Suggesting Queries using Query-Flow Graphs to find Dutch Content with Curated Tags

Master's Thesis

Dirk Jacob Guijt
Suggesting Queries using Query-Flow Graphs to find Dutch Content with Curated Tags

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

Dirk Jacob Guijt
born in Katwijk aan Zee

Web Information Systems
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
http://wis.ewi.tudelft.nl

Sanoma Media Netherlands
Capellalaan 65
2132 JL
Hoofddorp, the Netherlands
http://www.sanoma.nl
Abstract

One of the standard features of today’s major Web search engines are query suggestions, which aid the user in the formulation of their search queries. Over the years, a number of different approaches have been proposed that have commonly been evaluated in the standard Web search setting. In this thesis, we build a query suggestion pipeline based on query log data collected from a more constrained environment which, though also large-scale, differs considerably from standard Web search with respect to its users, indexing process and Web coverage. We implement a number of suggestion approaches based on query-flow and term-query graph models and investigate the extend to which we can replicate the results in the literature in this more constraint environment. In the process, we investigate the implementability and replicability of published Web-based query-log approaches and experiments. We find that it is possible to apply the query suggestion techniques to a constrained environment, but a trade-off between suggestion usefulness and query coverage is introduced when considering suggestion effectiveness.
Preface

After a little over eight years of being a computer science student, I can finally present my Master’s thesis to you. It has been a long road before I was ready to start my thesis work but I like to think that all the detours were worth the while and that they made me who I am today. My activities in mathematics and computer science study association Christiaan Huygens have helped me to structure and manage the research project of which this document is the result and to communicate and collaborate with all the parties involved in such a project. My part-time job at Tam Tam gave me the experience to work in a corporate environment and think about the practicality of approaching a problem. And finally, my time at student society DSB thought me how to socialize and network with people that have different backgrounds and interests. I learned that sharing a few beers is a good method to do this.

While taking a course in Web & Semantic Web Engineering, taught by Professor Houben, I realized that data science and processing large amounts of data was the field that I enjoyed working on. This ultimately led to my position as a graduate intern at Sanoma where I worked on this thesis. Their big data expertise and infrastructure was a perfect environment for my research. I would like to thank Jelmer Voogel and Sander Kieft for the opportunity they gave me and for having faith that I would be able to complete my assignment. Sorry that I never got you any coffee as expected from an intern, I just do not drink it myself. I also want to thank the members of the big data team, the Startpagina team and the data scientists for helping me when I needed something and for sparring with me about my research.

Furthermore I want to express a very big thank you to Claudia Hauff, who was the best supervisor I could wish for at the university. I could not have done my research properly without her guidance and feedback. I would also like to thank the other members of my thesis committee, Alessandro Bozzon and the WIS masters, and the rest of the people that were of value during my research project for giving feedback and for helping me write a better thesis.

And finally, I want to thank my parents for always supporting me while I was a student and for trusting that every decision I made was the right one, even if it meant that it would take me a little longer to graduate. Bedankt pap & mam!

Dirk Jacob Guijt
Delft, the Netherlands
November 26, 2014
Contents

Preface iii

Contents v

List of Figures vii

List of Tables ix

1 Introduction 1
1.1 Company profile ................................. 2
  1.1.1 About Sanoma ................................ 2
  1.1.2 About Startpagina ................................ 2
  1.1.3 Startpagina interface .......................... 3
1.2 Research objectives ............................... 5
1.3 Contributions .................................... 6
1.4 Thesis outline .................................... 6

2 Background 7
2.1 Query logs ........................................ 7
  2.1.1 Session splitting ................................ 8
  2.1.2 Query reformulations ............................ 9
2.2 Random walks ...................................... 10
  2.2.1 PageRank ...................................... 11
  2.2.2 Personalized PageRank .......................... 11
2.3 Query suggestion ................................... 12
  2.3.1 Query-flow graphs ............................... 13

3 Approach 15
3.1 Use case: Startpagina search ...................... 15
3.2 Selecting a query suggestion technique .............. 17
3.3 Pre-processing the query logs .................... 17
  3.3.1 Search session splitting ...................... 17
  3.3.2 Reformulation type classification ............. 19
3.4 From query-flow graphs to query suggestions ....... 20
3.4.1 Query-flow graph ........................................... 20
3.4.2 Slicing the query-flow graph ............................. 21
3.4.3 Term-query graph ........................................... 21
3.4.4 Generating suggestions .................................. 22
3.5 Training and evaluation ..................................... 23

4 Experimental Setup ........................................ 25
4.1 Data sets ......................................................... 25
4.2 Pre-processing the query logs ............................... 25
  4.2.1 Search session splitting .................................. 26
  4.2.2 Reformulation type classification ......................... 27
  4.2.3 Noise removal ............................................. 29
4.3 Implementation details ...................................... 30
  4.3.1 Query-flow graph generation ........................... 31
  4.3.2 Distributed PageRank calculation ....................... 32
  4.3.3 Real-time Monte Carlo Personalized PageRank .......... 33

5 Evaluation ..................................................... 35
5.1 Generated query-flow graphs ............................... 35
5.2 Evaluation setup ............................................. 36
  5.2.1 Evaluation form .......................................... 36
  5.2.2 Evaluation task examples ................................ 38
5.3 Evaluation results .......................................... 40
  5.3.1 Overall performance ..................................... 41
  5.3.2 Influence of the reformulation types .................... 42
  5.3.3 Influence of the term nodes ............................. 42
  5.3.4 Comparison with the CSE ............................... 42
  5.3.5 Suggestion overlap ...................................... 43
5.4 Discussion ..................................................... 43
  5.4.1 Reflection ................................................ 44

6 Conclusions .................................................. 47
6.1 Research questions ......................................... 47
6.2 Future work ................................................ 49

Bibliography .................................................... 51

A Evaluation form ............................................. 55
# List of Figures

1.1 Screenshot of the startpagina homepage. ........................................ 3  
1.2 Screenshot of a startpagina search result page. .............................. 4  
1.3 Screenshot of startpagina when a query does not return results. ....... 4  

2.1 Graphical depiction of the reformulation types [16]. .......................... 10  
2.2 Example of a query-flow graph based on two sessions. ....................... 13  

3.1 High-level overview of startpagina’s current user search flow. ............ 16  
3.2 Example of query-flow graph $G_{\text{orig}}$. ................................... 21  
3.3 Example of (a) an annotated query-flow graph equal to $G_{\text{orig}}$ and (b) the slice $G_{\text{SC}}$ taken from $G_{\text{orig}}$. .................................................. 22  
3.4 Example of a term-query graph. .................................................. 22  

4.1 Overview of the pre-processing steps from query logs to query-flow graphs. 26  
4.2 Example of a session splitting annotation task. ............................... 26  
4.3 Example of a reformulation type annotation task. ........................... 28  
4.4 Noise removal process in a session. .............................................. 30  

5.1 First part of the evaluation form where the evaluators’ capability to judge the suggestions is tested. ......................................................... 37  
5.2 Second part of the evaluation form where the evaluators judge the usefulness of the query suggestions. ................................................. 37  

A.1 Screenshot of the evaluation introduction text. ................................. 55  
A.2 Screenshot of the evaluation form. ................................................. 56
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example of a query log taken from a single date.</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>The splitted sessions of the query log example given in table 2.1.</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Examples of the four reformulation types [15, 16].</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>Description of the 23 features used in both session splitting and query reformulation type classification. The following notation has been used in this table: $q^{prefix} \sqsubseteq q$ means that $q^{prefix}$ is a prefix of $q$, $q^{url}$ means that all URL elements like http:// and .com have been stripped from $q$, and, $w(q)$ is the set of words in query $q$ where the space character was used as a boundary between words.</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview of search session splitting results achieved on two data sets.</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>Overview of reformulation type classification accuracy on data set $SP1$.</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Overview of the types of reformulations found in data set $SP1$ and in previously published works [16].</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>Statistics of the generated query-flow and term-query graphs.</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>Example of query suggestions generated for the query: egon.</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>Example of query suggestions generated for the query: speelschema.</td>
<td>39</td>
</tr>
<tr>
<td>5.4</td>
<td>Negative example of query suggestions generated for the query: vochtophoping na borstamputatie.</td>
<td>40</td>
</tr>
<tr>
<td>5.5</td>
<td>Overview of the query-flow graph models’ effectiveness. All but the CSE suggestions are generated using $SP1$ as training data.</td>
<td>41</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Web search engines have become indispensable for today’s Internet usage. As the Web grows larger and larger, it becomes more difficult to find the exact information or services that a user is looking for. When users do not have the address of the web page that might fulfill their needs, they turn to search engines to help them in finding pages that do. In order to be successful in finding the pages, users must understand how to use a search engine.

Traditionally, a user enters a query in the search engine to retrieve a list of results. If these results are not satisfactory, the user would use the new information learned from the results to formulate a new, hopefully better query that does return relevant results. Search engines can simulate or model this query behavior and generate query suggestions that help users that are less proficient in using a search engine [44, 34, 40]. These query suggestion techniques often rely on the content of the initially retrieved results and on previous behavior of users who posted similar queries in the past and clicked on similar documents in the result set. By providing the suggestions, the search engine actively supports the search process.

However, result lists might be sparse or non-existing when a search engine is limited in the search index size and in the way the documents in the index are described, e.g., only having document abstracts or tags opposed to having the full document text. This does not only make it harder for users to find the content that satisfies their information need, it also limits the resources that can be used for training query suggestion models. Resources such as query logs are available regardless of the presence of document contents and the indexing method. That makes the logs attractive to use for query suggestion in a search engine with the mentioned limitations. The logs lose value for this purpose though if users themselves are less able to formulate proper queries.

In this thesis, we research the effectiveness of the state-of-the-art query suggestion techniques that use the user generated data from such a limited environment as input. We implement the state-of-the-art and test if the behavior of the collective set of users can successfully feed a query suggestion model such that query suggestions can be generated to help them. Even though the individual users themselves are not always able to reformulate their queries to fulfill their information need.
1.1 Company profile

This research was performed in collaboration with startpagina\(^1\), a Dutch website owned and operated by Sanoma Media Netherlands\(^2\). startpagina employs a search engine on their website and they were willing to provide us with their query logs. We will introduce startpagina in this section. A more detailed description of the startpagina search environment is provided in section 3.1 as well as a description of the limitations that the search engine has and the motivation behind choosing startpagina as a use case for our research.

1.1.1 About Sanoma

startpagina is owned by Sanoma, an international media group with operations in more than 10 European countries. The Sanoma Media Netherlands group (over 2000 employees) is the largest media company in the Netherlands, based on reach, and is active in the fields of magazines, television, customer media, events, websites, mobile sites, apps and e-commerce. Some well known Sanoma magazine, internet and television brands in the Netherlands are: Libelle, Donald Duck, AutoWeek, Nu.nl, Startpagina.nl, Kieskeurig.nl, SBS6, Net 5 and Veronica.

1.1.2 About Startpagina

startpagina is a collection of links to other websites that startpagina considers to be useful for their visitors. They strive to be the starting point to the internet for their Dutch users, hence the name startpagina, which refers to the first page you visit when starting the browse the Web. The many links are displayed on their homepage or made available after a user enters search queries in the search engine.

startpagina is unique in the sense that they are a Dutch commercial website for which the content is selected by an editorial staff, opposed to the automatic Web crawling (e.g. Google, Bing, etc.). startpagina can best be compared with the Open Directory Project (DMOZ [2]), where links are also gathered, annotated and moderated by a human crowd. The difference is that startpagina is focused solely on the Dutch market, has full control over the people that select and annotate the content and last but not least, they are looking to make a profit out of their search engine through clicks and sales on the websites they refer to.

As stated before, startpagina does not only have a front-page with various links, they also provide search functionality. At this moment, a team of human annotators is responsible for gathering the links and assigning tags (keywords and keyword combinations) to the links. The links appear in the search results if the users searching on startpagina enter a query that is an exact match with one of the assigned tags. The whole process of entering links in the system and tagging them is done manually using insights of the annotator. No form of automated tagging is available at this moment.

startpagina does not provide its own query recommendations based on startpagina data. They currently implement query suggestions that are provided by another Commercial Search Engine (CSE). Whether the suggestions generated by the

\(^1\)http://www.startpagina.nl/
\(^2\)http://www.sanoma.nl/
CSE are useful or not, they often do not lead to results on startpagina itself. If the users follow such a suggestion, they are taken to a standard Web search result page with results provided by the CSE, even when startpagina does have high quality results available. This is unwanted behavior as startpagina wants to keep these users on their website.

We are not able to publish exact numbers, like usage statistics and conversion rates, in this thesis due to commercial reasons. Although this prohibits readers from grasping the scale of startpagina and the size of problem at hand, we were able to write this thesis in such a way that the reader is enabled to understand the problem, our efforts to tackle it and the impact we made with the proposed solution.

1.1.3 Startpagina interface

The startpagina website has a relatively simple layout. The homepage, shown in Figure 1.1, is a long page filled with various types of blocks. The most common type is a blocks of links, shown with a yellow background in the Figure. Each block of this type is of a certain category to which all the links in it belong. For example, the second block in the second column is of the category fashion stores and contains links to various online clothing webshops. There are also widget blocks, shown with a blue background. These blocks contain dynamic content which is based either on the user’s profile or on information retrieved from external APIs, such as weather or traffic data.

