UNIVERSITÀ DI PISA
Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
Ingegneria dell'Informazione

Ph.D. Thesis

Query Log Mining to Enhance User Experience in Search Engines

Franco Maria Nardini

2011

Università di Pisa

Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
Ingegneria dell'Informazione

Ph.D. Thesis

Query Log Mining to Enhance User Experience in Search Engines

Candidate:

Franco Maria Nardini

Supervisors:

Prof. Eng. Luca Simoncini

Dr. Fabrizio Silvestri

2011

SSD ING-INF/05

Sommario

Il Web rappresenta la più grande collezione di documenti che l'uomo abbia mai costruito. La sua importanza e le sue dimensioni crescono giorno dopo giorno. I motori di ricerca Web sono lo strumento attraverso il quale gli utenti cercano nel Web le informazioni di cui necessitano. Il servizio tipicamente offerto da un motore di ricerca prevede che, a fronte dell'invio di una query testuale che rappresenta il bisogno informativo dell'utente, sia restituita una lista di documenti considerati altamente rilevanti per la query inviata. Tutte le interazioni che gli utenti svolgono con un motore di ricerca Web sono registrate in archivi chiamati *query logs*. Con il nome *query log mining* si fa riferimento ad un insieme di tecniche che hanno lo scopo di estrarre "conoscenza" dai query log dei motori di ricerca Web che sia successivamente fruibile per altri scopi. Questa conoscenza è, infatti, il primo strumento di miglioramento del motore di ricerca stesso e, quindi, dell'esperienza di ricerca degli utenti. In accordo con questa visione, in questa tesi di dottorato si studia il fenomeno dell'invecchiamento dei modelli di conoscenza derivati dai query log e si individuano soluzioni atte a risolvere questo fenomeno. Inoltre, si propongono nuovi algoritmi di raccomandazione di query che, mantenendo i modelli aggiornati evitano l'invecchiamento della base di conoscenza associata. Un terzo contributo consiste nello studio di un algoritmo efficiente per la raccomandazione di query che sia particolarmente efficace per la produzione di suggerimenti per query rare. Infine, viene studiato il problema della diversificazione dei risultati dei motori di ricerca. In particolare, viene definita una metodologia basata su conoscenza estratta dai query log per identificare quando e come i risultati dei motori di ricerca devono essere diversificati. A questo proposito viene inoltre proposto un algoritmo efficiente per diversificare risultati di motori di ricerca Web.

Abstract

The Web is the biggest repository of documents humans have ever built. Even more, it is increasingly growing in size every day. Users rely on Web search engines (WSEs) for finding information on the Web. By submitting a textual query expressing their information need, WSE users obtain a list of documents that are highly relevant to the query. Moreover, WSEs tend to store such huge amount of users activities in *query logs*. Query log mining is the set of techniques aiming at extracting valuable knowledge from query logs. This knowledge represents one of the most used ways of enhancing the users search experience. According to this vision, in this thesis we firstly prove that the knowledge extracted from query logs suffer aging effects and we thus propose a solution to this phenomenon. Secondly, we propose new algorithms for query recommendation that overcome the aging problem. Moreover, we study new query recommendation techniques for efficiently producing recommendations for rare queries. Finally, we study the problem of diversifying Web search engine results. We define a methodology based on the knowledge derived from query logs for detecting *when* and *how* query results need to be diversified and we develop an efficient algorithm for diversifying search results.

Acknowledgments

I strongly desire to thank a long list of people for helping me in achieving this result. Firstly, I would like to thank my supervisors Fabrizio Silvestri and Luca Simoncini. I am particularly grateful to Fabrizio Silvestri whose advises started from my M.Sc. thesis and continued during my Ph.D. helping me in finding interesting problems and their solutions, always encouraging me to ask the best to myself.

I also would like to thank all my co-authors with which I share part of the results of this thesis: Ranieri Baraglia, Daniele Broccolo, Gabriele Capannini, Lorenzo Marcon, Raffaele Perego from ISTI-CNR, Ophir Frieder from Georgetown University and Debora Donato, Carlos Castillo from Yahoo! Research Labs in Barcelona.

I am also very grateful to the colleagues of the High Performance Computing Laboratory of the ISTI-CNR in Pisa for the precious ideas, discussions, tips, and moments we had since I was a Master student there.

I can not forget my colleagues with which I shared the room C-64 at ISTI-CNR: Gabriele Capannini, Gabriele Tolomei and Diego Ceccarelli for the wonderful "life" experiences spent in our room and all around the world.

I also would like to thank my family and all my friends. The achievement of this important result would not have been possible without the continuous support of them, which always encouraged me in any moment, and helped me whenever I needed it. A warm thank to a special woman, Serena, whose support during all these years was very precious, and with which I have shared all the pains and successes I have been living so far.

A Fosco ...
... luce d'un alba d'Ottobre.

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

The Web is the biggest repository of information that humans have ever built. It grows very quickly in size and importance every day. These unique characteristics carry many new challenges for Web researchers, which include high data dimensionality, highly volatile and constantly evolving contents. For these reasons it has become increasingly necessary to create new and improved approaches to traditional data mining techniques that can be suitably applied to the Web. The idea of "automatically identifying interesting and valuable information" has become a very relevant problem when processing large quantities of data.

In this thesis we focus our efforts on analyzing and extracting valuable knowledge from the behavior of Web search engine users. As we will discuss throughout this work, queries are crucial to understand how users interact with search engines. Much of this information is provided implicitly by users and recorded in search engine *query logs*. Implicit user feedback provide a unique insight into actual user needs on the Web. The intuition is that queries and their clicked results implicitly reveal the opinion of users about specific documents, i.e. they constitute a form of the so called *wisdom of the crowds* [14].

1.1 Contributions of the Thesis

We are going to illustrate four new contributions in two important fields of Web information retrieval and query log mining: query recommendation and Web search engine results diversification.

