N results), MRR (Mean Reciprocal Rank), and MTRR (Mean Total Reciprocal Rank). User surveys are also adopted in some research [12, 13, 41]. Different from all these methods to measure the correctness and understandability of cluster labels, our work evaluates cluster labels by considering them as query suggestions and uses the MCC and MEC scores proposed by Jiang et al. [19].

5.3.1 Descriptor Ranking Evaluation Methods

Treeratpituk et al. [50] reformalized the document cluster labeling task as a descriptor ranking problem. The corresponding evaluation task is reformalized too. Because they ran their experiment on the web directory Open Directory Project (ODP)[10], the manual labels of different categories in ODP

[10]ODP is an open content directory of Web links and is constructed and maintained by humans. The website is at `http://www.dmoz.org/`.

are used as the ground truth. In order to compare the cluster labels, they consider a cluster label to be correct if it is identical to, an inflection of, or a Wordnet synonym of the correct label. In addition, if there is more than one term in the correct label, containing at least one of them is regarded as correct. Supposing one or multiple labels for one cluster are supplied, with the best label first, the following evaluation measures are proposed.

- Match at top N results (Match@N): Match@N is a binary value indicating whether the top N suggested labels for one cluster contain any correct labels.

- Precision at top N results (P@N): P@N is calculated as shown in the following equation.

$$P@N = \frac{|S_N \cap C|}{N}$$

  For one document cluster, S_N is the set of the top N suggested labels and C is the set of the correct labels. P@N measures the percentage of correct labels in the top N suggested labels.

- Mean Reciprocal Rank (MRR): RR is the reciprocal of the rank of the first correct label in the suggested labels. If the first correct label appears as the 3rd suggested label, then RR is 1/3. If none of the suggested labels is correct, RR is 0. If the first suggested label is correct, then RR is 1. MRR is the mean of the RR values of all document clusters.

- Mean Total Reciprocal Rank (MTRR): TRR is similar to RR except that TRR considers all the correct suggested labels rather than only the first correct one. For example, if the set of the correct labels includes the label "fruit and health", the set of the suggested labels contains the label "fruit" (2nd place) and the label "health" (4th place), then the TRR score is $1/2 + 1/4 = 3/4$. MTRR is the mean of all the TRR values of all document clusters.

Carmel et al. [4] used two data collections namely 20 News Group (20NG)[11] and ODP to run their experiment. Both of these data collections have manual labels for categories. These manual labels are considered as ground truth. Given the number of the required cluster labels K, they follow the evaluation methods Match@N and MRR@N (here, $N = K$) of Treeratpituk et al. [50].

Ferragina et al. [12] implemented a web search engine based on snippet clustering. They used 77 test queries, and for each test query, a cluster hierarchy of the search results is supplied. They asked people to manually tag whether the generated cluster labels are correct and then used P@N to evaluate their cluster labeling method. Since they generate a cluster hierarchy, if a label is manually tagged as "ambiguous", they consider the label correct if the majority of its children labels are correct. They reported P@3, P@5, P@7 and P@10 in the end.

Zeng et al. [59] asked 3 people to label the clusters for 30 queries extracted from query logs as the ground truth. Then they utilized P@N ($N \in \{5, 10, 20\}$) to evaluate their system. Treeratpituk et al. [51] used labels in ODP as ground truth and MRR as the evaluation method.

[11]The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The website is at `http://people.csail.mit.edu/jrennie/20Newsgroups/`.

5.3.2 User Surveys

Descriptor ranking evaluation methods evaluate cluster labels by following several rules, for example, whether the label contains a term in the correct label. Therefore, good cluster labels might be ignored if they do not comply with those rules. User surveys solve this problem. But compared with descriptor ranking evaluation methods, a user survey is difficult to carry out and proven to be affected by subjectivity. We introduce several studies [12, 13, 41] that adopt user surveys to evaluate their cluster labels.

Geraci et al. [13] implemented a meta-search engine that groups the web snippets returned by auxiliary search engines into disjoint labeled clusters. In order to evaluate their cluster labeling algorithm, they performed a user study on 22 computer science master students, doctoral students and post-doctorates. 35 test queries are supplied to the user in a round robin way. For each test query, all the labels of the clusters are supplied and three questions are asked in the following order (quoted from [13]):

1. "Is the label syntactically well-formed?"
2. "Can you guess the content of the cluster from the label?"
3. "After inspecting the cluster, do you retrospectively consider the cluster as well described by the label?"