Not all of the content that startpagina has is shown on the homepage. startpagina has something called dochterpagina’s which are pages similar to the homepage that contain the contents of a specific topic. A few links to such pages are shown in the blue bar below the startpagina logo in Figure 1.1. There are many more hidden blocks and links that are not available on any page. This content, as well as the content

![Figure 1.1: Screenshot of the startpagina homepage.](image)
that is available on at least one page, can be reached through the search functionality. A user can search the website using the search form, which is a widget block at the top of the second and third column on the homepage.

As the user is typing in the search box, the query is matched with the tags in the index after each edit of the query. If a match is found, the interface instantly shows the results for that query. An example of a result page is shown in Figure 1.2. Again four columns are shown with both link blocks and widget blocks, the latter having a yellow background here. Three query suggestions are shown as well, right below the search box. If users think that the results are inadequate, they can follow up on these suggestions. Figure 1.3 shows a page where only the query suggestions are shown because no results are available for the user’s query.

![Figure 1.2: Screenshot of a startpagina search result page.](image)

![Figure 1.3: Screenshot of startpagina when a query does not return results.](image)
1.2 Research objectives

All major Web search engines employ some form of query suggestion. In this research we will attempt to replicate the state-of-the-art and investigate the effects of applying those methods in a more limited and specialized environment, which though large-scale, differs considerably from standard Web search with respect to its users (mostly native Dutch speakers), indexing process (documents are indexed by their human assigned tags) and Web coverage (a limited set of high-quality websites are included). We therefore propose the following main research question:

MRQ: Can we successfully apply the state-of-the-art query suggestion pipeline of a Web search environment to a more limited and specialized environment?

The answer to this question can only be given if we understand what success means for query suggestion in a CSEs, how our case differs from the CSEs and how this effects the query suggestion capabilities throughout the whole suggestion pipeline. We therefore propose the research questions stated below to aid in answering the main research question. For readability, we will refer to the limited and specialized search engine as an SSE.

RQ1: What are the state-of-the-art methods to generate query suggestions in a CSE?
RQ2: How does the SSE differ from a CSE?
RQ3: What are the effects on the query suggestion effectiveness when applied to a SSE?

The effectiveness will be measured in both the usefulness of the generated suggestions, rated by human evaluators, and the coverage, i.e., the number of queries for which suggestions can be generated. Since we are researching the data set provided by startpagina, we can compare the state-of-the-art against a realistic use case. That is why we propose a fourth research question:

RQ4: Can we propose a better query suggestion method for Startpagina?

Our approach to answering these questions is to first investigate the state-of-the-art models that are available and usable in our SSE (RQ1). We will then discuss the differences between the limited and specialized environment and regular Web search environment (RQ2). To be able to answer (RQ3) and (RQ4), we derive six different models from the state-of-the-art query-flow and term-query graph models [14, 15, 16, 17] and compare their effectiveness to the suggestions provided by the CSE (our use case baseline). We implement and review a four-step query suggestion pipeline based on state-of-the-art approaches in (1) search session splitting [30, 29], (2) the classification of query reformulations [16, 32], (3) the generation of query-flow graphs, and, (4) the extraction of suggestions from such graphs [14, 15, 16, 17]. To measure the suggestion usefulness, we generate suggestions for a randomly drawn set of queries and ask human evaluators to rate them.
1.3 Contributions

We successfully engineered and reviewed the complete query suggestion pipeline from gathering the data to generating the suggestions, including the training of classifiers that are necessary in the pipeline itself. We were able to generate query suggestions that were comparable to the literature in terms of usefulness, as rated by our human evaluators. We also find that the investigated techniques that improve the usefulness in the literature have a negative effect on the coverage, i.e., the percentage of queries for which suggestions can be made. Therefore, it is not always advised to use these techniques in a more limited environment.

Although the suggestions generated by the CSE have been rated with a higher usefulness by the human evaluators than our approach, we also found that CSE generated suggestions often do not lead to results on startpagina itself. This makes these suggestions unsuitable to support searching on startpagina. The models that we experimented with always return suggestions that lead to results. We therefore suggest that startpagina uses our models to generate suggestions if they are able to, and provide CSE generated suggestions otherwise.

1.4 Thesis outline

In Chapter 2, we discuss the state-of-the-art query suggestion methods and the techniques used in the state-of-the-art query suggestion pipeline. Our approach for implementing and evaluating our query suggestion pipeline is discussed in Chapter 3. This includes a more detailed description of the startpagina search environment (our use case). The experimental setup for this approach is discussed in Chapter 4. This includes the implementation and evaluation of setting up the pipeline. The results of our query suggestion experiments using this setup are discussed in Chapter 5. Finally, we answer the main research question and conclude our work in Chapter 6.
Chapter 2

Background

To goal of this chapter is to provide the reader with enough knowledge to understand the other contents of this thesis. We will discuss all the background information regarding the query suggestion pipeline. This includes the techniques used for pre-processing the query logs as well as the state-of-the-art query suggestion methods.

2.1 Query logs

A search engine usually logs the queries that are entered by its users. For a large scale search engine, the number of queries ranges in the millions or even billions of queries each month. The large data files resulting from this logging are called query logs. A typical entry in the query logs consists of a date and time, an (anonymized) user ID and the entered query. The query logs generally also contain information about the URLs that were clicked in the result set of a particular query. An example of a query log is given in table 2.1.

Table 2.1: Example of a query log taken from a single date.

<table>
<thead>
<tr>
<th>Time</th>
<th>User</th>
<th>Event</th>
<th>Query</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:53:01</td>
<td>user A</td>
<td>query</td>
<td>african elephant</td>
<td></td>
</tr>
<tr>
<td>14:53:06</td>
<td>user A</td>
<td>click</td>
<td>african elephant</td>
<td><a href="http://www.reallybiganimals.com/">http://www.reallybiganimals.com/</a></td>
</tr>
<tr>
<td>14:53:53</td>
<td>user A</td>
<td>query</td>
<td>elephant</td>
<td></td>
</tr>
<tr>
<td>14:54:34</td>
<td>user A</td>
<td>query</td>
<td>elephant tusk</td>
<td></td>
</tr>
<tr>
<td>14:54:53</td>
<td>user B</td>
<td>query</td>
<td>flowers</td>
<td></td>
</tr>
<tr>
<td>14:55:11</td>
<td>user A</td>
<td>query</td>
<td>elephant ears</td>
<td></td>
</tr>
<tr>
<td>14:55:12</td>
<td>user B</td>
<td>query</td>
<td>buy flowers</td>
<td></td>
</tr>
<tr>
<td>14:55:15</td>
<td>user B</td>
<td>click</td>
<td>buy flowers</td>
<td><a href="http://www.gloriousflowers.com/">http://www.gloriousflowers.com/</a></td>
</tr>
<tr>
<td>14:56:09</td>
<td>user B</td>
<td>query</td>
<td>flower vase</td>
<td></td>
</tr>
<tr>
<td>14:56:23</td>
<td>user A</td>
<td>query</td>
<td>car</td>
<td></td>
</tr>
<tr>
<td>14:56:29</td>
<td>user A</td>
<td>query</td>
<td>car radio</td>
<td><a href="http://www.veryloudcarradios.com/">http://www.veryloudcarradios.com/</a></td>
</tr>
<tr>
<td>14:56:53</td>
<td>user A</td>
<td>click</td>
<td>car radio</td>
<td></td>
</tr>
</tbody>
</table>

The information stored within the query logs can be used for various purposes including query suggestion. In this section, we will discuss two analysis techniques that are used in training the state-of-the-art query suggestion models that we used in our research: session splitting and query reformulation types.
2.1 Query logs

2.1.1 Session splitting

Throughout a visit to a search engine, a user can enter multiple queries serving multiple different search goals and click on many URLs in the result sets. Grouping the queries that belong to the same search goal together is often one of the necessary pre-processing steps to generate query-log based query suggestions. Such a group of queries is called a session. We derived the following formal definition of a session:

**Definition:** A session is a chronologically ordered set of queries that have a common search goal and are entered by a single user within a certain time interval.

The act of extracting a list of sessions from the query log is called session splitting. This topic has been tackled by various researchers in the past, including [28, 30, 29]. Most work relies heavily on lexicographical and temporal properties of two subsequent queries; it was shown that those low-level features achieve nearly the same effectiveness for search session splitting as more complex semantic or result-set based features [29]. Hagen et al. [29] attained a $F_1$-measure of 0.93 using only low-level features and up to 0.95 when adding more advanced techniques, including semantic analysis.

The first step in session splitting is the reordering of the chronologically ordered entries in the query log. The entries are sorted by user ID first, and then by the date & time. A session splitting algorithm must then find boundaries between two sessions by analyzing all subsequent query pairs that were entered by the same user. If we would split the queries from the example query log (Table 2.1) into sessions, we would obtain the session segmentation shown in Table 2.2. In this table, a horizontal line indicates a boundary between two sessions. The queries of user A and user B obviously cannot belong to the same session so there is a boundary in between queries of these users. Another boundary is found at the point where user A stopped searching for elephant related topics and started to enter car related queries. The result is three separate sessions.

Table 2.2: The splitted sessions of the query log example given in table 2.1.

<table>
<thead>
<tr>
<th>Time</th>
<th>User</th>
<th>Event</th>
<th>Query</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:53:01</td>
<td>user A</td>
<td>query</td>
<td>african elephant</td>
<td></td>
</tr>
<tr>
<td>14:53:06</td>
<td>user A</td>
<td>click</td>
<td>african elephant</td>
<td><a href="http://www.reallybiganimals.com/">http://www.reallybiganimals.com/</a></td>
</tr>
<tr>
<td>14:53:53</td>
<td>user A</td>
<td>query</td>
<td>elephant</td>
<td></td>
</tr>
<tr>
<td>14:54:34</td>
<td>user A</td>
<td>query</td>
<td>elephant tusk</td>
<td></td>
</tr>
<tr>
<td>14:55:11</td>
<td>user A</td>
<td>query</td>
<td>elephant ears</td>
<td></td>
</tr>
<tr>
<td>14:56:23</td>
<td>user A</td>
<td>query</td>
<td>car</td>
<td></td>
</tr>
<tr>
<td>14:56:29</td>
<td>user A</td>
<td>click</td>
<td>car radio</td>
<td><a href="http://www.veryloudcarradios.com/">http://www.veryloudcarradios.com/</a></td>
</tr>
<tr>
<td>14:56:53</td>
<td>user A</td>
<td>query</td>
<td>car radio</td>
<td></td>
</tr>
<tr>
<td>14:54:53</td>
<td>user B</td>
<td>query</td>
<td>flowers</td>
<td></td>
</tr>
<tr>
<td>14:55:12</td>
<td>user B</td>
<td>query</td>
<td>buy flowers</td>
<td></td>
</tr>
<tr>
<td>14:55:15</td>
<td>user B</td>
<td>click</td>
<td>buy flowers</td>
<td><a href="http://www.gloriousflowers.com/">http://www.gloriousflowers.com/</a></td>
</tr>
<tr>
<td>14:56:09</td>
<td>user B</td>
<td>query</td>
<td>flower vase</td>
<td></td>
</tr>
</tbody>
</table>
2.1.2 Query reformulations

Within a search session, a user might change their queries multiple times. This transformation of a query into its successor is considered a reformulation. Information on the type of query reformulations can be used to predict a user’s next type of reformulation and ultimately aid in predicting and suggesting a useful query for this user [15, 16]. It is therefore useful to learn how to classify the reformulation types in the query logs. We derive the following formal definition of a reformulation:

**Definition:** A query reformulation is a transition from a query into its direct successor in the same session.

Early work on user query behavior in web search defined various reformulation types [20, 36]. Many of them are on the term level, such as the addition, substitution, deletion, splitting, joining, abbreviation and the expansion of terms and spelling corrections. These types form the basis for the four reformulation types introduced in [43, 15, 16], which are now used in the state-of-the-art query suggestion methods. The four classes are: specialization, generalization, parallel move / equivalent rephrase and same query / error correction. An example of a query transition for each reformulation type is given in Table 2.3. Both [16] and [32] provide an extensive list and description of features, mostly lexicographical and temporal, that can be used to classify the reformulation type of two subsequent queries.

| G | generalization                        | african elephant → elephant |
| S | specialization                        | elephant → elephant tusk    |
| P | parallel move / equivalent rephrase   | elephant tusk → elephant ears|
| C | same query / error correction         | elephant → elephant         |

The four reformulation types can be depicted in a two dimensional plane as shown in Figure 2.1. When a reformulation from one query into another is a transition into a more general query, the query should be more to the top of the plane while a transition into a more specialized query should be more to the bottom. The horizontal axis, or dissimilarity axis, is a measure of how different a query is compared to its predecessor. Note that when two subsequent queries are very dissimilar, the search goal of the two queries is likely to be different and the search mission thus changed. This is exactly what is detected in session splitting (Section 2.1.1) which shows that these two tasks are very similar. This is also depicted in Figure 2.1 where the red striped lines indicate the boundaries that have to be detected in reformulation type classification and the blue striped line indicates the boundary that has to be detected in session splitting. Because these tasks are similar in nature, the same set of features can be used for creating both an automatic session splitter and a reformulation type classifier.
2.2 Random walks

Performing a random walk is the stochastic process of taking steps of a random length in a random direction. Together, the succession of steps form a random walk. The type of steps that are taken can differ depending on the application. An often used analogy to explain random walks and their use is the one of a drunken man, which has existed since the term ‘random walk’ was invented in 1905 [42]. The drunken man has the ability to walk but has no idea what he is doing or what he has just done. One could for example analyze the chances of such a person reaching his home after he leaves the bar or how many steps it would take him to do so. It is these types of analyses that are used in various scientific fields such as physics, biology and chemistry where the behavior of particles is modeled or for example in economics where the stock market can be simulated using random walks.