The Effects of Time on Query Flow Graph-based Models for Query Suggestion

This chapter based on [23, 24] presents a study of the effects of time on recommendations generated using *Query Flow Graphs* [35] (QFGs). These models aggregate information contained in a query log by providing a markov-chain representation of the query reformulation process followed by multiple users. We show how to extend QFG-based recommendation models to evolving data. Furthermore, we

show that the interests of search engine users change over time and new topics may become popular, while other that focused for some time the attention of the crowds can suddenly loose importance. The knowledge extracted from query logs can thus suffer an aging effect, and the models used for recommendations rapidly become unable to generate useful and interesting suggestions. We show how to overcome the time-expensiveness problem of building new fresh QFG from scratch by introducing an *incremental* algorithm for updating an existing QFG. The solution proposed allows the recommendation model to be kept always updated by incrementally adding fresh knowledge and deleting the aged one.

Incremental Algorithms for Effective and Efficient Query Recommendation

In Chapter 4 we prove that the knowledge extracted from historical usage data can suffer an aging effect. Starting from this result, this chapter based on [41] now presents a study of the effects of incremental model updates on the effectiveness of two query recommendation algorithms. We introduce a new class of query recommender algorithms that update *incrementally* the model on which recommendations are drawn. Starting from two state-of-the-art algorithms, we design two new query recommender systems that continuously update their models as queries are issued. The two incremental algorithms differ from their static counterparts by the way in which they manage and use data to build the model. In addition, we propose an automatic evaluation mechanism based on four new metrics to assess the effectiveness of query recommendation algorithms. The experimental evaluation conducted by using a large real-world query log shows that the incremental update strategy for the recommendation model yields better results for both coverage and effectiveness due to the "fresh" data that are added to the recommendation models. Furthermore, this improved effectiveness is accomplished without compromising the efficiency of the query suggestion process.

Generating Suggestions for Queries in the Long Tail with an Inverted Index

This chapter based on [42] presents a very efficient solution for generating effective suggestions to Web search engine users based on the model of *Search Shortcut* [22]. Our original formulation of the problem allows the query suggestion generation phase to be re-conducted to the processing of a full-text query over an inverted index. The current query issued by the user is matched over the inverted index, and final queries of the most similar satisfactory sessions are efficiently selected to be proposed to the user as query shortcuts. The way a satisfactory session is represented as a virtual document, and the IR-based technique exploited, allows our technique to generate in many cases effective suggestions even to rare or not previously seen queries. An additional contribution regards a new evaluation methodology used, based on a publicly-available test collection provided by a highly reputed organization such as the NIST. The proposed methodology is objective and very general, and it would grant researchers the possibility of measuring

the performance of their solution under exactly the same conditions, with the same dataset and the same evaluation criterium. On the basis of the above evaluation methods conducted by means of a user-study, and by using an automatic evaluation approach based on a previously defined metric we show that the proposed method remarkably outperforms its competitors in all the tests conducted.

Efficient Diversification of Web Search Results

This chapter based on [51, 50] presents a general framework for query result diversification comprising: i) an efficient and effective methodology, based on state-of-the-art query recommendation algorithms, to detect ambiguous queries that would benefit from diversification, and to devise all the possible common specializations to be included in the diversified list of results along with their probability distribution, ii) OptSelect: a new diversification algorithm which re-ranks the original results list on the basis of the mined specializations, iii) a Web search architecture based on additive Machine Learned Ranking (MLR) systems extended with a new module computing the diversity score of each retrieved document. A novel formulation of the query result diversification problem has been proposed and motivated. It allows the diversification problem to be modeled as a maximization problem. The approach is evaluated by using the metrics and the datasets provided for the TREC 2009 Web Track's Diversity Task. Our experimental results show that our approach is both efficient and effective. In terms of efficiency, our approach performs two orders of magnitude faster than its competitors and it remarkably outperforms its competitors in all the tests.

1.2 Outline

This thesis is organized as follows: Chapter 2 presents an overview of Web search engines, the main components constituting them and their interactions, Chapter 3 presents a survey of the major state-of-the-art contributions concerning the extraction and the use of valuable knowledge extracted from Web search engines' query logs. We particularly focus on discussing statistical properties of different query logs and on highlighting the main techniques aiming at enhancing the user experience and the effectiveness in WSEs. Chapter 4 is devoted to the analysis of the effects of time on the recommendations produced by a state-of-the-art query suggestion techniques called Query Flow Graph. In Chapter 5 we introduce new incremental query recommender techniques capable of taking the recommender model updated without the need of rebuilding it from scratch every a fixed period of time. In Chapter 6 an efficient and effective technique for producing query recommendation is introduced and evaluated. Chapter 7 presents a general framework for efficient query result diversification based on knowledge extracted from query logs. Finally, in Chapter 8 we present some conclusions and discuss future work proposals.

2

Web Search Engines

The World Wide Web, (WWW) and commonly known as the Web, is a system of interlinked hypertext documents accessed via the Internet. It has been proposed in 1989 by Tim Berners-Lee. Its implementation follows a client–server model. Users relies on a program (called the *Web browser*) to connect to a remote machine (the *Web server*) where the data are stored. Web browsers work by sending requests to remote servers for information and then interpreting the returned documents written in HTML and laying out the text and graphics on the user's computer screen on the client side. The Web relies on the structure of its hypertext documents.

In the latest years the Web is enormously increased in size and importance. With billions of pages, users could spend a lot of time surfing the Web, following links from one page to another. So, where does an user start? Searching the Web requires skills, luck and a little bit of art.

Web Search Engines (WSEs) are the primary way users access the contents of the Web. By using a WSE, users are able to look for some information they might need either for work or for leisure: news about the latest football match, or about the last confidence vote of the Parliament.

Search engine *directories*, such as Yahoo![1], are good at identifying general information. Like a card catalog in a library, they classify websites into categories, such as accounting firms, English universities and natural history museums. The results of a user search will be a list of websites related to the user's search terms.

What if an user want specific information, such as "biographical information about Leonardo da Vinci?". Web indexes are the right things to use. By querying such indexes, users obtain a ranked list of Web documents that are highly related with their information needs.

Web search engines are part of a broader class of software systems, namely Information Retrieval (IR) Systems. Basically, IR systems were born in the early 1960s due to two major application needs: i) the need to allow searching through

[1] `http://www.yahoo.com/`

digital libraries, and ii) the need for computer users to search through the data they were collecting in their own digital repositories.