The first question measures the "elegance" of the cluster labels; the second one evaluates how well the label allows to predict the content of the cluster a priori; the third one measures whether the content of the cluster, in hindsight, is well represented by the label. Three answers $\{$yes, sort-of, no$\}$ are possible for each question.

Popescul et al. [41] compared 4 cluster labeling methods: the most frequent and predictive words method, χ^2 method, most frequent words method, and most predictive words method. They conducted a user survey to supply 4 cluster labels from these 4 methods respectively for one cluster, and asked three computer science PhD students to rank these 4 cluster labels with the best in the first place. After this, each method is scored by its average ranking.

Ferragina et al. [12] used P@N to evaluate cluster labels. In addition, since they implemented a whole search engine, they gave a general evaluation of their system including the quality of cluster labels by three user surveys.

1. First study: Is web clustering beneficial? They asked 45 people to use Vivisimo[12] for a test period of 20 days. 85% of them reported they got a good sense of range alternatives with the meaningful labels, and 72% reported that the ability to produce on-the-fly clusters with labels extracted from the text in response to a query is really important.

[12]Vivisimo is a commercial search engine based on clustering. `http://vivisimo.com/`.

2. Second study: Comparison of their system with other systems, such as Mooter, CIIRarchies, Highlight, Carrot2. 18 test queries were used and three users were asked to help. The authors collected the general opinions of users for each system, such as users do not like Mooter because the cluster labels are single words.

3. Third study: Comparison of their system with Vivisimo. The authors asked 20 students to issue 18 test queries on the two systems and collected their general opinions about the quality of cluster hierarchy and cluster labels.

Chapter 6

Conclusions

6.1 Summary

This thesis proposed three query suggestion systems based on the query suggestion method (QSQS) developed by Jiang et al. [19]. Our approach follows the same objective that Jiang et al. follow, namely to generate query suggestions to cover most reference documents. In addition, we use the MCC and MEC scores proposed by Jiang et al. to evaluate query suggestions.

This thesis proposed Improved Query Suggestion by Query Search (IQSQS), Greedy Query Suggestion by Query Search (GQSQS) and Query Suggestion by Document Clustering (QSDC). IQSQS follows the structure of QSQS and provides alternative implementations for various components in QSQS. Specifically, IQSQS generalizes the lexical alias search phase in QSQS to a search term selection phase and suggests the three scoring mechanisms LA, OC, and EOC to evaluate terms from the search engine side. In addition, IQSQS replaces the top-down search in QSQS by a bottom-up search method and enhances it by a beam search. An experiment was carried out to find the best configuration of IQSQS (called IQSQS*). The combination of the search term selection method CSW and the query search method QoBS generates query suggestions with highest MCC and MEC scores. Our experiment also demonstrates that terms in the user's query are important to form desired query suggestions, and also strengthens the same conclusion drawn by Jiang et al.

GQSQS modifies the control structure of QSQS and provides 10 query suggestions in 10 rounds. Instead of selecting search terms for each reference document, GQSQS extracts search terms from all the uncovered reference documents in one round with a modified coverage score function. GQSQS attains high MCC and MEC scores with much less effort compared to QSQS.

QSDC applies document clustering to query suggestion and aims to create a query suggestion for a document cluster so that the query suggestion represents the content of the cluster as a cluster label and also covers the web documents in the cluster. Within QSDC, we tested four K-means clustering methods, four agglomerative hierarchical clustering methods and one spectral clustering method. The experimental results show that the document clustering methods we tested do not help much in terms of MCC and MEC.

6.2 Limitations and Future Work

Since this thesis is based on QSQS [17], our query suggestion methods are limited by the assumptions of QSQS. One crucial assumption is on the user's query: Jiang et al. assume the user's query is able to return some relevant documents among the reference documents. Therefore, when the user forms a terrible query for which no document in the top 120 results is of interest to the user, our query suggestion method will fail.

The target web documents this thesis deals with are those ranked between 21-120. There is no guarantee that they are relevant, they are just more likely to be relevant than documents ranked beyond 120. And there may be other useful web documents out of this range that our method will miss.

Concerning the quality of our query suggestions, we mentioned in Chapter 1 that we only use MCC and MEC for evaluation. This just measures some aspects of query suggestions, we did not evaluate the understandability of query suggestions which is an important factor in practical use. Additionally, our method can not avoid that query suggestions are too similar to each other. Therefore, there may still be ways to make the query suggestions generated by our system more practical.

Another limitation is that we know almost nothing about how the search engine works. If we knew why a query returns a web document and how the order of terms in a query influences the search results, our search term selection and query search could be improved substantially.