In the field of computer science, random walk algorithms have been researched extensively for all types of graph based models [38] and can be used in numerous applications. When random walking on a graph, the ‘walker’ is positioned at a particular node and then has the possibility to randomly step to an adjacent node. The chance of picking a particular adjacent node depends on the weight of the outgoing edge to that node and the total weight of all outgoing node, making the walk on the graph a walk on a Markov chain. Some well-known analysis analysis metrics for graphs using random walks are hitting time (the expected number of steps needed to reach a node \(v\) when starting a walk from node \(u\)\)), commute time (the expected number of steps needed to reach a node \(v\) and walk back to the starting node \(u\)) and cover time (the expected number of steps needed to visit all nodes in the graph at least once when starting a walk from \(u\)).

A broadly used application of random walking on a graph is node ranking. Especially when the Internet started to emerge, commercial search engines had a growing interest in a proper ranking algorithm for the graph model of the Web. This led to the (simultaneous) development of two of the most famous and influential algorithms: the Hyperlink-Induced Topic Search algorithm (HITS) [35] and PageRank [18]. Both
Background

2.2 Random walks

algorithms analyze the link structures of the web-pages. HITS uses the query content to find hubs and authorities, which both are web-pages, on the query topic and then applies a ranking algorithm to find the most interesting pages. Authorities contain relevant information on the query topic, while hubs are the pages that point to the various available authorities. PageRank is different as the ranking is query independent. We will discuss the PageRank algorithm in more detail.

2.2.1 PageRank

The basic idea behind PageRank is that a page that gets referred to by other important pages, must be important as well. The number of references to a page is of importance here, as well as the number of pages that the referencing page refers to. Essentially, each page passes a part of their ranking value on to all the pages they refer to.

What the PageRank value represents can be explained by the random (web) surfer model in similar fashion to the drunken man that we mentioned before. The surfer starts surfing on a random page on the Web. On each page he visits, he randomly clicks a link. This takes the surfer to the referred page and the clicking process repeats there. With a probability of \( 1 - \alpha \), the surfer gets bored and randomly jumps to any page on the Web and starts clicking links again. The PageRank value of a particular page is the chance that the surfer reaches that page in its random walk.

Equation 2.1 shows the formal notation of the PageRank value. In this equation, \( R_{in}(A) \) and \( R_{out}(B) \) are, respectively, the set of pages that refer to page \( A \) and the set of pages that page \( B \) refers to. \( N \) is the total number of pages on the web.

\[
PR(A) = \frac{1 - \alpha}{N} + \alpha \cdot \sum_{B \in R_{in}(A)} \frac{PR(B)}{|R_{out}(B)|}
\]  

Equation 2.1 shows the formal notation of the PageRank value. In this equation, \( R_{in}(A) \) and \( R_{out}(B) \) are, respectively, the set of pages that refer to page \( A \) and the set of pages that page \( B \) refers to. \( N \) is the total number of pages on the web.

The equation consists of two parts that are split by the \( + \)-symbol. The left part represents the chance that the surfer randomly jumps to node \( A \) when the surfer gets bored. The right part represents the chance that the surfer takes a random step node \( A \) from one of the nodes that have an edge to node \( A \). Together, they cover all the possibilities through which the surfer can visit node \( A \) in a random walk.

2.2.2 Personalized PageRank

The original PageRank paper [18] already mentioned that PageRank could be personalized to the user that is searching but regarded this to be future work. It did not take long before another paper was published that described this Personalized PageRank [41]. In regular PageRank, the surfer has a \( 1 - \alpha \) probability of jumping to any page on the web. When this random jump occurs, each page has a \( \frac{1}{N} \) chance of being jumped to. This explains the \( \frac{1 - \alpha}{N} \) part of equation 2.1.

If one would already know what pages are interesting to the user, one could increase the probability of jumping to these pages compared to the other, non-interesting pages. For example, if we would know that the user thinks that particular pages are interesting, we could use Personalized PageRank to find other pages that also might be interesting. Equation 2.2 shows the Personalized PageRank notation.
\[ PR(A) = \beta_A + \alpha \cdot \sum_{B \in R_{in}(A)} \frac{PR(B)}{|R_{out}(B)|} \] (2.2)

\[ \beta_A = \begin{cases} 
A \in \text{Pages}_{\text{interesting}} & : \frac{1-\alpha}{|\text{Pages}_{\text{interesting}}|} \\
A \notin \text{Pages}_{\text{interesting}} & : 0
\end{cases} \]

Again, the equation consists of two parts. The right parts remain the same as the surfer is still taking random steps in the graph. Only the left part changes as the chances of random jumping to node A now depend on whether the surfer thinks that page A is interesting and the total number of pages that are interesting to the surfer.

### 2.3 Query suggestion

Search engine users are not always capable of formulating proper queries that serve their desired search goals. Query suggestion can be employed to assist the user in formulating the right queries and ultimately help them in finding the results that satisfy their needs. Kato et al. [34] investigated when people use query suggestions and for what reasons. Their findings indicate that in relation to the query that has been entered by the user, especially rare queries or single term queries lead to the use of query suggestions. Suggestions are also used when they are unambiguous, when the suggestions are error corrections and/or when the suggestions are generalizations of the initial query. Lastly, users turn to suggestions when the clicks on the results of earlier queries do not satisfy their information need.

Suggestions for a given query can be generated from a multitude of sources: document content, taxonomies (e.g., WordNet), query logs or a combination of those. The use of query logs is particularly popular, but at the same time limited to use cases where large-scale query logs are available. Huang et al. [31] extracted suggestions based on co-occurring query terms in users’ search sessions. They found this approach to outperform a pure document content analysis-based approach (performed over the top-k retrieved documents). Baeza-Yates et al. [7] combined clicks and retrieved documents by only considering those top-k documents that were actually clicked by users; new query terms are suggested based on the analysis of the clicked documents. Graph models that combine multiple methods have also been suggested in [22].

The notion of click graphs started to appear in 2000 [13]. A click graph is a bipartite graph consisting of two groups of nodes, query nodes and document nodes (URLs). There is an undirected edge in between two nodes if the document appears in the result set of a query and was clicked as well. Generating such a graph from a query log enables the use of a range of graph theory algorithms. In particular random walks on click graphs are a popular strategy to generate query suggestions, e.g., [23, 21, 39]. The main drawback of click graphs is the large amount of data required to generate high-quality graphs; not every user query yields a click, and not all search portals are frequented by a sufficient number of users to make click graphs a viable option.
2.3.1 Query-flow graphs

The drawback of not having sufficient click data has led to the use of query-flow graphs for generating query suggestion. This modeling strategy was first proposed by Boldi et al. [14]. A query-flow graph contains nodes constructed from the unique set of queries in the query log. These nodes are connected by a directed edge if the queries succeeded one another in a single search session. The weights of the edges correspond to the likelihood of the two queries belonging to the same query chain. These weights could be based on either the chaining probabilities or the relative frequencies. Query suggestions are be generated by performing random walk ranking algorithms from the query node that the user entered. The top ranked nodes are likely to be of interest to the user and are therefore returned as query suggestions.

For calculating the chaining probabilities, machine learning methods are applied to determine the likelihood that two queries belong to the same query chain (or session). Various lexicographical and temporal features are used, similar to those used in session splitting and reformulation classification, as well as features based on session statistics such as co-occurrence, number of clicks in the session, average session length and others. Finally, a threshold $\theta$ is used to discard edges with a low chaining probability.

The weighting scheme based on relative frequencies is somewhat simpler and easier to calculate. For every query $u$, the number of occurrences $f(u)$ in the query log is counted. For every query $v$ that directly succeeded the query $u$ in any session, the number of times that this specific transition occurred $f(u,v)$ is counted. The weight of the edge $(u,v)$ is then $\frac{f(u,v)}{f(u)}$. There is thus no edge when a transition from $u$ to $v$ never occurred in a session in the query logs. Note that this weighting scheme turns the query-flow graph into a Markov chain. Boldi et al. [14] found that the weighting scheme based on relative frequencies is better suited when using query-flow graphs for query suggestion. A simple example of a query-flow graph constructed from two sessions using the relative frequency weighting scheme is presented in Figure 2.2.

![Figure 2.2: Example of a query-flow graph based on two sessions.](image)

Numerous extensions to the original query-flow graph model have been proposed, for instance, in [15, 16], different graphs are constructed by considering only a subset of edges based on reformulation types; in [10, 9] the random walk is biased towards the query intent. Altering the graph itself, by adding shortcuts for very likely paths in the graph has been proposed by Anagnostopoulos et al. [4].
2.3 Query suggestion

More recently, work has also began to extend such graphs with nodes on the term level: instead of considering only queries as atomic units (i.e., nodes in a graph), each unique term within a query is considered to be a node as well. An advantage of such a term-query graph [17] is that the model can generate suggestions for queries that have not been “seen” before, which is not possible in query-flow graphs. The transition of individual query terms has been added to the query-flow graph in Szpektor et al. [46] and Song et al. [45]. Both of these models have a semantic component as well through named-entity recognition and click-based topic extraction respectively.

Since users’ querying behaviour and the search system’s back-end are likely to change over time, [11] investigated the effects of time on the construction of query-flow graphs. They concluded that query-flow graph indeed age due to the change in user interest.
Chapter 3

Approach

In this chapter, we will present the use case startpagina which is used in our research. We will discuss how startpagina works and how it differs from regular web search engines. We then proceed to discuss the query suggestion technique we will use and the research approach we take to investigate this technique.

3.1 Use case: Startpagina search

As stated in the Chapter 1, startpagina gathers websites and links that they think are high quality and interesting to their users. They focus specifically on the Dutch market and therefore, the queries they receive in their search portal are mostly Dutch. The search portal employs human annotators who manually determine whether or not a website \(d\) (written in Dutch) should be included in its index. For indexing purposes, annotators provide a number of tags \(T_d = \{t_i^1, ..., t_i^m\}\) for each \(d\). Users can only retrieve \(d\) if their complete query matches a tag in \(T_d\). The entire tag space over all \(n\) documents presented in the index is \(T = \bigcup_{k=1}^{k=n} T_d\). Startpagina currently employs both auto-completion (offered by the employed commercial search engine) and instant-result mechanisms. Each typing action by a user results in one of the following:

- If the user’s current query is a tag found in \(T\), the correspondingly tagged results are shown (instant results provided by startpagina).
- Query suggestions are retrieved from the CSE’s suggestion API, whether the user’s current query is in \(T\) or not. The suggestions are agnostic to the tag space \(T\), i.e., they are not restricted to terms and phrases that appear in \(T\). A click on any of those suggestions leads the user to a standard Web search result page (SERP) provided by the CSE.

Figure 3.1 presents an overview of startpagina’s user search flow. The outcome on the left (leading the user to a target website) is the desired case: the user remains on startpagina until the final click. Unfortunately, the user are not always able to find relevant results by using their own queries, even when startpagina has relevant results available. If this is the case, the users then enter the outcome on the right (showing a SERP). This is the desired case only if startpagina does not have relevant results for the user.
3.1 Use case: Startpagina search

Approach

On average, startpagina has more than 1.5 million unique visitors who conduct tens of millions of searches each month. A significant fraction of searches result in the user leaving the site as their queries are not in \( T \) and thus no startpagina results are available. In this use case, our goal is to provide for each user query which is not found in \( T \), a list of query suggestions that are found in \( T \). In this manner, when a user submits a search query for which no results are available, she instead receives a number of related query suggestions, all of which will lead to results within startpagina.

startpagina is different from a regular CSEs in two ways. Firstly, startpagina has only a limited number of web-pages in its index. The startpagina editorial staff only selects those web-pages that they think are interesting to their users. Not all search missions can be completed on startpagina itself because of this. The result is that we are unable to suggest a relevant query to the user in all cases. The suggestion itself might be relevant, but there are no results to show for such a query. Secondly, startpagina manually assigns the tags to the web-pages which have to be matched perfectly for the results to be shown. This again will impact the number of queries that can be suggested as we do not want to suggest queries for which there are no results available. CSEs have an almost limitless set of web-pages in their index and almost any query will return at least some web-pages. The possibilities for query suggesting are therefore limitless as well.
3.2 Selecting a query suggestion technique

Our use case, startpagina, has limitations in the number of documents in the index and the way these documents are described in the index. This means that a large part of the queries do not lead to results and thus no documents can be clicked for those queries. In addition, the content on startpagina consists of links to other websites which means that the queries entered in the startpagina search engine are mostly of navigational nature [19]. It has been shown the navigational queries also result in less clicks than, for example, informational queries [33, 5]. For these reasons, we do not use any click-based approach in our research. We also cannot use content-based approaches because the content is not available, the index only contains the tags that have been assigned by the startpagina team.