An IR system is basically a software whose main purpose is to return a list of documents in response to a user query. This description makes IR systems similar to DB systems. Indeed, the most important difference between DB and IR systems is that DB systems return objects that exactly match the user query, whereas IR systems have to cope with natural language that makes it simply impossible for an IR system to return perfect matches. As an example: what does *meta* refer to? A meta character? The meta key in computer keyboards? Every single query may mean different things to different users. Furthermore, polysemy also comes into play. The spanish meaning of the word *meta* is *goal*. A Web search engine is thus an IR system on a very large scale.

The first systems similar to modern web search engines started to operate around 1994. World Wide Web Worm (WWWW) [107] created by Oliver McBryan at the University of Colorado, and the AliWeb search engine [97] created by Martijn Koster in 1994, are the two most famous examples. Since then many examples of such systems have been around the Web: AltaVista, Excite, Lycos, Yahoo!, Google, ASK, MSN. Nowadays, searching is considered one of the most useful application on the Web.

In a paper overviewing the challenges in modern Web search engines' design, Baeza-Yates *et al.* [10] state:

> The main challenge is hence to design large-scale distributed systems that satisfy the user expectations, in which queries use resources efficiently, thereby reducing the cost per query.

Therefore, the two key performance indicators in this kind of applications, in order, are: (i) the quality of returned results (e.g. handle quality diversity and fight spam), and (ii) the speed with which results are returned.

2.1 Architecture of a Web Search Engine

A search engine is one of the most complicated software a company may develop. Consisting of tens of interdependent modules, it represents a big challenge in today's computer engineering world. Many papers and books sketch the architecture of web search engines. For example Barroso *et al.* [25] present the architecture of Google as it was in 2003. Other search engines are believed to have similar architectures. A Web search engine consists in three major applications [40, 103]: *crawling*, *indexing*, and *query processing*. Figure 2.1 shows the way the three applications interact and how the main modules of a web search engine are connected.

Web search engines get their data from different sources: the Web, image and video repositories (e.g. Flickr, or YouTube), etc. Crawling is the process responsible for finding new or modified pages on the Web and is made by several software

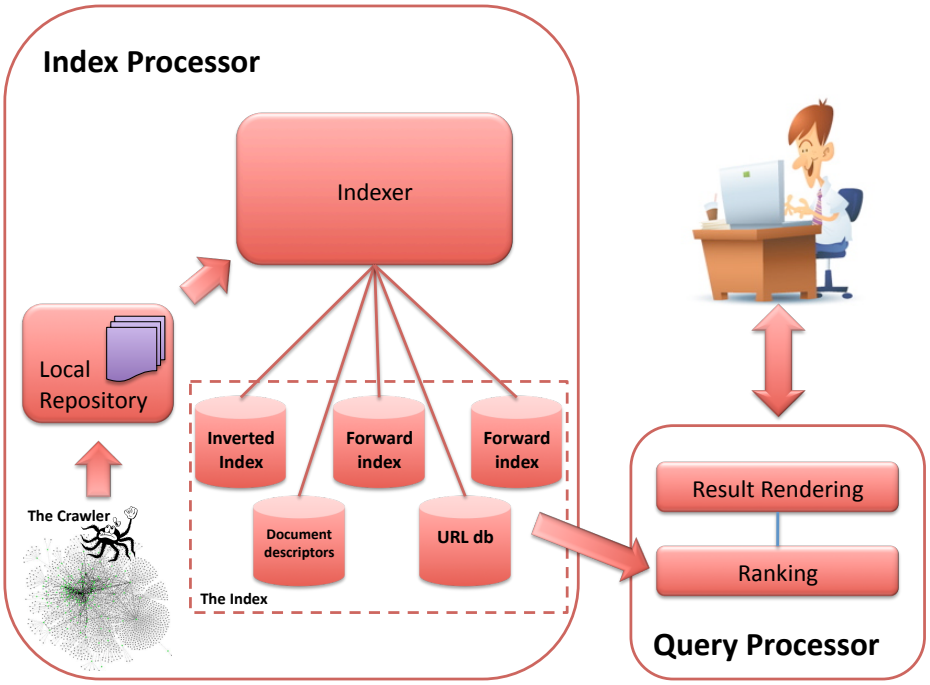


Fig. 2.1. The typical structure of a Web search engine. From [141].

agents called *crawlers* or *spiders*. In general, a crawler starts from a list of URLs, called *seeds*; then for each page, copies the page into the repository. Furthermore, the crawler fetches all URLs in the page and adds them to the list of the URLs to visit, called the *crawl frontier*. In particular, a crawler scours through hypertext pages searching for new documents, and detecting stale, or updated content. Crawlers store the data into a repository of content (also known as Web document cache), and structure (the graph representing how Web pages are interconnected). The latter being used, mainly, as a feature for computing static document rank scores (e.g. PageRank [115], or HITS [94]). In modern Web search engines, crawlers continuously run and download pages from the Web updating incrementally the content of the document cache.

The textual (i.e., hyper-textual) content is indexed to allow fast retrieval operations (i.e., query requests). The index (built by the Indexer) usually comprises of several different archives storing different facets of the index. The format of each archive is designed for enabling a fast retrieval of information needed to resolve queries. Indexes to support such text-based retrieval can be implemented using any of the access methods traditionally used to search over classical text document collections. Examples include suffix arrays, inverted indexes or inverted files, and signature files. In Web domain, *inverted indexes* are the index structure used.

An inverted index is made by a *dictionary* D of terms. In the following, let D be the data structure, and $V = \{t_1, t_2, \cdots, t_m\}$ the *vocabulary* i.e., the set of m terms of the whole document collection. For each term, we have a list that records in which documents the term occurs. This list is called *posting list* and its elements (*postings*) contain the IDs of the documents containing the term (and often the position of the match in the document).

Usually in real systems the design is tailored to distribute requests through query servers to avoid peaking server response time [25]. In real-world search engines, the index is distributed among a set of query servers coordinated by a broker. Figure 2.2 shows the interactions taking place among query servers and the broker. The broker, accepts a query an user and distributes it to the set of query servers. The index servers retrieve relevant documents, compute scores, rank results and return them back to the broker which renders the result page and sends it to the user. The broker is usually the place where the activities of users (queries, clicked results, etc.) are stored in files called *query logs*. A module dedicated to analyze past queries is also usually available within the architecture components.