The coverage score functions we used in different systems are important for selecting search terms. There are four scoring mechanisms, LA, OC, EOC and CC, presented in this thesis. A further study is needed to find out the best weightings for them in the coverage score function.

GQSQS achieving high MCC and MEC scores with much less effort indicates the unnecessity for IQSQS to process every reference document. How to shorten the processing time of IQSQS while maintaining similar MCC and MEC scores is worth studying.

For QSDC, a deeper investigation is needed to analyze why document clustering does not help much in terms of MCC and MEC. Query suggestions created by QSDC might represent different aspects of the user's query compared to query suggestions generated by IQSQS and GQSQS, because query suggestions created by QSDC are based on document clusters. However, we did not evaluate the query suggestions generated by QSDC in this respect. Concerning document clustering methods, there are more advanced web document clustering methods we could try. In addition, this thesis links web documents with queries by issuing queries to the search engine. If we consider web documents covered by the same query similar, we might be able to create a new web document clustering method based on this.

There are cluster labeling methods to create understandable and distinguishable cluster labels, another direction of future work could be to utilize these labeling methods in our query suggestion system.

6.3 Final Word

This thesis proposes three query suggestion systems, IQSQS, GQSQS and QSDC, to generate query suggestions which could return reference documents to the top positions for the user to see. IQSQS selects a list of search terms for each reference document, applies query search on the search terms to generate query suggestion candidates, and greedily selects the final query suggestions to maximize MCC and MEC. GQSQS modifies the control structure of IQSQS, it selects a list of search terms, applies query search, and determines a final query suggestion in each round. QSDC inserts a document clustering phase into the structure of GQSQS, and follows the same control structure of GQSQS. QSDC supplies one query suggestion for each reference document cluster. The experiment shows that all three systems improve the performance of QSQS substantially.

This thesis selects terms for query suggestions in a new way, in that terms are evaluated by their coverage scores. The coverage score reflects how well a term covers reference documents when included in a query. In addition, this thesis utilizes document clustering methods for query suggestion. Creating a query suggestion for a reference document cluster may still be a promising approach, even though our experiment provides no evidence of the usefulness of document clustering in terms of reference document coverage.

Appendix A

Query Data

The query data we tested (50 short queries and 50 long queries) in all the experiments in this thesis are randomly sampled from the AOL transaction log in 2006[1], Jiang et al. [19] tested on exactly the same query data.

The 50 short queries (containing at most 2 terms after removing stop words) are listed in Table A.1, and the 50 long queries (containing at least 3 terms after removing stop words) are listed in Table A.2.

[1]This log was downloaded from `http://gregsadetsky.com/aol-data/`.

Short Queries	Long Queries
volcanos in italy	dress up game for kid
google	50ml of corn syrup
samos	exercises for thoracic spine instability
cheat codes	african american festivals
donbest	how to get fat off the thigs of the leg
ebay	maryland zoning map
carytown va	enviroment board in preschool
zoot suite	a sex tape with jenen ackles
wild act	savannah morning news
savings institute	fleur de lis draperies
teapot swinery	kraft cream cheese dessert recipes
supreme court	gifts for army mom
teenies	baton rouge louisiana
apollo heights	marine dock lights
argentina time	caribean hilton hotel
directv apocalypse	lutheran churches fairfax va
dillon cranes	castle in the sky songs
cancer survivor	when your husband accidently kisses another girl
asl friend	funny peterbilt t-shirts
lincoln ls	personality disorders and how to recognize them
viva owen	marine base in virginia
sensual kissig	coloring pages tullips
decorative paper	circuit court of palm beach county florida
running scared	ibm thinkpad 760c
watermelon art	coming soon nextel
pills	yorkie hair cuts
whirlygigs	racine journal times
clive owen	ged reading standards
portable generators	congressional candidate john holmes
adelphia cable	small business tax deductions
herbs	fn 5.7 reviews
myspace layouts	ptarmigan country club
oreo cookie	description of sit-up music
ask jeeves	work at home customer service jobs
drazen komarica	community college baltimore county
walgreens	lawrence county in pennsylvania
c j brown	key west campground
hugglunds hydraulic	myspace music playlists
tribes of rwanda	hotels in temple texas
nervous system	underworld 2 movie
arlington high	wholesaler for god teeth
japanese lure making	consulado peruano en georgia
sex for mothers	wake county nc records
venice hotels	mgm hotel las vegas
sister sister	eglise de la madeleine paris
eileen fisher	mexican student walkout & california & 1968
kara janx	rockingham trading post
jet corp	aol apprentice message board
refrigerator magnets	divorced over 40 moms photos
allison mack	south elgin houses for sale

Table A.1: The query data.