We therefore turn to the query logs. Query-flow graphs and term-query graphs have shown to be a popular and successful approach to query suggestion generation by using the historical information stored in the query logs. We will use this technique in our research. We investigate two query-flow graph models (the original model [14] and variations of the reformulation type model [15, 16]) and one term-query graph model [17] and examine their performance to our particular use case startpagina. In order to do this, we create a pipeline consisting of four components: (1) search session splitting, (2) classification of query reformulation types, (3) generation of query-flow graphs as described in the literature, and, (4) deriving suggestions from them. The first two are necessary pre-processing steps to convert the query logs into a suitable representation.

3.3 Pre-processing the query logs

Before we can convert the query logs into suggestion models, we first have to pre-process the query logs. We will discuss the approach to setting up the pre-processing in this section. Both a session splitter and a reformulation type classifier are necessary components in the pre-processing steps. Session splitting is necessary in all models because we do not want to have paths in the query-flow graphs between query nodes that have different search missions. The reformulation type classification is needed for one specific model and is also used for noise removal in the query log.

3.3.1 Search session splitting

Given a typical query log (i.e. user-id, date/time, query and logged click if any), the first pre-processing step involves aggregating the queries submitted by a single user into search sessions. An often used heuristic is to split a search session after 30 or 60 minutes of inactivity [28].

Apart from splitting simply by time, classifiers have been trained in the past [28] to determine whether or not a session boundary exists between two subsequent queries \( q_i \) and \( q_{i+1} \) submitted by the same user. We opted for this approach as it generally yields better results for regular data. We derived twenty-three features (F1 to F23) based on \( q_i \) and \( q_{i+1} \) individually as well as their temporal and syntactic relationship, in line with previous and related works [30, 29, 16, 32]. These features are shown in Table 3.1.
3.3 Pre-processing the query logs

Approach

The following notation has been used in this table:

- $\Delta_{ms}$ means that all URL elements like http:// have been stripped from $q_i$ and $q_{i+1}$ means that all URL elements like http:// have been stripped from $q_i$ and $q_{i+1}$
- $q_j^{ps}$ is a prefix of $q_j$.

<table>
<thead>
<tr>
<th>#</th>
<th>Notation</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>$\Delta_{ms}$</td>
<td>time difference between $q_i$ and $q_{i+1}$ in milliseconds</td>
</tr>
<tr>
<td>F2</td>
<td>$</td>
<td>q_i</td>
</tr>
<tr>
<td>F3</td>
<td>$</td>
<td>q_{i+1}</td>
</tr>
<tr>
<td>F4</td>
<td>$F_3 - F_2$</td>
<td>difference in length between $q_i$ and $q_{i+1}$</td>
</tr>
<tr>
<td>F5</td>
<td>$\text{Equals}(q_i, q_{i+1})$</td>
<td>boolean equality of $q_i$ and $q_{i+1}$</td>
</tr>
<tr>
<td>F6</td>
<td>$\text{Equals}(q_i^{ps}, q_{i+1})$</td>
<td>boolean equality of $q_i^{ps}$ and $q_{i+1}$, with $</td>
</tr>
<tr>
<td>F7</td>
<td>$\text{Equals}(q_i, q_{i+1}^{ps})$</td>
<td>boolean equality of $q_i$ and $q_{i+1}^{ps}$, with $</td>
</tr>
<tr>
<td>F8</td>
<td>$\text{Equals}(q_i^{ml}, q_{i+1}^{ml})$</td>
<td>boolean equality of $q_i^{ml}$ and $q_{i+1}^{ml}$</td>
</tr>
<tr>
<td>F9</td>
<td>$</td>
<td>w(q_i)</td>
</tr>
<tr>
<td>F10</td>
<td>$</td>
<td>w(q_{i+1})</td>
</tr>
<tr>
<td>F11</td>
<td>$F_{10} - F_9$</td>
<td>difference of number of words in $q_i$ and $q_{i+1}$</td>
</tr>
<tr>
<td>F12</td>
<td>$</td>
<td>w(q_i) \cup w(q_{i+1})</td>
</tr>
<tr>
<td>F13</td>
<td>$</td>
<td>{x \in w(q_i), y \in w(q_{i+1})</td>
</tr>
<tr>
<td>F14</td>
<td>$</td>
<td>{x \in w(q_i), y \in w(q_{i+1})</td>
</tr>
<tr>
<td>F15</td>
<td>$</td>
<td>w(q_{i+1}) - w(q_i)</td>
</tr>
<tr>
<td>F16</td>
<td>$</td>
<td>w(q_i)</td>
</tr>
<tr>
<td>F17</td>
<td>$\text{Lev}(q_i, q_{i+1})$</td>
<td>Levenshtein distance between $q_i$ and $q_{i+1}$</td>
</tr>
<tr>
<td>F18</td>
<td>$\text{Lev}(q_i, q_{i+1}) / (F_2 + F_3)$</td>
<td>Levenshtein distance normalized by average length</td>
</tr>
<tr>
<td>F19</td>
<td>$\text{Lev}(q_i, q_{i+1}) / F_1$</td>
<td>Levenshtein distance normalized by time difference</td>
</tr>
<tr>
<td>F20</td>
<td>$\text{CosSim}(\theta_{word,i}, \theta_{word,i+1})$</td>
<td>cosine similarity of character 3-grams extracted from the individual words in a query</td>
</tr>
<tr>
<td>F21</td>
<td>$\text{CosSim}(\theta_{query,i}, \theta_{query,i+1})$</td>
<td>cosine similarity of the character 3-grams extracted from the query string as a whole</td>
</tr>
<tr>
<td>F22</td>
<td>$\text{CosSim}(\theta_{query,i}, \theta_{query,i+1})$</td>
<td>cosine similarity of the character 3-grams extracted from the query string as a whole</td>
</tr>
<tr>
<td>F23</td>
<td>$\text{CosSim}(\theta_{query,i}, \theta_{query,i+1})$</td>
<td>cosine similarity of the character 3-grams extracted from the query string as a whole</td>
</tr>
</tbody>
</table>
To train a classifier, we manually annotated a set of randomly drawn super-sessions (a session of a single user split by the 60 minute inactivity heuristic is one super-session) from our query log and determined for each pair of subsequent queries whether or not they belong to the same session. We employed a decision tree classifier (C5.0) in line with the reformulation type classification work [16], as both task are similar in nature. As a comparison, we also applied our implementation to a search session data set provided in [29] (we refer to as Hagen13), which is based on the AOL query log.

We evaluate the effectiveness of the classifier through 5-fold cross-validation and report: correctness (% of sessions that are correct, i.e. the start/end query are identified and no split is introduced in-between), precision and recall in terms of correctly identified boundaries, and, $F_{1.5}$-measure. The latter is motivated by the fact that false positives are less harmful than false negatives which merge different search sessions into one, leading to a degradation of the query-flow graph.

- **Correct**: the percentage of sessions that are completely correct, i.e. the start and end query are correctly identified and no split is introduced in-between;
- **Precision**: the percentage of boundaries that were correctly identified from all the boundaries that were classified as boundaries;
- **Recall**: the percentage of boundaries that were correctly identified of all the boundaries that should have been identified; and,
- **F-Measure**: with $\beta = 1.5$, as we consider false positives to be less harmful than false negatives (in the later case different search sessions are merged into one, leading to a degradation of the query-flow graph).

### 3.3.2 Reformulation type classification

One of the query-flow graph models we investigate is based on the intuition that different graphs should be built for different query reformulation types [16] as not all reformulations are equally useful for the generation of query suggestions. Thus, as a second component we implemented a classifier to determine the reformulation type of a pair of queries issued subsequently within a single search session. We distinguish between the following five types [16]: specialisation ($S$), generalisation ($G$), same query/error correction ($C$), parallel move/equivalent rephrase ($P$), and, typing ($T$).

All but the last type ($T$) can also be found in [16]. We decided to add the typing class as our search portal already implements the instant result feature, i.e., it shows results (if there are any) while the user is typing, instead of waiting for a click on the search button. The logging mechanism logs a search string as an individual query, if the user stopped typing for more than one second. Some users type slowly and others may be focusing on the instant results shown, leading to noise in the query logs.

We randomly selected query pairs from the identified search sessions using techniques described in Section 3.3.1 and manually annotated them according to their reformulation type. Since session splitting and reformulation classification are similar tasks, we rely on the same set of twenty-three features and the same type of classifier (C5.0) for both tasks. We experimented both with a single $n$-class classifier (where $n$ is the number of reformulation types) and a classifier cascade. The classifier cascade
consists of \( n \) binary classifiers as well as the \( n \)-class classifier. The \( n \) binary classifiers are ordered according to their accuracy. If a query pair is positively classified by a classifier in the cascade, the remaining classifiers are skipped for that pair [16]. In this manner, easy cases are handled by using only the first, more accurate steps, while difficult or ambiguous cases have to run through the entire cascade. The performance of a classifier is measured by calculating the percentage of correctly classified reformulation types.

### 3.4 From query-flow graphs to query suggestions

In our research, we experiment with eight models, five based on the query-flow graph and different reformulation types, one based on the term-query graph and two relying on the current system’s suggestions that are generated by a commercial search engine:

- **\( M_{\text{orig}} \):** basic query-flow graph including all reformulations types;
- **\( M_{\text{GPC}} \):** query-flow graph with generaliz., parallel move and error correction;
- **\( M_{\text{GPS}} \):** query-flow graph with generaliz, parallel move and specializ.;
- **\( M_{\text{GCS}} \):** query-flow graph with generaliz, error correction and specializ.;
- **\( M_{\text{PCS}} \):** query-flow graph with parallel move, error correction and specializ.;
- **\( M_{\text{TGM}} \):** the term-query graph model including all reformulations types;
- **CSE suggestions overall:** we retrieve the top-5 suggestions for each test query from the system’s currently employed CSE, independent of whether or not the suggestions are in \( T \) (i.e. leading to results in \texttt{startpagina}). We can only extract the first five suggestions due to limitations imposed by the API of the employed CSE;
- **CSE suggestions \( \in T \):** we use the suggestions from the previous model (CSE suggestions overall) as a starting point and filter out all those that do not appear in tag space \( T \) (i.e. those that do not lead to results within \texttt{startpagina}). Note that it is quite possible that none of the top-5 suggestions are in \( T \) and that we cannot extract other suggestions beyond the five suggestion limit.

In this section we will discuss the various query-flow graph and term-query graph models that we investigate, as well as the method we use to generate suggestions from these models.

#### 3.4.1 Query-flow graph

Boldi et al.’s query-flow graph [14, 15, 16] is a weighted, directed and annotated graph \( G_{\text{orig}} = (V, E, W, T) \). The set of nodes \( V \) is defined as \( V = V_{\text{query}} \cup \{ t \} \) with \( V_{\text{query}} \) being the set of unique queries in a query log and \( t \) being a special node denoting the end of a search session. The set of edges \( E \subseteq V \times V \) is a subset of all possible edges. An edge appears between nodes \( u \) and \( v \) if \( u \) was reformulated into \( v \) in at least one session in the query log. The edge \((u, t)\) appears when \( u \) is the last query in a session. Edges are associated with weights, i.e. \( w(u, v) = (0..1) \in W \). The weight is dependent on the frequency of the transition in the query log. Let \( \text{OUT}(u) \) be the set of nodes that
have an incoming edge from \( u \). For every node it holds that:

\[
\sum_{v \in \text{OUT}(u)} w(u, v) = 1,
\]

which makes the graph essentially a Markov chain. Then, \( w(u, v) = \frac{r(u, v)}{\sum_{u \in \text{OUT}(v)} r(u, v)} \), where \( r(u, v) \) is the number of times that the transition from \( u \) to \( v \) occurred in the log.

Finally, \( T \) is the set of annotations for each edge in \( E \): each edge is annotated with the type of reformulation occurring between \( u \) and \( v \). Edges directed to end node \( t \) are marked with special type \( X \). An example of a query-flow graph is shown in Figure 3.2. Note that the original model does not have any reformulation type annotations.

![Figure 3.2: Example of query-flow graph \( G_{\text{orig}} \).](image)

### 3.4.2 Slicing the query-flow graph

As already mentioned, prior work [15, 16] has investigated the use of a limited set of reformulation types to generate a more refined version of the query-flow graph. To create a slice of \( G_{\text{orig}} \), we only retain those edges that are annotated with our chosen reformulation types; as an example, \( G_{\text{SC}} \) is the slice of \( G_{\text{orig}} \) which only contains reformulations of types specialization and error correction. The edge weights are normalized so that \( \sum_{v \in \text{OUT}(u)} w(u, v) = 1 \). The special reformulation type \( X \) is assumed to be present in every slice.

Figure 3.3a shows the query-flow graph that is annotated with reformulation types. At this moment, the graph is equal to graph \( G_{\text{orig}} \), which is shown in Figure 3.2, and will generate the exact same query suggestions. Figure 3.3b shows the slice \( G_{\text{SC}} \) of \( G_{\text{orig}} \). The edges that have the classifications generalization, parallel move or typing are omitted and the weights of the remaining edges are adjusted accordingly. Note that nodes are never removed when creating a slice of \( G_{\text{orig}} \).