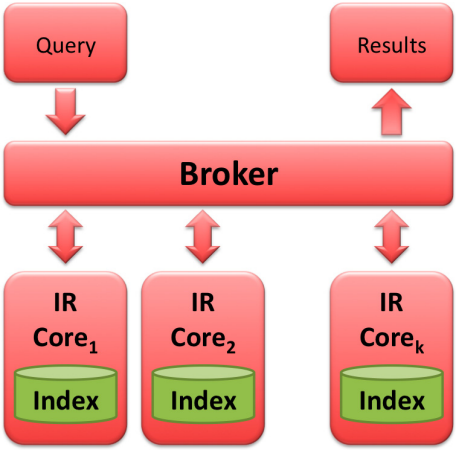


Fig. 2.2. The typical structure of a distributed web search engine. From [141].

Searching is the goal of a Web search engine. When a user enters a query, the user's browser builds a URL (for example `http://www.google.com/search?q=franco+maria+nardini`). The browser, then, looks up on a DNS directory for mapping the URL main site address (i.e., `www.google.com`) into a particular IP address corresponding to a particular data-center hosting a replica of the entire search system. The mapping strategy is done accordingly to different objectives such as: availability, geographical proximity, load and capacity. The browser, then, sends an HTTP request to the selected data-center, and thereafter, the query processing is entirely local to that center. The typical interaction between a user and a WSE

thus starts with the formulation of a query q, representing the user's *information need*. Note that the information need is different from the query: the first is the topic about which the user desires to know more, while the second is what the user conveys to the computer in an attempt to communicate the information need. A query consists of a list of r terms

$$q = t_1, t_2, \ldots, t_r$$

Once the user has submitted her query q, document indexes are accessed to retrieve a single, uncategorized list with the most k relevant items appearing first

$$search(q) = \{d_1, d_2, \cdots, d_k\}$$

where k usually is ten.

Sorting documents by relevance requires computing for each document a *relevance score* with respect to the query. Formally, each relevance of each document d_i is evaluated through a scoring function

$$score(q, d_i) = s_i \qquad s_i \in \mathbb{R}^+ \cup \{0\}$$

that returns a score s_i. The highest is the score, the highest is the relevance of the document d_i for the query q.

There are several methods for computing scoring function. A popular scoring function is *Okapi BM25* [128], based on a *bag of words model*: the rank of a document is given by the query terms appearing into it, without taking into consideration the relationships between the query terms within the documents.

After documents are sorted by relevance, the top-k results are returned in the form of an HTML page to the user.

3

Query Log Mining

The uncertainty in users' intent is a key problem in Web search engines and, differently from smaller scale IR systems, Web IR systems can rely on the availability of a huge amount of usage information contained within past queries to solve it. Previously submitted queries represent, in fact, a very important mean for enhancing effectiveness and efficiency of Web search systems.

Query log mining is concerned with all the techniques aiming at discovering interesting patterns from query logs of Web search engines with the purpose of enhancing an online service provided through the Web. It can be seen as a branch of the more general Web Analytics [80] scientific discipline. According to the Web Analytics Association,

> "Web Analytics is the measurement, collection, analysis and reporting of Internet data for the purposes of understanding and optimizing Web usage" [8].

It can be also seen as a special type of Web usage mining [151]. Web usage mining, in fact, refers to the discovery of user access patterns from Web usage logs. Furthermore, query log mining is not only concerned with the search service (from which queries usually come from) but also with more general services like, for instance, search-based advertisement, or web marketing in general [81].

Query logs keep track of information regarding interaction between users and the search engine. They record the queries issued to a search engine and also a lot of additional information such as the user submitting the query, the pages viewed and clicked in the result set, the ranking of each result, the exact time at which a particular action was done, etc. In general, a query log is comprised by a large number of records $\langle q_i, u_i, t_i, V_i, C_i \rangle$ where for each submitted query q_i, the following information is recorded: i) the anonymized identifier of the user u_i, ii) the timestamp t_i, iii) the set V_i of documents returned by the WSE, and iv) the set C_i of documents clicked by u_i.

From query log information it is possible to derive *Search Sessions*, sets of user actions recorded in a limited period of time. The concept can be further refined into: i) *Physical Sessions*, ii) *Logical Sessions*, and iii) *Supersessions*.

Physical Sessions: a physical session is defined as the sequence of queries issued by the same user before a predefined period of inactivity. A typical timeout threshold used in web log analysis is $t_0 = 30$ minutes. [118, 155].

Logical Sessions: a logical session [16] or *chain* [122] is a topically coherent sequence of queries. A logical session is not strictly related to timeout constraints but collects all the queries that are motivated by the same information need (i.e., planning an holiday in a foreign country, gathering information about a car to buy and so on). A physical session can contain one or more logical session. Jones *et al.* [92] introduced the concepts of *mission* and *goal* to consider coherent information needs at different level of granularity, being a goal a sub-task of a mission (i.e., booking the flight is one of the goal in the more general mission of organizing an holiday).

Supersessions: we refer to the sequence of all queries of a user in the query log, ordered by timestamp, as a supersession. Thus, a supersession is a concatenation of sessions.

Sessions are, thus, sequences of queries submitted by the same user in the same period of time. This data can be used to devise typical query patterns, used to enable advanced query processing techniques. Click-through data (representing a sort of implicit relevance feedback information) is another piece of information that is generally mined by search engines. In particular, every single kind of user action (also, for instance, the action of not clicking on a query result) can be exploited to derive aggregate statistics which are very useful for the optimization of search engine effectiveness. How query logs interact with search engines has been studied in many papers. Good starting point references are [141, 9, 135].

An important key issue in query log mining is the pre-processing of logs in order to produce a good basis of data to be mined. An important step in usage analysis is thus the *data preparation*. This step includes: data cleaning, session identification, merging logs from several applications and removing requests for robots. This techniques aims to remove irrelevant items, so that the resulting associations and statistics reflects accurately the interactions of users with the search engine.

Very few query logs have been released to the public community in the last years due to their commercial importance and to privacy issues. Starting from 1997 the query logs that have been released to the public are: Excite (1997), AltaVista (1998–1999), AOL (2003–2004), AOL (2006), MSN (2006). Table 3.1 resumes the most important features of the query logs that have been examined in the latest years.