Appendix B

Query Suggestion Examples

We select the query suggestions generated for several queries by Google.com and different methods in Table B.1. We always generate 10 query suggestions, but Google.com may provide none or less than 10 query suggestions for different queries as shown in Table B.1.

In Table B.1, we notice that some words in Q_0 occur twice in the query suggestion. We take the query suggestion "herbs herbs" in the second part of Table B.1 as an example. The first "herbs" is included as Q_0 and the second "herbs" is a search term from the reference documents. Since we consider Q_0 and the reference documents as two different information sources, we do not eliminate words from the reference documents if they have already appeared in Q_0. The reason why "herbs herbs" is selected is because it covers more reference documents than other query suggestion candidates under our evaluation method, though this might seem counterintuitive. We do not really know how the search engine works. However, the search results of "herbs" and "herbs herbs" are clearly not the same. For many top results of "herbs herbs", the term "herbs" appears twice in important positions of the corresponding web documents, like the title, in many cases. For instance, the titles of some top results for "herbs herbs" from Google.com (Nov. 24th 2010) are "Herbs To Herbs", "Herbs Herbals herb and herbal remedies - HerbsHerbals.com", "Herb's Herbs & Such", "Medicinal herbs - Affordable herbs", etc. These are not among the top results for the query "herbs".

As we discussed in the thesis (Section 1.1), our query suggestion method is based on the behaviour of the search engine. Hence it is possible that our query suggestions are difficult to understand for humans. Also, we mentioned that there is still some processing needed to turn our query suggestions into practically usable query suggestions.

Google	QSQS	IQSQS	GQSQS	QSDC
major volcanoes in italy	italy volcanoes	volcanos in italy studies volcano volcanoes	volcanos in italy volcanos	volcanos in italy giant information active
famous volcanoes in italy	volcanos worldwide	volcanos in italy italy	volcanos in italy italy org	volcanos in italy volcanos
many volcanoes italy	volcano etna italy	volcanos in italy volcano erupted volcanoes	volcanos in italy pacific volcanos growing	volcanos in italy constitute
three volcanoes italy	volcanoes italy active	volcanos in italy cams active east	volcanos in italy japan	volcanos in italy pacific volcanos worldwide
	volcano diagram photo	volcanos in italy volcano feb 7	volcanos in italy lands	volcanos in italy pacific time
	italian volcanos	volcanos in italy tv etna	volcanos in italy 3350	volcanos in italy 08
	volcanoes forces nature mount	volcanos in italy moderate eruptions	volcanos in italy brief pacific world	volcanos in italy fire volcanos description
	volcano information encyclopedia com	volcanos in italy explore eruption	volcanos in italy uploaded	volcanos in italy information definition
	online volcano information volcanoes	volcanos in italy volcanos specifically	volcanos in italy deal	volcanos in italy tv
	amazon com volcano adventure guide	volcanos in italy pacific	volcanos in italy diagram	volcanos in italy lands
MCC=5 MEC=1.5	MCC=35 MEC=3.7	MCC=60 MEC=7.2	MCC=49 MEC=6.3	MCC=50 MEC=6.6
list of herbs	herbs herbal	herbs com herbs	herbs herbs learn herb	herbs herbs learn herb
types of herbs	herbs website	herbs company herbs site	herbs herbs co	herbs com herbs website
cooking herbs	herbs com	herbs herbal provides	herbs herbs com website	herbs information
growing herbs	herbs herb gardens gardening	herbs herbs gardens	herbs information	herbs co
culinary herbs	information herbs	herbs information	herbs medical herb site	herbs herbs
pictures of herbs	herbs organic	herbs com vitamins	herbs herbs	herbs gardens
medicinal herbs	herb store herbs herbal	herbs drying seeds method	herbs gardens	herbs medicinal
herbal medicine	medicinal herbs	herbs herbs chinese herbal	herbs herbs information database	herbs herbal
	herbs home	herbs herbology 1	herbs remedies	herbs com
	site herb growing herb herbal	herbs herbs education programs	herbs seeds	herbs seeds
MCC=15 MEC=1.9	MCC=47 MEC=6.1	MCC=54 MEC=5.7	MCC=52 MEC=6.8	MCC=49 MEC=6.7
	thinkpad 760c replacement	ibm thinkpad 760c 755 760 ibm	ibm thinkpad 760c wholesale 760	ibm thinkpad 760c 370 60
	thinkpad 760c	ibm thinkpad 760c 365 760 ibm	ibm thinkpad 760c 560 365	ibm thinkpad 760c 760 755
	760c 9547	ibm thinkpad 760c 760 dont mailing	ibm thinkpad 760c car	ibm thinkpad 760c 380 365
	ibm centre thinkpad 755cv	ibm thinkpad 760c lcd 24	ibm thinkpad 760c 355	ibm thinkpad 760c fix
	memory ibm thinkpad 760c	ibm thinkpad 760c 9546 page laptop	ibm thinkpad 760c repair	ibm thinkpad 760c 340
	760c 760cd	ibm thinkpad 760c fix	ibm thinkpad 760c 29	ibm thinkpad 760c cover
	760c 9546 product	ibm thinkpad 760c 1995	ibm thinkpad 760c 370 shopping	ibm thinkpad 760c 760 06
	ibm thinkpad 760 reviews	ibm thinkpad 760c replacement 760ld	ibm thinkpad 760c shop	ibm thinkpad 760c 60
	thinkpad 760c win	ibm thinkpad 760c vista 760 ibm	ibm thinkpad 760c wholesale 755cd	ibm thinkpad 760c 365 760 29
	ibm 760c battery	ibm thinkpad 760c 760 image com	ibm thinkpad 760c 560e 755 380	ibm thinkpad 760c 340 accessories 760
MCC=0 MEC=0.0	MCC=60 MEC=8.2	MCC=80 MEC=11.0	MCC=63 MEC=11.4	MCC=65 MEC=11.4