### 3.4.3 Term-query graph

The third model we consider is the term-query graph-based model [17], which again is an extension of \( G_{\text{orig}} \). Typically, queries consist of a number of terms. In this model, we enlarge the graph to also add the set of unique terms of the query log as nodes, that is, \( V = V_{\text{query}} \cup V_{\text{term}} \cup \{t\} \). The edge set \( E \) is also enlarged: there is an edge between a node \( u \in V_{\text{term}} \) and \( v \in V_{\text{query}} \) if \( v \) contains the term \( u \). Note that no edges are directed towards term nodes and no direct edges exist between two term nodes. For a node \( u \in V_{\text{term}} \) and \( v \in V_{\text{query}} \), it holds that: \( w(u, v) = \frac{1}{|\text{OUT}(u)|} \).

Figure 3.4 shows an example of a term-query graph. The nodes with rounded corners form a regular query-flow graph as described in section 3.4.1. There are three unique terms in the queries that form this regular graph which become term nodes. These are indicated in the example by a square node.
3.4 From query-flow graphs to query suggestions

Approach

Figure 3.3: Example of (a) an annotated query-flow graph equal to $G_{\text{orig}}$ and (b) the slice $G_{\text{SC}}$ taken from $G_{\text{orig}}$.

3.4.4 Generating suggestions

Having discussed the different query-flow and term-query graphs, we now turn to the generation of query suggestions. We formulate this problem as follows: given a query $q$ and a graph model $G$, generate a set of (usually no more than 5) suggestions that are relevant to $q$’s information need.

It is important to note that if $G$ is a query-flow graph model (either $G_{\text{orig}}$ or $G_{\text{reformulationTypes}}$), we can only generate suggestions for “seen” queries, i.e. queries that are present in the query log from which $G$ has been generated. If a query does not appear in the query log, we cannot generate suggestions for it. If $q$ appears in $G$, we perform a random walk with restart from $q$’s node, known as Personalized PageRank (PPR) in the literature [41]. The result of the random walk is a probability distribution (PPR scores), i.e. the probability of walking across a particular node in $G$ when starting the walk in $q$. This is in effect the probability of users’ submitting a particular query after the initial query $q$ has been submitted. Since in this setup queries that occur very frequently in the query log have a large number of incoming edges with high weights, we normalize the PPR score by the standard PageRank score that each query (node) has. We use $\alpha = 0.85$ in both PPR and PageRank calculations, which is a standard value. We then filter out all the suggestions that are not in tag space $\mathcal{T}$, with the exception of special node $t$. 

Figure 3.4: Example of a term-query graph.
The normalized PPR score is then used to rank the queries (nodes) and the highest scoring queries are selected as suggestions for our starting query $q$. The rank of node $t$ in our generated ranking is also of importance: all queries (nodes) ranked below $t$ have a lower likelihood of being submitted after $q$ than the user stopping her search session. Thus, these queries are considered not to be useful and are stripped from the suggestion list (as well as $t$ itself).

When $G$ is the term-query graph, we first have to split the initial query $q$ into $w_1, \ldots, w_m$ individual terms. We perform $m$ random walks with restart, each one starting at one of the $w_i$ (if they exist in the graph). The random walks are conducted in the same manner as described for the query-flow graphs. The result is $m$ probability distributions, one for each of $q$’s terms. We then calculate the Hadamard product [17] of these probability distributions, which assigns only to those suggestions that appear in all $m$ distributions a score greater than zero. The suggestions are ranked by their Hadamard score and as before suggestions ranked below node $t$ are removed from the final list of suggestions.

### 3.5 Training and evaluation

To train and evaluate our suggestions, we split our query log (ordered by time) into two sets, equivalent to “seen” and “unseen” queries. We generate the query-flow and term-query graphs on the seen set. We sample queries from the unseen set and derive query suggestions for them employing the eight models just described. The overall effectiveness of a model is then measured in usefulness and coverage.

In order to evaluate the usefulness, we ask human evaluators to judge the usefulness of the presented query suggestions to the initial query $q$ as either Useful, Somewhat Useful, Not Useful or Don’t know. The usefulness measure, or $u$-score, is the percentage of queries for which the top-5 suggestions contain at least one Somewhat Useful or useful suggestion [15, 16, 17].

Additionally, we also evaluate the models according to their coverage, i.e. the number of queries for which suggestions can be made at all. We are not able to generate suggestions for every single query, because our use case has a limited tag space $T$. If we can produce very useful suggestions for only for a limited set of queries, then this suggestion method is not very effective. This also holds the other way around. Therefore the $u$-score alone is not sufficient to evaluate the effectiveness of our models and thus we measure the coverage as well. The evaluation process is discussed extensively in Chapter 5.
Chapter 4

Experimental Setup

In this chapter, we will discuss the setup we used for our query suggestion pipeline. This includes the query log data extraction, setup and evaluation of the pre-processing steps and details on the implementation of the query-flow graphs and random walk algorithms. The evaluation of the query suggestion effectiveness and the results of this evaluation is discussed in the Chapter 5.

4.1 Data sets

We extracted two data sets from startpagina's query log: (1) data set SP1, i.e., the 'seen' data, contains tens of millions of queries sampled from startpagina's query log collected between April 14, 2014 and June 13, 2014; (2) data set SP2, i.e., the 'unseen' data, contains all queries issued in the seven days starting at June 14, 2014. We use SP1 to train our pre-processing components and query-flow graph models. Data set SP2 is employed to evaluate the query suggestions generated by our pipeline. This is a realistic setup, as we may be only able to (re-train) our models at particular moments in time.

For data set SP1, we only extracted the queries and omitted the click data as it is not needed to feed the models. During extraction the raw query logs are ordered, firstly by user ID and then by date and time. This ordering is necessary in the pre-processing steps and the data platform allows us to do this more easily at this moment in the pipeline then later on.

4.2 Pre-processing the query logs

Before we can generate query-flow graphs, we have to pre-process the query logs. There are a few steps involved in pre-processing. First we use the query logs to train a session splitter, which is then used to split the full query logs into sessions. We use subsequent query pairs in the sessions to train a query reformulation type classifier. This is used to remove noise that is present in the query logs resulting from the instant results feature of startpagina. Finally, we can use the cleaned session files and the reformulation type classifier to generate the query-flow graphs. We will discuss each of these steps in this section. An overview of the pre-processing steps and components is shown in Figure 4.1.
4.2 Pre-processing the query logs

Experimental Setup

4.2.1 Search session splitting

The first step in training our session splitter is to split data set \( SP_1 \) into super-sessions based on the user ID and the 60 minute heuristic. We randomly selected 974 super-sessions and manually marked the boundaries of sessions within the super-sessions\(^1\). This is done by marking the first query of each session in the supersession. An example of this task is shown in Figure 4.2, in which queries 0, 11 and 19 are marked to be the session starters. The topics of three search missions are: buying a specific house, losing weight and car taxes.

\( \text{Figure 4.2: Example of a session splitting annotation task.} \)

\(^1\)The author of this thesis was the sole annotator.
Experimental Setup

4.2 Pre-processing the query logs

In 38 cases, we were unable to split the super-sessions into sessions in our manual annotation process. Reasons for this were unclear search goals within the super-sessions or uncertainty of what the boundaries should be. A total of 936 annotated super-sessions were left, yielding 2489 search sessions containing a combined total of 9410 queries. We calculated the 23 features described in Section 3.3.1 for each subsequent query pair in the super-sessions and then started a 5-fold cross-validation process to train and evaluate the performance of both the C5.0 classifier and the baseline which is only based on a timeout value. We also performed this process on the annotated (Hagen13) [29] data set to validate our method and for comparative purposes. The results of the evaluation are shown in Table 4.1.

<table>
<thead>
<tr>
<th>Splitting method</th>
<th>Data set</th>
<th>Correct</th>
<th>Precision</th>
<th>Recall</th>
<th>F$_{15}$-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inactivity (2 min)</td>
<td>SP1</td>
<td>68.46%</td>
<td>0.65</td>
<td>0.95</td>
<td>0.84</td>
</tr>
<tr>
<td>Classifier (C5.0)</td>
<td>SP1</td>
<td>82.64%</td>
<td>0.79</td>
<td>0.99</td>
<td>0.92</td>
</tr>
<tr>
<td>Inactivity (11 min)</td>
<td>Hagen13</td>
<td>67.35%</td>
<td>0.83</td>
<td>0.88</td>
<td>0.86</td>
</tr>
<tr>
<td>Classifier (C5.0)</td>
<td>Hagen13</td>
<td>84.51%</td>
<td>0.82</td>
<td>0.99</td>
<td>0.93</td>
</tr>
</tbody>
</table>

We were able to recreate the classifier performance of the literature [29] with our own implementation (when considering only their non-semantic methods). Furthermore, we attain similar results on our own data set. The inactivity based approach leads to 68% fully correctly identified sessions for SP1 while our classifier-based approach makes correct decisions for 83% of the sessions. We therefore choose to move forward using the classifier approach in our query suggestion pipeline as we are able to split the Dutch startpagina sessions with sufficient accuracy for our purposes.

Notable is the difference in precision and recall in these results. Precision is a measure of false positives and recall is a measure of false negatives. In training our boundary classifier, we configured false negatives to have a higher cost penalty than false positives. We did this because we consider false negatives to be more harmful to a query-flow graph than false positives (see Section 3.3.1 for an explanation).

The C5.0 classifier allows us to extract the most significant features used in the classification. For session splitting, these feature are the cosine similarity (both on the word and query level) as well as the Levenshtein distance normalized by the queries’ time difference.

4.2.2 Reformulation type classification

We ran our session splitting classifier on the entire SP1 data set and then randomly selected 1806 subsequent query pairs from the identified sessions. We then manually annotated the reformulation types of these query pairs$^2$. An example of this task is shown in Figure 4.3. All the available information of the session was provided in the task to give the annotator an understanding of what this particular user was searching for and what queries the user entered to do this. The two queries with the arrows in front of them form the query pair for which the reformulation type must be marked by the annotator. In this case, the reformulation type is marked as a specialization.

$^2$Again, the author of this thesis was the sole annotator.
4.2 Pre-processing the query logs

We were able to manually annotate 1738 query pairs with their respective reformulation type. The other 68 query pairs were marked as unclear. We used the annotated pairs to train and evaluate our reformulation type classifiers in a 5-fold cross-validation setup. We investigated the five binary classifiers, the $n$-class classifier and the classifier cascade setup, all of which are based on the C5.0 algorithm.

The accuracy of each classifier is shown in Table 4.2. We see that the accuracies of the binary G, P and C classifiers are very high. The type generalization (G), is the one class where words are often removed and the query is often reduced in length. For a parallel move (P), a significant change is that some words are replaced by others. And for an error correction (C), the change is often very small and is not located in the end of a query where changes often are the result of users that are still typing. These significant features make it easy to classify these classes which results in high accuracies for the binary classifiers.

The cascaded classifier performs somewhat better than the $n$-class classifier, which is expected. Although the difference is small, the cascade successfully classifies the sure cases first by using the binary classifiers and uses the $n$-class classifier if a reformulation type could not be classified by its binary counterparts. Although the difference is small, we move forward with the cascaded classifier as its accuracy is higher than the $n$-class classifier, and the extra costs in calculation time are minimal.

Table 4.2: Overview of reformulation type classification accuracy on data set SP1.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary G</td>
<td>99.47%</td>
</tr>
<tr>
<td>binary P</td>
<td>97.18%</td>
</tr>
<tr>
<td>binary C</td>
<td>97.15%</td>
</tr>
<tr>
<td>binary S</td>
<td>89.22%</td>
</tr>
<tr>
<td>binary T</td>
<td>88.40%</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Classifier setup</th>
<th>$n = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-class classifier</td>
<td>84.8%</td>
</tr>
<tr>
<td>cascade classifier</td>
<td>85.9%</td>
</tr>
</tbody>
</table>

(b)

We also tested with classifiers where we consider a specialization (S) and typing (T) to be the same class. The reason for this is the fact that typing can be considered to be a special form of specialization and thus differentiating between the two is very hard. The binary ST classifier has an accuracy of 95.79%, the $n$-class classifier and the cascaded classifier have accuracies of respectively 94.3% and 94.5% when $n = 4$. We
did not use these classifiers as we consider T types to be noise when using the logs for query-flow graph generation.

Having determined the effectiveness of our classifier cascade, we classified all query pairs in SP1 and considered the distribution of reformulation types (Table 4.3). Interestingly, compared to previous work, in particular [16], which use Yahoo! query log data from 2008, we find considerable differences for the parallel move and specialisation types: parallel moves are found in less than 17% of all cases in SP1 when we filter out T types, while it is the most prevalent reformulation type in the Yahoo! logs (roughly 50% of all reformulations). In contrast, we find that in SP1 the S and C types occur with considerably higher frequency than found in the standard Web search setting. Despite the fact that for many queries startpagina does not return any results (because the query does not appear in T), more than 50% of the reformulations lead to more specific queries. This is contrary to our intuition, as we would expect users to back-off to more general queries in order to receive startpagina results. More research is needed to investigate this behavior as well as the lack of parallel moves - we leave this to future work.

Table 4.3: Overview of the types of reformulations found in data set SP1 and in previously published works [16].