The most famous query log is undoubtedly AOL. The AOL data-set contains about 20 million queries issued by about 650,000 different users, submitted to

Query Log Name	Public	Period	# Queries	# Sessions	# Users
Excite (1997)	Y	Sep 1997	1,025,908	211,063	∼410,360
Excite Small (1997)	Y	Sep 1997	51,473	–	∼18,113
Altavista	N	Aug 2, 1998 Sep 13, 1998	993,208,159	285,474,117	–
Excite (1999)	Y	Dec 1999	1,025,910	325,711	∼540,000
Excite (2001)	Y	May 2001	1,025,910	262,025	∼446,000
Altavista (public)	Y	Sep 2001	7,175,648	–	–
Tiscali	N	Apr 2002	3,278,211	–	–
TodoBR	Y	Jan 2003 Oct 2003	22,589,568	–	–
TodoCL	N	May 2003 Nov 2003	–	–	–
AOL (big)	N	Dec 26, 2003 Jan 01, 2004	∼100,000,000	–	∼50,000,000
Yahoo!	N	Nov 2005 Nov 2006	–	–	–
AOL (small)	Y	Mar 1, 2006 May 31, 2006	∼20,000,000	–	∼650,000
Microsoft RFP 2006	Y	Spring 2006 (one month)	∼15,000,000	–	–

Table 3.1. Features of the most important query logs that have been studied in the latest years. The dash sign (–) means that the feature in the relative column was non-disclosed.

the AOL search portal over a period of three months from 1st March, 2006 to 31st May, 2006. After the controversial discussion related to users' privacy issues followed to its initial public delivery, AOL has withdrawn the query log from their servers and is not offering it for download anymore.

```
507  kbb.com            2006-03-01 16:45:19  1   http://www.kbb.com
507  kbb.com            2006-03-01 16:55:46  1   http://www.kbb.com
507  autotrader         2006-03-02 14:48:05
507  ebay               2006-03-05 10:50:35
507  ebay               2006-03-05 10:50:52
507  ebay               2006-03-05 10:51:24
507  ebay               2006-03-05 10:52:04
507  ebay               2006-03-05 10:52:36  69  http://antiques.ebay.com
507  ebay               2006-03-05 10:58:00
507  ebay               2006-03-05 10:58:21
507  ebay electronics   2006-03-05 10:59:26  5   http://www.internetretailer.com
507  ebay electronics   2006-03-05 11:00:21  20  http://www.amazon.com
507  ebay electronics   2006-03-05 11:00:21  22  http://gizmodo.com
507  ebay electronics   2006-03-05 11:00:21  22  http://gizmodo.com
507  ebay electronics   2006-03-05 11:18:56
507  ebay electronics   2006-03-05 11:20:59
507  ebay electronics   2006-03-05 11:21:53  66  http://portals.ebay.com
507  ebay electronics   2006-03-05 11:25:35
```

Fig. 3.1. An example of the AOL query log [116].

Figure 3.1 shows a fragment of the AOL query log. Each row of this query log consist of records collecting five fields: i) the ID referring to the user issuing the query, ii) the issued query, iii) the time the query was issued to the search engine, iv) the position of the clicked result in the results page, and v) the host of the clicked document.

Topics covered by queries contained in search engines are the most disparate. Figure 3.2 highlights the 250 most frequent terms in the AOL query log by means of a tag cloud. The dimension of each term in the tag cloud is directly related to its frequency in the log. The more a term is big in the tag cloud, the more the term is frequent in the log. As an example,"google" and "free" are the two most frequent terms in the log. Other very frequent words are: "yahoo", "new", "county", "pictures", "http".

Fig. 3.2. A tag cloud of the 250 most frequent words in the AOL query log [116]. Picture has been generated using wordle.net. From [141].

Furthermore, the queries submitted by users to search engines and thus contained in query logs are sometimes incredible. Just to give an idea, from the AOL query log user #2386968 submitted the query "why is my husband so talkative with my female friends". Another funny example is the query submitted by the user #427326 looking for "where is my computer?".

3.1 A Characterization of Web Search Engine Queries

In this section we aim at presenting a characterization of query logs by showing how queries are distributed over time. In particular, we analyze topic-popularity, term-popularity, differences with respect to the past, variations of topics during day hours, etc.

Some important efforts have been spent in the past to study how people interacted with small scale IR systems[1] [79, 67, 139, 146]. The nature of query logs coming from large scale Web search engines is different with respect to small scale IR systems. As an example, Web search engine queries unlikely contain more than three terms while small IR systems, i.e. commonly used for digital libraries or legal document retrieval, receive queries with a number of query terms ranging from 7 to 15 depending on the experience of the users. Search operators (like quotes, "+", "-", etc.) are rarely used on the web.

The characteristics of query logs coming from some of the most popular Web search engines have been deeply studied [27, 28, 82, 83, 84, 86, 96, 106, 112, 114, 116, 147, 148, 169]. Typical statistics that can be drawn from query logs are: query popularity, term popularity, average query length, distance between repetitions of queries or terms.

The first contribution in analyzing query logs comes from Silverstein *et al.* [140]. Authors propose an exhaustive analysis by examining a large query log of the AltaVista search engine containing about a billion queries submitted in a period of 42 days. The study shows some interesting results including the analysis of the query sessions for each user, and the correlation among the terms of the queries. Similarly to other works, authors show that the majority of the users (in this case about 85%) visit the first page of results only. They also show that 77% of the users' sessions end up just after the first query. The query log analyzed contains a huge number of queries and account to 285 million users. Furthermore, the results shown in the paper are considered precise and general due to the high number of queries and users analyzed.

Jansen *et al.* [84] analyze a log consisting of 51,473 queries submitted by 18,113 Excite users. The query log is anonymized. As the information are completely decontextualized, no user profiling can be carried out on this query log. The log from which experiments are carried out is publicly available to scholars.

Lempel and Moran [100], and Fagni *et al.* [66] study the content of a publicly available AltaVista log. The log consist of 7,175,648 queries issued to AltaVista during the summer of 2001. No information referring the number of users logged is released. This second AltaVista log covers a time period almost three years after the first studies presented by Jansen *et al.* and Silverstein *et al.* Furthermore, the log is smaller than the first one. Indeed it represents a good picture of search engine users.