Table B.1: The query suggestions for the queries “volcanos in italy”, “herbs” and “ibm thinkpad 760c” by Google.com and our methods.

Bibliography

[1] N. Andrews and E. Fox. Recent developments in document clustering. Technical Report TR-07-35, Computer Science, Virginia Tech, 2007.

[2] P. Boldi, F. Bonchi, C. Castillo, D. Donato, and S. Vigna. Query suggestions using query-flow graphs. In *Proceedings of the 2009 Workshop on Web Search Click Data*, pages 56–63, 2009.

[3] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 875–883, 2008.

[4] D. Carmel, H. Roitman, and N. Zwerdling. Enhancing cluster labeling using Wikipedia. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 139–146, 2009.

[5] C. Carpineto, R. Mori, G. Romano, and B. Bigi. An information-theoretic approach to automatic query expansion. *ACM Transactions on Information Systems (TOIS)*, 19:1–27, 2001.

[6] J. Chen, O. R. Zaiane, and R. Goebel. An unsupervised approach to cluster web search results based on word sense communities. In *IEEE/WIC/ACM Conferences on Web Intelligence*, pages 725–729, 2008.

[7] P. Chirita, C. Firan, and W. Nijdl. Personalized query expansion for the web. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 7–14, 2007.

[8] T. Cover and J. Thomas. *Elements of information theory*. Wiley-Interscience, 2nd edition, 2006.

[9] S. Cucerzan and R. White. Query suggestion based on user landing pages. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 875–876, 2007.

[10] H. Cui, J. Wen, J. Nie, and W. Ma. Probabilistic query expansion using query logs. In *Proceedings of the 11th International Conference on World Wide Web*, pages 325–332, 2002.

[11] D. R. Cutting, D. R. Karger, J. O. Pederson, and J. W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.

[12] P. Ferragina and A. Gulli. A personalized search engine based on web-snippet hierarchical clustering. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, pages 801–810, 2005.

[13] F. Geraci, M. Pellegrini, M. Maggini, and F. Sebastiani. Cluster generation and labeling for web snippets: A fast, accurate hierarchical solution. *Internet Mathematics*, 3:413–443, 2006.

[14] J. Guo, G. Xu, H. Li, and X. Cheng. A unified and discriminative model for query refinement. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 379–386, 2008.

[15] X. He, H. Zha, C. H. Q. Ding, and H. D. Simon. Web document clustering using hyperlink structures. *Statistics and Computing*, 17:395–416, 2007.

[16] J. Huang and E. Efthimiadis. Analyzing and evaluating query reformulation strategies in web search logs. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 77–86, 2009.

[17] S. Jiang. Searching for queries to improve document retrieval in web search. Master's thesis, University of Alberta, 2009.

[18] S. Jiang, S. Zilles, and R. Holte. Empirical analysis of the rank distribution of relevant documents in web search. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'08)*, pages 208–213, 2008.