<table>
<thead>
<tr>
<th>Reformulation type</th>
<th>SP1 (with T)</th>
<th>SP1 (without T)</th>
<th>Yahoo! UK, 2008</th>
<th>Yahoo! US, 2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>7.51%</td>
<td>13.01%</td>
<td>4.40%</td>
<td>9.50%</td>
</tr>
<tr>
<td>P</td>
<td>9.38%</td>
<td>16.26%</td>
<td>47.70%</td>
<td>55.50%</td>
</tr>
<tr>
<td>C</td>
<td>10.48%</td>
<td>18.16%</td>
<td>10.40%</td>
<td>5.00%</td>
</tr>
<tr>
<td>S</td>
<td>30.33%</td>
<td>52.56%</td>
<td>37.50%</td>
<td>30.10%</td>
</tr>
<tr>
<td>T</td>
<td>42.29%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

4.2.3 Noise removal

Due to the instant result feature of startpagina, some noise is introduced into the query logs. Part of this noise are duplicate queries, which can easily be removed. The other part of the noise is introduced when users are typing slow or are distracted in some way which causes the query they typed so far to be logged. Clear evidence for this can be found in the query logs where pyramid-like structures are present, such as the example shown below. In this example, the only queries where the user stopped typing for a while are queries 5 and 9. These are the only two interesting queries for the query-flow graphs, thus the only edge that must be created is between their nodes.

1. acti
2. activi
3. activitei
4. activiteiten
5. activiteiten hemel
6. activiteiten hemelvaa
7. activiteiten hemelvaarts
8. activiteiten hemelvaartsda
9. activiteiten hemelvaartsdag
We use the reformulation type classifier to filter out this noise. First we classify all the reformulation types in a single session. If the type of a subsequent query pair is T, then we remove the first query of the pair. After we removed all the reformulations of type T, we reclassify the reformulation types of the query pairs that have not been removed which leaves us with only useful queries. A schematic overview of this procedure is shown in Figure 4.4.

![Diagram showing noise removal process in a session.](image)

**Figure 4.4: Noise removal process in a session.**

### 4.3 Implementation details

When choosing the technologies and algorithms for our implementation, we considered the fact that Sanoma wants to test and potentially use the query suggestion pipeline in a production environment. Although it was not our goal to produce a system that is ready to be used in a production environment, we attempted to choose our technologies such that Sanoma has the ability to further develop this for live usage.

Since we are dealing with large amounts of data, tens of millions of queries each month, we decided to implement the pipeline such that we could process the data in a distributed way. This allowed for further growth and smaller graph generation times when recalculating the model. We could have used single machine efficient sparse graph representation libraries but we expected the calculation of these graphs to be large in both time and space, making this an unsuitable solution. Another reason for not using a standard library is that we were in full control of the representation by implementing our own format. It is also possible to implement a query-flow graph that can be updated opposed to recalculating the whole model [12]. We did not choose to do this as it adds unnecessary complexity for this research (we only calculate the graph for SP1 once) and we expected the distributed calculation of the model to be fast enough for live usage. To create a distributed implementation, we chose to use the Apache Hadoop framework [1] and the MapReduce model [25] as Sanoma already has the infrastructure to use this. They currently maintain an Hadoop cluster of 42 nodes. Furthermore, the logs are already imported into the Hadoop Distributed File System, making it easy to access this data from our MapReduce jobs.
4.3.1 Query-flow graph generation

After we have extracted data set SP1, the data is in tab-separated form. Each line in the query logs contains a user ID, timestamp and query. The lines themselves are ordered firstly by user ID and then by the timestamp. This ordered data is used as input for the first MapReduce job that converts the query log into individual sessions. The output of this job are files containing lists of queries that form a session on each line, including the reformulation types of every subsequent query pair. The following line is an example of a line in the session files:

```
snow owl#P#forest owl#G#owl#S#owl food
```

The session files are used as input for the second job that converts the sessions into the query-flow graphs. We used a single representation for all three query-flow graph models that we experimented with. The output of the second job is an adjacency list representation of the graph, of which the following line is an example:

```
snow owl 0.004 1|2|0|0|0|0 !#forest owl#0|2|0|0|0|0#owl#1|0|0|0|0|0
```

In this tab-separated example, the initial query snow owl is the first value. The second value is the initial PageRank value of node snow owl. PageRank values are discussed in the next section. The third value is the number of queries that succeeded the preceding query where the sums are split by the reformulation types. The |-separated numbers respectively belong to types G, P, C, S, T and special type X. The example shows that a generalization to another query has occurred only once and parallel moves occurred two times. No other reformulations types occurred. Finally, the fourth value of the line is the edge list, which contains the succeeding queries and the number of times that a reformulation to this query occurred using the same notation as used in the third value. Note that a subsequent query pair can have multiple reformulation types as annotations because of the temporal factor that is used in reformulation type classification.

The term nodes are also present in the adjacency list. The node values of a term node are proceeded with a #-sign to separate them from the query nodes. The output of these lines is similar to the line mentioned above. Note that we do not have to calculate the sum of transitions here, but we do count the number of unique queries in which a term occurs. An example of a line is show below:

```
#owl 0 3 !#snow owl#forest owl#owl
```

As stated before, the adjacency list representation can be used for all models we experimented with. Due to the distributed setup, generating this graph from tens of millions of queries in the query log takes less then two minutes on Sanoma’s Hadoop cluster. Note that the graph itself is complete but we cannot generate query suggestions from it yet because we need the PageRank value of each node in order to rank the query nodes.
4.3 Implementation details

4.3.2 Distributed PageRank calculation

After we obtained the adjacency list representation of the graph, the next step is to calculate the PageRank values for each node. The PageRank values can be calculated using the power method, but this would take a lot of time on a single machine. We implemented a distributed power method algorithm [37] to speed up the process and make the calculation scalable. We will first explain the basics of a MapReduce job and then describe the job for one single iteration of the power method used to calculate the PageRank values.

A MapReduce job consists of two phases: the map phase and the reduce phase. When the job starts, the input is split into pieces and send to a number of mappers. The number of mappers often depends on the total size of the input. The mapping phases is often used for pre-processing purposes: the data is filtered and sorted and grouped into key-value pairs. These intermediate values are then sent to the reduce phase where the actual processing on the data is done. Again, there are usually multiple reducers performing the same processing. Values with the same key are all sent to the same reducer. Finally, the output of all the reducers make the output for the whole job.

We will first discuss the mapping phase of our job. Equation 2.1 in Section 2.2.1 shows that the PageRank value of a node A is composed of a random jump value and of the sum of the PageRank values of nodes that have an edge to node A. Obtaining the random jump value is straightforward, but we do not have a list of incoming edges in our graph representation. We thus need to reverse the adjacency list, which is done in the map phase of our distributed algorithm. For every node A, there is a list B of nodes to which A has an edge. For every node C in B, we write an output line with node C as the key and a combination of node A and A's current PageRank value as the output value. We also write a special output line with node A as the key and the adjacency list entry (the line containing node A) of node A as an output value. These key-value pairs are sent to the reducers.

In the reduce phase, a reduce task receives an input key, which is a node, and a list of nodes that have an edge to that node and their current PageRank values. We now have all the information we need to calculate the new PageRank value of the input key node. There is also one input value that contains the adjacency list entry of the input key node. This entry is modified with the new PageRank value and written to the output of the job.

Note that this method requires the nodes to have a starting PageRank value which is used by the first iteration of the power method. This particular algorithm only requires that the starting PageRank values for each node are equal, but it is common to use $\frac{1}{N}$ where $N$ is the number of nodes in the graph. We choose $N$ to be the number of query nodes and gave all term nodes a starting value of 0. We do not included the term nodes in the PageRank calculation as they are not eligible as query suggestions.

It takes an average of 45 seconds to perform one iteration of this distributed power method on a graph with $\sim$15 million nodes and $\sim$18 million query edges. We used 25 iterations to converge the values to a steady state. After each iteration, the adjacency list is equals to its predecessor except for the updated PageRank values.
4.3.3 Real-time Monte Carlo Personalized PageRank

The only differences between the global PageRank calculation and the Personalized PageRank (PPR) calculation are the probabilities of random jumping to a particular node. If we would use the distributed power method to calculate the PPR values for the queries, then we cannot calculate suggestions at query-time and would have to pre-calculate every query in order to do real-time suggestions. This is impractical as it takes a lot of time to do this. We therefore turn to Monte Carlo Personalized PageRank calculations [27, 6, 8].

In Monte Carlo random walks, an actual random walk on the graph is performed. One starts at a particular node and randomly walks across an edge to another node with a weighted chance proportional to the edge weights. With a probability $1 - \alpha$, the random walk stops. We count the number of times each individual node is visited and use those values to approximate the PPR scores. The approximation becomes more accurate if the number of performed random walks is increased.

Although this is often being used in a distributed setting [27, 6, 8], we implemented a single machine version. We used Redis [3], an open source in-memory key-value store, to store the adjacency list. The first entry on the tab-separated line, the node, is used as a Redis key while the other entries are stored in the Redis value. We then implemented a Java application that performed the Monte Carlo random walk 5000 times and then estimated the PPR value for the nodes that were visited. This PPR score is then divided by the global PageRank value, which is also stored in Redis, which creates the scores needed for ranking the queries.

Using this setup, we could generate query suggestions for a given query in real-time (usable in a live search engine). Suggesting using the term-query graph takes somewhat longer due to term nodes with a high number of edges. We pre-calculated the PPR scores for these term nodes in order to generate real-time query suggestions using this model as well. This takes considerable less space than pre-calculating every single query node.
Chapter 5

Evaluation

We built the query-flow graphs from SP1 using our implemented pipeline: the query logs were split into sessions, the reformulations were annotated, query log noise was removed and we build the query-flow and term-query graphs. Now we will discuss the evaluation setup we used to measure the query suggestion effectiveness as well as the results of this evaluation. These results are used to draw conclusions in chapter 6.

5.1 Generated query-flow graphs

The graph models we experiment with in our research are: the original query-flow graph model $M_{\text{orig}}$, four slices of $M_{\text{orig}}$ where one reformulation type is omitted in each slice and the term-query graph model $M_{\text{TGM}}$. We generated the models from dataset SP1 and calculated basic statistics for these models. These are shown in Table 5.1.

The original model has a node to edge ratio (or average outdegree) of 1:1.385, which makes this a very sparse graph. In a slice of the original graph the edges of particular reformulation types are removed but the number of nodes remains the same. This makes the sliced graphs even sparser, even to the point where there are less edges than nodes in slice $M_{\text{GPC}}$. The amount of edges that are omitted in each slice is consistent with our expectations, which are based on the distribution of reformulation types (see Table 4.3 in Section 4.2.2). The term-query graph does have an increase in number of nodes as the term nodes are added to the original graph. This also causes a dramatic increase of edges in the graph as new edges are formed between the term nodes and the query nodes that contain the terms.

For the original graph, the query node having the maximum outdegree is the query camping. This is also the case for three of the slices. In slice $M_{\text{GPC}}$, the term marktplaats has the highest outdegree. The maximum outdegree in the term-query graph is the term node of the Dutch article de. Because the graph is very sparse and because we have stopwords in the graph, we believe that pre-processing techniques such as stop-word removal and dangling node removal can be employed to scale down the graph size, which increases the performance of graph algorithms such as calculating PageRank. Changing the graph structure has to be done with care though, as the PageRank values are influenced by changes in graph structure. We did not implement such pruning techniques in this research and consider it to be future work.
5.2 Evaluation setup

We evaluated our graph models constructed from SP1 on a second data set SP2, which is used to simulate unseen search behavior. We first split SP2 into search sessions and randomly selected 282 of them. On average, each session contained 2.9 ($\sigma = 1.65$) queries. We then randomly drew a single query from each session and generated the top-5 query suggestions using each of our six models. We also captured the top-5 query suggestions from the CSE that is currently employed by startpagina. We merged the suggestions generated by all models for a given query into a single list, removing duplicates. Due to the overlap in generated suggestions between the different models, the final list had far fewer than the possible $5 \times 7 = 35$ suggestions. On average 9.85 ($\sigma = 3.49$) unique suggestions were generated per query.

We recruited 31 native Dutch speakers as human evaluators (11 human annotators employed at startpagina and 20 startpagina users). The evaluators were presented with a search session and the selected query from that session for which the suggestions were generated. To ensure that our evaluators made high-quality judgments, we asked them two initial questions for each session/query pair: (1) do all queries in the session have the same search goal (yes/somewhat/no), and, (2) do you understand the user’s search intent (yes/somewhat/no). The evaluator was not allowed to judge the suggestions if one of these questions was answered with no. Each session/query pair was judged by a single evaluator. An evaluator could judge up to a maximum of 20 session/query pairs to ensure that there was no bias to a single evaluator’s opinion.

5.2.1 Evaluation form

Before the evaluators started their judging tasks, they were shown a short introduction text that describes the task they are required to do. After filling out a form containing their personal data, they were taken to the suggestion evaluation form. Figures 5.1 and 5.2 show the two parts where the evaluators were asked questions. The Dutch introduction text and a full screenshot of the evaluation form is provided in appendix A of this thesis.