On average web search engines queries are quite short. The average length of a query in the Excite log (1998) is 2.35 terms. Furthermore, less than 4% of the queries contains more than 6 terms. In the case of the AltaVista log, the average query length is slightly greater: 2.55. These numbers are deeply different compared with classical IR systems where the length of a query ranges from 7 to 15 terms. A possible explanation of this phenomenon is that, for instance,

[1] The IR systems whose studies were referring do not directly deal with Web users.

the Web is a medium used by different people from different part of the world looking for disparate information, while past IR systems were instead manually used by professionals and librarian looking for very focused information thus trying to precisely formulate their information needs. That studies highlight that users querying search engines, i.e. web IR systems, are different from people that used *classical* IR systems.

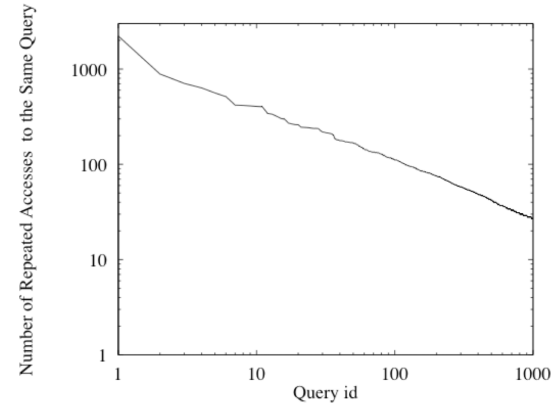
The distribution of query popularity, and term popularity have been shown to follow a power-law. This means that the number of occurrences y of a query or a term is proportional to $x^{-\alpha}$, where x is the popularity rank, and α is a real parameter measuring how popularity decreases against the rank. In a formula, $y = Kx^{-\alpha}$, where K is a real positive constant corresponding to the query with the highest popularity. Power-law distributions have the form of a straight line when plotted on a log-log scale.

Markatos [105] is the first to show that query popularity follows a power-law with an exponent $\alpha = 2.4$. He analyzes the Excite query log and plots the occurrence of the first 1,000 most popular queries. Figure 3.3 shows that the popularity follows the usual linear trend in a log-log scale. Furthermore, the plot reveals that the most popular query is submitted 2,219 times while the 1,000-th most popular query is submitted only 27 times. Later studies confirmed the previous power-law trend in two other query logs: i) AltaVista [100] and Yahoo! [15].

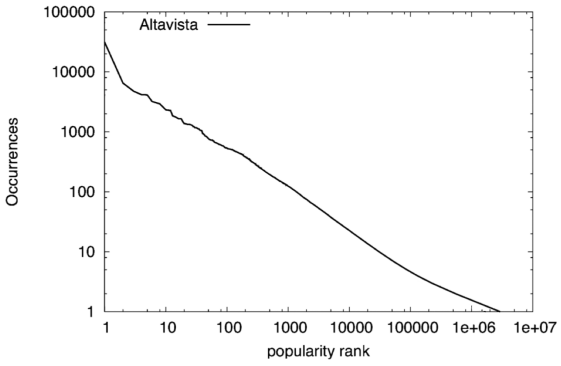
Tables in Figure 3.4 detail the top-20 queries for the Excite [105] and AltaVista [100] logs, respectively. Many queries in both logs refer to sex and sexually explicit topics while many others can be somewhat related to the same topics as well. Furthermore, there are some unexpected outcomes in query logs. For instance, the most frequent query in the Excite query log is the empty query. This request accounts for 5% of the queries. Authors in [84] give a possible explanation of this phenomenon. It could be due both i) to possible errors in typing queries in the search box or ii) to how Excite react to user actions. As an example, Excite result pages had a link pointing to a "More Like This" function that, if clicked, returned pages related to the ones selected. Excite count that behavior as an empty query thus raising the empty query count.

Many different topics can be found in query logs. It can be easily seen in Figure 3.4. A very first result in categorizing queries is [147]. Authors show the percentage of queries submitted for each topic to the Excite search engine in 1997. Categorizing queries into topics is not a trivial task. Recent papers showing techniques for assigning labels to each query [74, 137, 163, 30, 31, 45] adopts a set of multiple classifiers subsequently refining the classification phase.

Classification of the Excite queries made by Spink *et al.* in [147] shows that in no ways is pornography a major topic of web queries, even though the top ranked query terms may indicate this. Only one in about six web queries have been classified as about sex ("XXX"). Web users look interested on a wide range of different topics. Commerce, including travels, employment, and a number of



(a)



(b)

Fig. 3.3. Query popularity of the first 1,000 queries in the Excite [105] (a), and ii) AltaVista [100] (b) logs.

economic topics are also high in the list. Furthermore, about 10% of the queries are about health and the sciences.

Authors of [30, 27] show similar results on a different query log. The log is made up of billions of web queries constituting the total query traffic for six months of AOL. Categories are different, and results (in terms of category percentages breakdown) are quite different. The difference is due to the different period of time in which the analysis was conducted: while the Excite log refers to queries issued in 1997, the AOL log is younger as it consists of queries issued in 2003. Furthermore, porn queries fell considerably.

Terms are distributed according to a power-law as well (in particular a double-pareto log-normal distribution). In fact, the curve of term distribution fall sharply denoting that the most frequent terms are much more frequent that the rest of

query	freq.
Empty Query	2,586
sex	229
chat	58
lucky number generator	56
p****	55
porno	55
b****y	55
nude beaches	52
playboy	46
bondage	46
porn	45
rain forest restaurant	40
f****ing	40
crossdressing	39
crystal methamphetamine	36
consumer reports	35
xxx	34
nude tanya harding	33
music	33
sneaker stories	32

(a)

query	freq.
christmas photos	31,554
lyrics	15,818
cracks	12,670
google	12,210
gay	10,945
harry potter	7,933
wallpapers	7,848
pornografia	6,893
“yahoo com”	6,753
juegos	6,559
lingerie	6,078
symbios logic 53c400a	5,701
letras de canciones	5,518
humor	5,400
pictures	5,293
preteen	5,137
hypnosis	4,556
cpc view registration key	4,553
sex stories	4,521
cd cover	4,267

(b)

Fig. 3.4. Query terms of the first 20 queries in the Excite [105] (a), and AltaVista [100] (b) logs.

the terms. Figure 3.6 shows log-log plots of the term popularity distribution in the case of two query logs: Excite [105], and AltaVista [100].