[19] S. Jiang, S. Zilles, and R. Holte. Query suggestion by query search: a new approach to user support in web search. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'09)*, pages 679–684, 2009.

[20] Y. Jing and W. B. Croft. An association thesaurus for information retrieval. In *Proceedings of RIAO*, pages 146–160, 1994.

[21] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[22] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry: Theory and Applications*, 28:89–112, 2004.

[23] R. Kraft and J. Zien. Mining anchor text for query refinement. In *Proceedings of the 13th International Conference on World Wide Web*, pages 666–674, 2004.

[24] K. Krishna and R. Krishnapuram. A clustering algorithm for asymmetrically related data with applications to text mining. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 571–573, 2001.

[25] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[26] K. Kummamuru, R. Lotlikar, S. Roy, and K. Singal. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *Proceedings of the 13th International Conference on World Wide Web*, pages 658–665, 2004.

[27] D. J. Lawrie and W. B. Croft. Generating hierarchical summaries for web searches. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, pages 457–458, 2003.

[28] J. Lin. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[29] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transaction of Information Theory*, 28:129–137, 1982.

[30] J. B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968.

[31] H. Ma, H. Yang, I. King, and M. R. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 709–718, 2008.

[32] C. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[33] J. Martin and R. Holte. Searching for content-based addresses on the world-wide web. In *Proceedings of the 3rd ACM Conference on Digital Libraries*, pages 299–300, 1998.

[34] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 469–478, 2008.

[35] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 206–214, 1998.

[36] B. Mohar. Some applications of Laplace eigenvalues of graphs. In *Hahn, G., Sabidussi, G. (eds.) Graph Symmetry: Algebraic Methods and Applications*, pages 225–275, 2007.

[37] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference*, pages 849–856, 2001.

[38] S. Osinski and D. Weiss. A concept-driven algorithm for clustering search results. *IEEE Intelligent Systems*, 20(3):48–54, 2005.

[39] T. Phelps and R. Wilensky. Robust hyperlinks: cheap, everywhere, now. In *Proceedings of Digital Documents and Electronic Publishing (DDEP)*, pages 13–15, 2000.

[40] R. Plackett. Karl pearson and the chi-squared test. *International Statistical Review (International Statistical Institute (ISI))*, 51(1):59–72, 1983.

[41] A. Popescul and L. H. Ungar. Automatic labeling of document clusters, 2000.

[42] M. F. Porter. An algorithm for suffix stripping. *Program*, 14:130–137, 1980.

[43] Y. Qiu and H. P. Frei. Concept based query expansion. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–169, 1993.

[44] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review*, 18(2):95–145, 2003.

[45] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *Proceedings of the 19th International Conference on World Wide Web*, pages 841–850, 2010.

[46] G. Salton. *The SMART retrieval system - experiments in automatic document processing*. Prentice-Hall, Inc., 1971.

[47] B. Stein and S. M. Zu Eissen. Topic identification: framework and application. In *Proceedings of the International Conference on Knowledge Management*, pages 522–531, 2004.

[48] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, pages 109–111, 2000.

[49] Z. Syed, T. Finin, and A. Joshi. Wikipedia as an ontology for describing documents. In *Proceedings of the Second International Conference on Weblogs and Social Media*, pages 136–144. AAAI Press, 2008.

[50] P. Treeratpituk and J. Callan. Automatically labeling hierarchical clusters. In *Proceedings of the 2006 International Conference on Digital Government Research*, pages 167–176, 2006.

[51] P. Treeratpituk and J. Callan. An experimental study on automatically labeling hierarchical clusters using statistical features. In *Proceedings of the 29th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 707–708, 2006.

[52] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17:395–416, 2007.

[53] E. M. Voorhees. Query expansion using lexical-semantic relations. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 61–69, 1994.

[54] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 479–488, 2008.

[55] R. White, C. Clarke, and S. Cucerzan. Comparing query logs and pseudo-relevance feedback-for web-search query refinement. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 831–832, 2007.

[56] J. Xu and W. Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4–11, 1996.

[57] O. Zamir and O. Etzioni. Web document clustering: a feasibility demonstration. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 46–54, 1998.

[58] O. Zamir, O. Etzioni, O. Madani, and R. M. Karp. Fast and intuitive clustering of web documents. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 287–290, 1997.

[59] H. Zeng, Q. He, Z. Chen, W. Ma, and J. Ma. Learning to cluster web search results. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 210–217, 2004.