In the first part of an evaluation task (Figure 5.1), the evaluator is asked to look at the complete session, which consists of three queries in this example. For each query, it is given whether startpagina can show results for the query and if so, if the user clicked on one of the results. The evaluator must use this information to get an understanding of the search intent of the user. The underlined query is the query

<table>
<thead>
<tr>
<th>Model</th>
<th>#nodes</th>
<th>#edges</th>
<th>average outdegree</th>
<th>maximum outdegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_{orig}$</td>
<td>14,153,454</td>
<td>19,611,085</td>
<td>1.385</td>
<td>13,699</td>
</tr>
<tr>
<td>$M_{GPC}$</td>
<td>14,153,454</td>
<td>12,638,766</td>
<td>0.893</td>
<td>775</td>
</tr>
<tr>
<td>$M_{GPS}$</td>
<td>14,153,454</td>
<td>17,271,592</td>
<td>1.220</td>
<td>13,643</td>
</tr>
<tr>
<td>$M_{GCS}$</td>
<td>14,153,454</td>
<td>16,697,988</td>
<td>1.180</td>
<td>13,650</td>
</tr>
<tr>
<td>$M_{PCS}$</td>
<td>14,153,454</td>
<td>18,139,039</td>
<td>1.282</td>
<td>13,696</td>
</tr>
<tr>
<td>$M_{TGM}$</td>
<td>17,686,843</td>
<td>59,658,990</td>
<td>3.373</td>
<td>552,735</td>
</tr>
</tbody>
</table>
Figure 5.1: First part of the evaluation form where the evaluators’ capability to judge the suggestions is tested.

Figure 5.2: Second part of the evaluation form where the evaluators judge the usefulness of the query suggestions.
for which suggestions are generated. In this case, the suggestions are thus generated for query *occasion lease* (*occasion* is the Dutch word for *second hand car*). The suggestions are thus generated in the situation before the other two queries are entered. The models have no knowledge of this two follow up queries.

The evaluator is then asked, in question one, if the queries in the session are coherent and serve one single search goal. If the evaluator thought that this was not the case, then the evaluation task was done and the evaluator moved on to the next task. Any other answer allowed the evaluator to answer the second question. The evaluator must state if he or she understands the search goal of the user. Again, if this question is answered with no, the task was done. Any other answer allowed the evaluator to judge the query suggestions, which is part 2 of the evaluation task shown in Figure 5.2.

In this second part the query for which the suggestions are generated is shown again. It is the first query in the session: *occasion lease*. The user apparently wants to lease a second hand car, preferably of the brand Fiat, judging by the session information. The evaluator has the ability to rate the suggestions as either *Useful*, *Somewhat Useful*, *Not Useful* or *Don’t know*. Again, the suggestions are ordered randomly and the evaluator does not know which suggestions were generated by what model(s).

### 5.2.2 Evaluation task examples

Table 5.2 contains an example of query suggestions generated for a query (underlined) of the following search session consisting of two queries: \{*egon* → *aegon*\}. Aegon is a multinational life insurance, pensions and asset management company. In the selected first query, the user misspelled the name of the company. The distinct list of suggestions from all models are listed in the table. Query suggestions that do not appear within *T* are marked with †. The final column contains the ratings given by our human annotators: either U (*Useful*), US (*Somewhat useful*), or, NU (*Not useful*).

The table shows that there is a lot of overlap among the top-5 suggestions that are generated by the models. Especially, between the original model its slices. The GPC-slice and the term-graph suggestions differs most from the other query-flow graphs. The CSE has no overlap in suggestions with any other model and none of its suggestions lead to results on *startpagina*. Overall, the suggestion got very positive ratings.

<table>
<thead>
<tr>
<th>Query suggestions</th>
<th>English translation</th>
<th>Model(s)</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>aegon</td>
<td>aegon</td>
<td>Orig</td>
<td>GCS</td>
</tr>
<tr>
<td>egon zehnder †</td>
<td>egon zehnder</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aegon hypotheek</td>
<td>aegon mortgage</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aego</td>
<td>aego</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>egon schiele †</td>
<td>egon schiele</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>mijn aegon</td>
<td>my aegon</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>egon †</td>
<td>egon</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>inloggen aegon</td>
<td>login aegon</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aeg</td>
<td>aeg</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>egon derksen †</td>
<td>egon derksen</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aegon verzekerings</td>
<td>aegon insurance</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aegon.nl</td>
<td>aegon.nl</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aegon inloggen</td>
<td>aegon login</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>aegon verzekerings</td>
<td>aegon insurances</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>egon krenz †</td>
<td>egon krenz</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Another example is shown in Table 5.3. The following session was presented to the evaluator: \{*\text{speelschema} \rightarrow \text{speelschema} \text{wk} \rightarrow \text{speelschema} \text{wk} 2014*\}. Again, the first and underlined query was the query for which suggestions were generated. The first thing to notice is that all models suggest the next two queries that the user will enter. The models did not know this while generating suggestions.

What we also see is that this query was somewhat ambiguous. Especially since three major sports tournaments were held at that moment: the world cup of soccer, the world cup of hockey (both men and women) and a major tennis tournament. We do not know what schedule is meant, even by looking at the whole session, although it is clear the user is looking for some sort of sports schedule. Suggesting queries for diverse sports is desired in this case. Especially the term-graph model does this well in the example as the regular \text{speelschema wk} query is suggested as well as the hockey and tennis schedules. Since soccer is the most popular sport in the Netherlands, a user may expect the \text{speelschema wk} query to show results for the world cup of soccer.

Table 5.3: Example of query suggestions generated for the query: \text{speelschema}.

<table>
<thead>
<tr>
<th>Query suggestions</th>
<th>English translation</th>
<th>Orig</th>
<th>GCS</th>
<th>GPC</th>
<th>GPS</th>
<th>PCS</th>
<th>TGM</th>
<th>CSE</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{speelschema wk}</td>
<td>schedule wc</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>\text{speelschema eredivisie} †</td>
<td>schedule eredivisie</td>
<td>x</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema wk hockey vrouwen}</td>
<td>schedule wc hockey women</td>
<td>x</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spel</td>
<td>playing</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema wk hockey}</td>
<td>schedule wc hockey</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema nederlands wk} †</td>
<td>schedule netherlands wc</td>
<td>x</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema ajax} †</td>
<td>schedule ajax</td>
<td>x</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema heren rolland garros}</td>
<td>schedule men rolland garros</td>
<td>x</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema w}</td>
<td>schedule w</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speel</td>
<td>island</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema 2014}</td>
<td>schedule 2014</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>SU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{speelschema wk 2014}</td>
<td>schedule wc 2014</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The examples above are all examples where the query suggester produced at least 'somewhat useful' suggestions. There are however examples where the suggestions were not useful at all. The results indicate that a common source for 'not useful' ratings are unfinished queries and spelling errors. A query containing spelling errors is less likely to be a node in the query-flow graph, so no suggestions can be generated for these queries. Common spelling errors do occur in the graph, but there is a high probability that users correct these errors and thus the chance of random walking to the correct spelling is high. This means that a spelling correction is given as a suggestion (if the suggestion in is \(T\)) and similar suggestions can be generated as the walk continues from the correct node.

Table 5.4 shows an example of a spelling error in a query. The user is searching for *vochtphoping na bortstamputatie*, which contains multiple spelling errors. It is obvious that the user wants to search for *vochtophoping na borstamputatie*, which translates to 'fluid accumulation after mastectomy'. In this case, there are multiple reasons that contribute the 'not useful' rated suggestions. Almost none of the suggestions are remotely related to the intended search goal. The last query is the exception to this as it relates to the mastectomy part of the query, although this word would probably never be used in that context in Dutch.

The first reason that leads to 'not useful' ratings is the misspelling itself. Although it is clear to humans what is meant by the user, both our models and the CSE could
5.3 Evaluation results

Not identify and correct the error. Apparently, this error has been made before as the query node corresponding to this query exists in the graph. It is unlikely that these spelling errors were introduced twice so the users probably copied the wrong spelling from another website, or there is a faulty link to the search engine somewhere on the web. We do not know this for sure. We do know that the user never corrected the error, as there are no edges to the node that corresponds to the correctly spelled query.

The user did try to formulate another query as there are in fact two edges to other nodes with equal weights in the graph: *tie* and *tatie*. We think that the user tried to select and remove the query entirely, but failed to do so and left some part of the query in the search box. This caused the remaining part of the query to enter the query logs as noise, which is the second reason for the negative ratings of the suggestions. These substrings of the original query occur in various other queries in the graph. There is a reasonable chance that the nodes corresponding to these substrings of queries have edges to other, unrelated query nodes. The random walk algorithm walks over these edges, which is exactly what happened here, causing the suggestions to be completely unrelated.

But even if the user would spell the query correctly, our models are still not able to generate proper suggestions in this case. The list of top ranked queries after random walking includes various queries containing the fluid accumulation part, but these are all not in the tag space $T$. After filtering, we only get the following ordered results that are in $T$: *gezondheids plein* and *gezondheidsplein*. The model states that none of the results after suggestions $T$ are useful as the probability that users end their sessions is higher then the probability of using one of the lower ranked queries. However, *gezondheidsplein* is a site containing all sorts of health care related information and might therefore be interesting to this user. It would be interesting to investigate how scaling the score of node $T$ effects the usefulness and the coverage score of the models. We did not investigate this in our research and consider this to be future work.

5.3 Evaluation results

Of the 282 session/query pairs we extracted for our evaluation, the evaluators judged 40 pairs as not having a single search goal, while for an additional 53 sessions the evaluators were not able to clearly identify the search intent of the session. For the remaining 189 session/query pairs the evaluators were asked to rate the list of distinct suggestions with respect to their usefulness. A total of 1886 suggestions were rated.
We find that more than half of all 1886 suggestions were judged at least somewhat useful by our evaluators. More specifically: 28.2% of suggestions were deemed useful, 23.44% where somewhat useful, 44.11% were not useful and 4.24% suggestions were rated as don’t know.

We use the ratings of the evaluators to measure query suggestion effectiveness which is a combination of usefulness and coverage. The usefulness is expressed in the u-score, which is the percentage queries of which the top-k suggestions contain at least one useful or somewhat useful suggestion. We measured the the u-scores for the top-1, top-3 and top-5 suggestions of each model. We also calculate the percentage of queries for which each model could generate suggestions, which is the coverage metric. An overview of these metrics is presented in Table 5.5.

Table 5.5: Overview of the query-flow graph models’ effectiveness. All but the CSE suggestions are generated using SP1 as training data.

<table>
<thead>
<tr>
<th></th>
<th>u-score</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>top-1</td>
<td>top-3</td>
</tr>
<tr>
<td>M_orig</td>
<td>50.90%</td>
<td>59.88%</td>
</tr>
<tr>
<td>M_GPC</td>
<td>49.61%</td>
<td>62.20%</td>
</tr>
<tr>
<td>M_GPS</td>
<td>50.68%</td>
<td>59.46%</td>
</tr>
<tr>
<td>M_GCS</td>
<td>59.12%</td>
<td>67.15%</td>
</tr>
<tr>
<td>M_PCS</td>
<td>55.05%</td>
<td>59.63%</td>
</tr>
<tr>
<td>M_TGM</td>
<td>64.71%</td>
<td>70.59%</td>
</tr>
<tr>
<td>CSE suggestions</td>
<td>77.50%</td>
<td>79.17%</td>
</tr>
<tr>
<td>∈T</td>
<td>79.67%</td>
<td>84.62%</td>
</tr>
</tbody>
</table>

5.3.1 Overall performance

The u-scores for the top-5 suggestions are comparable to the literature, which attains u-scores in between 50% and 60% [15, 16, 17]. Our models score are a little higher than the literature which we believe is caused by the fact that our generated suggestions have been deemed relevant for at least one topic by the annotation team of startpagina. They have assigned the tag to the content which makes it eligible as a suggestion.

We also observe that the u-scores improve by about ten to twenty percent when comparing the top-1 scores with the top-5 scores. The scores that are ranked higher in the suggestion list are thus useful more often, which confirms that our random walk algorithm is a proper ranking method for generating query suggestions. startpagina currently shows three suggestions in its search interface. The effect of omitting the last two suggestions from the top-5 is limited as shown in the table.

Although having a limited set of queries that we can suggest is of positive influence to the u-score, it has a negative impact on the coverage. The ranking algorithm considers all queries to be be returned as suggestions but only returns them if they have a match in T. This improves the chance that special node T becomes the top suggestion, which means that we decide to not return any suggestions at all. This decreases the coverage. Note that M_orig and the four slices can only return suggestions that are present as a query node in the graph, which means that no suggestions can be made for new, unique queries. This in itself already decreases the coverage metric.
5.3 Evaluation results

5.3.2 Influence of the reformulation types

We will discuss the influence of the reformulation types by looking at the top-5 u-scores and the coverage. When comparing the four reformulation type models, we observer that S and G are both important for coverage. This is not unexpected, as S-class reformulations account for more than 52% of all reformulations (Table 4.3). More interesting is the influence of the G-class edges. Although they account for only 13% of all reformulations (less than P or C), they are essential to achieve high coverage (Morig achieves 88% coverage, MPCS only 56%). We reason that startpagina’s annotations play a role here: complex long queries rarely appear in T, and thus simpler queries that generalize (i.e. are shorter and may appear in T) are valuable for suggesting queries.

With respect to the u-score, we find that including the P-class (i.e. parallel moves) has the most negative effect on the u-score. Models that include the P-class lose 9% in u-score on average. We believe that parallel moves connect several distinct search topic paths together, thus adding noise in the graph. This is exactly what we tried to prevent during the session splitting in the query logs. The influence of other reformulation types on the u-scores is negligible.