An interesting statistics obtained from query logs is how terms co-occur. In [150], a follow-up of the work presented in [84], Spink *et al.* present the first fifty most frequently co-occurrent terms. Figure 3.2 shows how terms co-occur in queries without reflecting topic popularity. The majority of term pairs concern non-XXX topics while in the same analysis they found that XXX queries are highly represented. This highlight that, for some topics, people use more terms to search for precise information, while for other topics the same user need can be satisfied by short queries.

Queries repeat themselves. Since many queries are seen only a few times, one could expect that in the majority of the cases the distance between subsequent submissions of the same query would be very large. Figure 3.7 shows the distance, in terms of queries, with which queries are submitted again.

Differently from what is expected, the majority of queries have distances that are less than 1000 queries. A possible explanation is the *bursty* nature of query logs: a large number of people start looking for a topic at the same time. This observation is very important: the bursty nature of queries is a feature that is extensively used in many techniques for enhancing both effectiveness and efficiency of web search engines.

Topic	Percentage
Entertainment or recreation	19.9%
Sex and pornography	16.8%
Commerce, travel, employment, or economy	13.3%
Computers or Internet	12.5%
Health or sciences	9.5%
People, places, or things	6.7%
Society, culture, ethnicity, or religion	5.7%
Education or humanities	5.6%
Performing or fine arts	5.4%
Non-English or unknown	4.1%
Government	3.4%

(a)

Topic	Percentage
Entertainment	13%
Shopping	13%
Porn	10%
Research & learn	9%
Computing	9%
Health	5%
Home	5%
Travel	5%
Games	5%
Personal & Finance	3%
Sports	3%
US Sites	3%
Holidays	1%
Other	16%

(b)

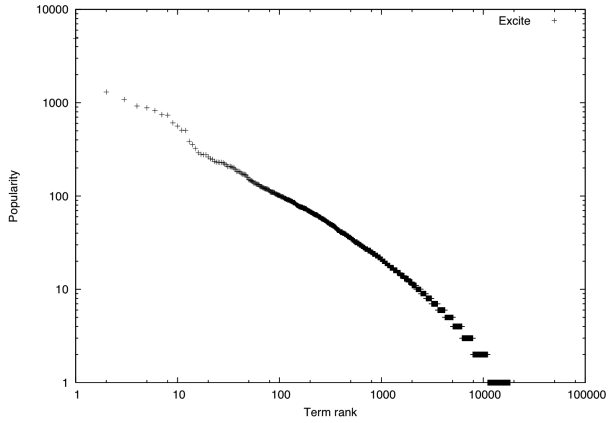
Fig. 3.5. Distribution of query samples across general topic categories for two different query logs: Excite [105] (a), and AOL [27] (b).

and-and	6,116	of-and	690	or-or	501	women-nude	382	sex-pics	295
of-the	1,901	pictures-of	637	sex-pictures	496	pics-nude	380	north-carolina	295
pics-free	1,098	how-to	627	nude-pictures	486	of-department	365	free-teen	293
university-of	1.018	and-the	614	for-sale	467	united-states	361	free-porn	290
new-york	903	free-pictures	637	and-not	456	of-history	332	and-nude	289
sex-free	886	high-school	571	and-sex	449	adult-free	331	and-pictures	286
the-in	809	xxx-free	569	the-to	446	of-in	327	for-the	284
real-estate	787	and-free	545	the-the	419	university-state	324	new-jersey	280
home-page	752	adult-sex	508	princess-diana	410	sex-nudes	312	of-free	273
free-nude	720	and-or	505	the-on	406	a-to	304	chat-rooms	267

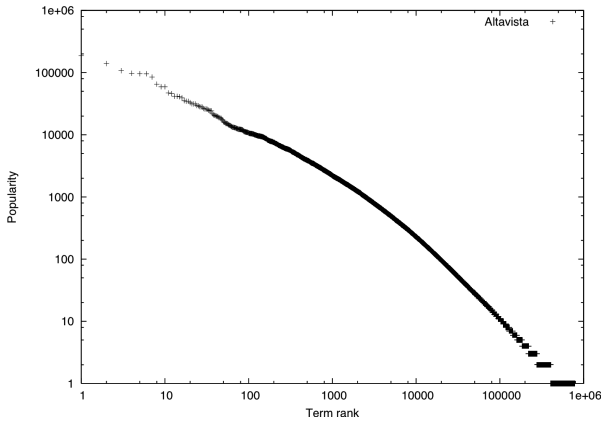
Table 3.2. List of the fifty most co-occurring terms ($term_1$–$term_2$, frequency) in the Excite log [150].

3.2 Time Analysis of the Query Log

Queries are issued on several different topics [116] depending also on the historical period [147]. Query logs can be analyzed at different levels of time granularity.



(a)



(b)

Fig. 3.6. Terms popularity of i) the first 1,000 queries in the Excite [105], and ii) AltaVista [100] logs.

On a daily granularity level of analysis, some of the topics are most popular in an hour than in another [27, 28].

Frequency of queries vary considerably during the day. Ozmutlu *et al.* [114] analyze query frequency against arrival time for the Excite query log in a time period ranging from 9AM to 5PM. Querying activity is higher during the first hours of the day than in the afternoon. There is a sharp decrease in the number of queries submitted going down from 679 at 9AM to 224 at 4PM. In particular, the number of queries at 4PM is about 30% of the queries that are usually submitted at 9AM. Note that these numbers are small if compared to the activity of today's web

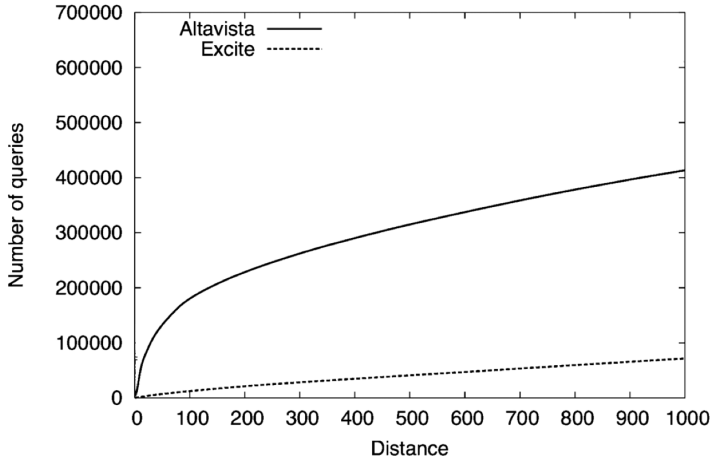


Fig. 3.7. Distances (in number of queries) between subsequent submissions of the same query for the AltaVista and Excite log.

search engines. Comparing these numbers with a similar statistic [116] performed in 2006, results are completely different. Figure 3.8 shows how frequencies are distributed within the hours of the day. At 9AM queries submitted are almost half of those submitted at 5PM.