5.3.3 Influence of the term nodes

The term-graph model (M_{TGM}) exhibits the highest u-score across all our models, i.e. it generates the best, most useful suggestions for those queries for which it is able to generate suggestions. At the same time though, the coverage is much lower than any of the other models (less than 36%), making it unusable in practice. This model was designed to have a higher coverage for queries that have not been seen before. It is therefore notable that the model has the lowest coverage in our evaluation. We believe that the reason for this is the limited tag space combined with the specific random walk method that is used for generating query suggestions using the term-query graph model. The individual lists of suggestions per random walk from a term are already sparse and the overlap between them is expected to be even more sparse (or nonexistent). Especially node T will be present the walks from each term and thus get a high rank, which means that no suggestions are made.

For future work on a CSE, it will be interesting to investigate a combination of reformulation type models and term-query graph models. They both improve the u-score and the term-query graph improves the coverage in a regular setting as well as it can suggest for unseen queries [17]. Unfortunately, this is not feasible for the startpagina use case as the coverage for the individual models is already lower than the original mode. A combination would result in an even lower coverage, thus almost no suggestions could be made.

5.3.4 Comparison with the CSE

When comparing our models’ results to the CSE suggestions we first notice the low coverage CSE suggestions achieve. Less than 26% of the top-5 generated suggestions contains suggestion that is a tag in startpagina’s tag space T. This is not surprising, as the CSE suggestions are agnostic to startpagina’s content. With respect to u-score (which ignores whether or not the suggestions are in T), CSE suggestions considerably outperform our models, indicating that there is a large potential for future
improvements. Thus, we conclude that although the suggestions provided by the CSE are useful ones, they do not help in finding startpagina content most of the time.

Note that the API that is used to retrieve the CSE suggestions only returns five suggestions which are not necessarily in $T$. Because we cannot request a top-5 with only suggestions in $T$, we had to filter the suggestions that are not in $T$ ourselves. This often leads to all suggestions being filtered out, leaving an empty top-5 list. We therefore expected the coverage of the CSE suggestions $\in T$ to be low.

5.3.5 Suggestion overlap

We measured the overlap between the top-5 suggestions of the various models. The metric we used is the percentage of queries for which the top-5 suggestions of two models have at least one suggestion that occurred in both top-5 lists. We disregard the rank of the suggestions in this metric. When we compare $M_{\text{orig}}$ to the four slices of itself, we find that the overlap is over 92% for $M_{\text{GPC}}$, and an average of 99% for the other three slices. This is not unexpected as the slices are the same models as the original with some edges removed.

For almost 85% queries, both $M_{\text{orig}}$ and CSE suggestions overall could generate suggestions. In 75% of these cases, there was no overlap in the top-5 suggestions of both models. This indicates that our investigated approaches relying on query-flow graphs and startpagina’s query logs are considerably different from the approaches and data employed by the CSE we compare our work against.

The term-query graph model and CSE could generate suggestions for the same query in about one third of the cases. Surprisingly, almost 47% of the top-5 lists generated by both models for the same query contained at least one overlapping suggestions. This is almost twice as much as the original model. The term-query graph is able to generate more diverse suggestions than our other models. We believe that this explains the notable difference in overlap.

5.4 Discussion

We were able to generate query suggestions with a usefulness that is comparable to the literature. We consider the u-scores of 60% to 70% sufficient for query suggestion purposes. Due to the limited space from which valid suggestions can be returned, we are unable to generate suggestions for all queries. This was already the case for new, unique queries but now seen queries also suffer from the degradation of coverage.

Choosing the right suggestion method for a given situation thus depends on a trade-off one has to make between coverage and usefulness of the suggestions. The suggestions generated by the CSE are more useful than our own suggestions but rarely help in finding results on startpagina itself. Our models can generate suggestions for more queries, but these are less useful than the CSE’s suggestions.

For the current situation, we advise startpagina to implement and use the original query suggestion model $M_{\text{orig}}$, as it has a good balance between usefulness and coverage, our two effectiveness metrics. If this model is unable to generate suggestions, then the suggestions from the CSE should be shown, disregarding whether the suggestions have a match in startpagina’s tag spec $T$. This might lead the users to
the SERP which is hosted by startpagina, but startpagina considers this a better alternative than letting the user leave their website with no results at all.

In the current startpagina setup, there could be multiple queries that lead to the exact same result page due to the fact that the content is often tagged with multiple tags at once. At this moment, our ranked list of suggestion do not consider this fact, so the top-5 generated suggestions could all produce the same result page. When this is the case, one could argue that this top-5 consists of the best suggestions so these should be shown to the user. On the other hand, one could make a case for diversifying the results and only keep the highest ranked suggestion for a certain sets of results. In that case, all suggestions lead to different results although there might be better suggestions available. It is up to the business to decide between these options.

5.4.1 Reflection

So how do these results fit in the search engine industry? We already discussed some notable differences between startpagina and other well known search engines like Google and Bing (CSEs) in Section 3.1, such as the limited websites that can be retrieved as results and the index only contains manually assigned tags opposed to full-text or snippet indexing. In addition, one can look at the type of content that startpagina presents as results to find more differences compared to the CSEs.

CSEs commonly present two types of search results [24]: organic and sponsored results. The organic results are obtained by matching the keywords to the indexed documents as good as possible and presenting the best ranked document to the top of the result list. Sponsored results are often also relevant to the keywords, but these are in fact advertisements. Companies can bid to have their website or products returned as a sponsored result in the CSE. The CSEs earn money if a user clicks on these results (CPC) or when a sale results from the user clicking on them (CPS).

There is an ongoing discussion about how sponsored results are deceptive to the users because it is not clear which results are organic and which ones are sponsored [47]. It has been shown that users evaluate sponsored results to be less relevant to their search goals, opposed to organic results, when looking at the result presentation itself and not the pages behind the result links [33].

The visual differences between the two result types is therefore kept small resulting in users clicking the advertisements more often. Thus the CSEs make more money without the users knowing that they are being tricked into this. CSEs are not allowed to do this as presenting advertisements to consumers while they are not aware of it is illegal.

The results on startpagina are somewhat in the middle between the two result types. All the results on startpagina have been manually entered by human taggers. When a user enters a search query, the results show the documents (web links) that best fit the entered keywords according to the human taggers. So far, this looks like organic results. However, startpagina earns money with most of these results through either the CPC or CPS models discussed before, making the results sponsored. startpagina communicates the fact that all results have been manually selected by an editorial staff but it is unclear which of the resulting links are paid for. In addition, we

\[\text{It is notable that this same research also found that users rate the pages behind organic and sponsored results to be equally relevant to their search goal. Even though they judge organic results to be more useful without looking at the pages.}\]
proposed a new query suggestion technique because the current suggester often leads the user to the SERP containing organic web results and clearly marked sponsored results. Our suggestion model suggests queries that keep users on startpagina, showing startpagina results in which it is unclear which links are sponsored and which ones are non-sponsored.

We therefore would advice startpagina to be more clear about which links in their results have been paid for, which suggestions keep them on their own site and which ones send the user to the SERP. This does not only prevent them from entering the ongoing discussion about user deception in search engines, it also helps in establishing trust between startpagina and their users. If the users are aware of the reasons the page switches and have the feeling that they are in control of what links they want to follow up on, they are more likely to be satisfied and return to the site [26].
Chapter 6

Conclusions

In this chapter, we will conclude our work by answering the research questions that we proposed in the introduction of this thesis. We will first answer our four sub research questions followed by the main research question. We will also discuss potential future work resulting from our research.

6.1 Research questions

RQ1: What are the state-of-the-art methods to generate query suggestions in a CSE?

The state-of-the-art methods for query suggestion are query-flow graphs, which are constructed from the historic query data in the query logs of a search engine. Each node in the graph represents a query. There is an edge in between two queries if they are a subsequent query pair in a single user session in the query logs. We used this model in our research, as well as two extensions of the query-flow graph. The first extension uses the reformulation type of two subsequent queries to annotate the edges in the graph. A slice of the graph based on certain edge annotations can be used to create more useful suggestions. The second extension uses the terms in the queries to expand the graph with term nodes. This makes it possible to generate suggestions for queries that are not present as a node in the graph.

RQ2: How does the SSE differ from a CSE?

A specialized and limited search engine (SSE) differs from a commercial Web search engine (CSE) in the number of documents that are available in the index and in the way that these document can be retrieved from that index. In our use case, the SSE only contains curated documents that are selected manually by an editorial staff. Only high quality documents for the users of startpagina are made available in the index. The documents are annotated with tags by human annotators. These tags are used to retrieve the documents. No abstracts, full texts or other information on the documents are used to retrieve documents. Additionally, our use case was in Dutch as opposed to the literature which use English query logs.
**RQ3:** What are the effects on the query suggestion effectiveness when applied to a SSE?  
We are able to successfully recreate the whole query suggestion pipeline for the state-of-the-art, including the session splitting and reformulation annotation pre-processing steps. This confirms that these techniques can be applied successfully on Dutch query logs. We measure the effectiveness of the query suggestion capabilities using two metrics: usefulness, the percentage of top-5 query suggestions that contain at least one useful or somewhat useful suggestion, and coverage, the percentage of queries for which suggestions can be made. The usefulness of the suggested queries generated by our models is comparable to the literature. However, the coverage of queries quickly deteriorates when we attempt to apply techniques that increase the usefulness. Depending on the situation, either a higher usefulness or a higher coverage makes a suggestion method more effective.

**RQ4:** Can we propose a better query suggestion method for Startpagina?  
When selecting a query suggestion method, a trade-off must be made between usefulness and effectiveness. For startpagina the current CSE suggestion method provides very useful suggestions, but does not have a high coverage for queries that lead to startpagina results. The original query-flow graph based suggestion method has the highest coverage but provides less useful suggestions compared to the CSE. We suggest that startpagina implements this model to provide suggestions to their users if possible, and use the CSE otherwise. This might lead the user unwillingly to the SERP, but they are at least do not leave the website empty handed. As discussed in Section 5.4, there is also a decision to be made in the discussion about diversifying the query suggestions opposed to only showing the highest ranked suggestions. If the business makes a decision on this and decides to use the query-flow graph model in their production environment, we advise to at least perform A/B-tests to ensure that the decisions they made are the right ones for their users.

**MRQ:** Can we successfully apply the state-of-the-art query suggestion pipeline of a Web search environment to a more limited and specialized environment?  
In our research, we implemented and evaluated a pipeline that generates query suggestions based on historic data stored in query logs. We built on prior work, and investigated established approaches for search session splitting, reformulation type classification and the generation of query suggestions from query-flow and term-query graphs. We were able to successfully apply the results of the literature for the Dutch curated content search of startpagina. While search session splitting and reformulation classification yielded similar results to existing work, the use of query-flow and term-query graphs showed a number of differences:

- We can improve the usefulness score (u-score) of the suggestions when only considering particular types of reformulations, as expected. In our case, leaving out class P (parallel move) increases the effectiveness. Contrary to prior work, S (specialization) reformulation edges do not play a significant role.
- In contrast to the literature, which usually focuses on the u-score as effectiveness measure, we also need to take the coverage of a model into account.
Here, we observe a considerable degradation in coverage when only a subset of reformulations is considered; thus, there is a trade-off between accuracy and completeness. Depending on the optimization goal, different models would have to be chosen.

- The term-query graph model produces the highest $u$-scores but, surprisingly, also has the lowest coverage by far. We reason that this is caused by the limited tag space $T$ combined with the use of the term-query model specific random walk method.

### 6.2 Future work

The fact that more than half of all generated suggestions were deemed at least somewhat useful indicates the potential of the query suggestion approaches we experimented with. This invites us to do more research into this topic. We now implemented and evaluated the most basic models of query-flow graphs, which show the applicability and potential in a limited search environment. We are curious what the other, more complex models based on query-flow graphs (see section 2) can achieve in terms of usefulness and coverage.

What we did not encounter in the literature is the possibility to combine models. For example, it might be interesting to investigate a cross-over between the reformulation graph models and the term-graph model in a regular CSE environment. Higher $u$-scores are obtained using either one of the extensions, so the combination might further increase this performance gain. Such a setup will not work for the startpagina use case though, as we expect the coverage in such a limited environment to become very low.

Another method of combining models is a cascaded setup of various models. We are curious what the performance would be when using this strategy. For example, the cascade can first extract the suggestions from the CSE and filter out all the suggestions that are not in $T$. If there are less than five suggestions left, the suggestions from the term-graph model are extracted and added to the suggestions list. They are ranked lower than the filtered CSE suggestions. Again, if the top-5 suggestions are not complete yet, the original model should be used to generate suggestions and complete the top-5 if possible. This cascade uses the strengths of each model which might eliminate the trade-off between usefulness and coverage. However, we expect that obtaining the suggestions for various models increases the suggestion generating run-time as well as the generated network traffic and/or resource usage. This should be considered in researching a cascaded setup.

There are also elements in the pipeline itself which can be revisited in future work. For example, it is also interesting to see if we can improve the session splitting accuracy for Dutch query logs using more advanced methods such as semantic analysis. The reformulation type classifier might also benefit from such techniques as it is a similar task. Furthermore, we now used PageRank and Personalized PageRank to rank the query suggestions. There are many more random walk ranking algorithms available that might be better suited to model user behavior. It would be interesting to experiment with these algorithms.


Appendix A

Evaluation form

In this appendix, we show the Dutch introduction text that was given to the query suggestion evaluators (Figure A.1). It explains the goal of the evaluation task. We also show a full screenshot of the query suggestion evaluation form (Figure A.2).

![Figure A.1: Screenshot of the evaluation introduction text.](image-url)
Figure A.2: Screenshot of the evaluation form.