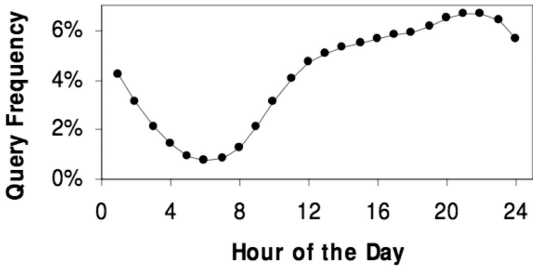


Fig. 3.8. Frequencies of query submitted to the AOL search engine during the day [116].

Spink *et al.* [147] show how time periods affects querying behavior of users. Table 3.3 shows how the querying behavior is not changed from a statistical point of view, in a period of four years. The mean number of terms per query is only slightly raised in 2001, while the number of terms per query, the mean queries per user, are basically unchanged in four years. Even if this study dates back to 2001, it is very likely that the results it presents are still valid today.

Obviously, the more penetrated a new technology is the more users become skilled with using it. From Table 3.4 it is clear that users querying for "People, Places or Things" was about 50% in 2002. Moreover, there is a clear rise of interest

Characteristic	1997	1999	2001
Mean terms per query	2.4	2.4	2.6
Terms per query			
1 term	26.3%	29.8%	26.9%
2 term	31.5%	33.8%	30.5%
3+ term	43.1%	36.4%	42.6%
Mean queries per user	2.5	1.9	2.3

Table 3.3. Comparative statistics for Excite Web queries [147].

from users for this category: in 1997 queries referring to "People, Places or Things" accounted for less than 7%. The 25% of users in 2002 query for "Commerce, Travel, Employment or Economy" and "Computers, Internet or Technology". This percentage shows an "up-and-down" trend, varying from a minimum of 25% and to a maximum of 35%. Furthermore, "Sex and Pornography" shows a falling trend: from 16.8% in 1997 to 3.3% in 2002.

Category	1997	1999	2001	2002
People, places, or things	6.7	20.3	19.7	49.3
Commerce, travel, employment, or economy	13.3	24.5	24.7	12.5
Computers or Internet	12.5	10.9	9.7	12.4
Health or sciences	9.5	7.8	7.5	7.5
Education or humanities	5.6	5.3	4.6	5.0
Entertainment or recreation	19.9	7.5	6.7	4.6
Sex and pornography	16.8	7.5	8.6	3.3
Society, culture, ethnicity, or religion	5.7	4.2	3.9	3.1
Government	3.4	1.6	2.0	1.6
Performing or fine arts	5.4	1.1	1.2	0.7
Non-English or unknown	4.1	9.3	11.4	0.0

Table 3.4. Comparison of categories breakdown (in %) for Excite Web queries (from 1997 to 2001), and Altavista (2002) [83].

Beitzel *et al.* [27] measure the relative popularity of different categories over the hours in a day. Figure 3.9 shows the percentage of total query volume broken-down to a selected group of category. Different topical categories are more or less popular at different times of the day. As an example, while "Personal Finance" popularity raises during the first hours of the morning between 7AM and 10AM, "Porn" is a category whose popularity raises during late-night until 6AM.

Figure 3.10 shows a different analysis on the same categories obtained by applying KL-divergence. The reason of applying this measure is to compare in a better way categories with a different relative level of popularity. The comparison is, in fact, affected by popularity shift. To overcome this issue, Beitzel *et al.* compute KL-divergence between the likelihood of receiving a query on any topic at a particular time and the likelihood of receiving a query in a particular category. KL-divergence measure a sort of *most surprising* category for a particular time

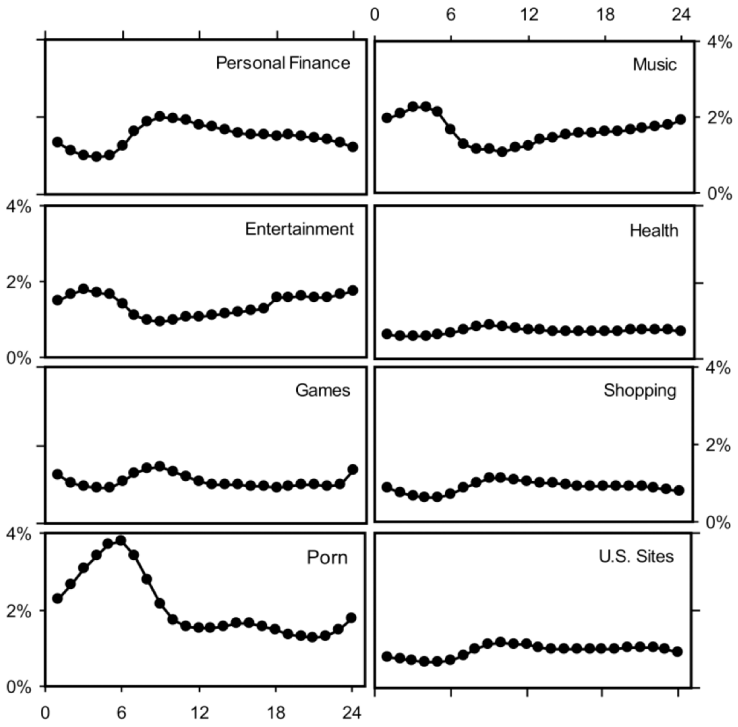


Fig. 3.9. Percentage of the query stream covered by selected categories over hours in a day [27].

of the day. Instead of measuring the popularity as the most numerous topic, the KL-divergence measures how popular is a query in terms of not being expected.

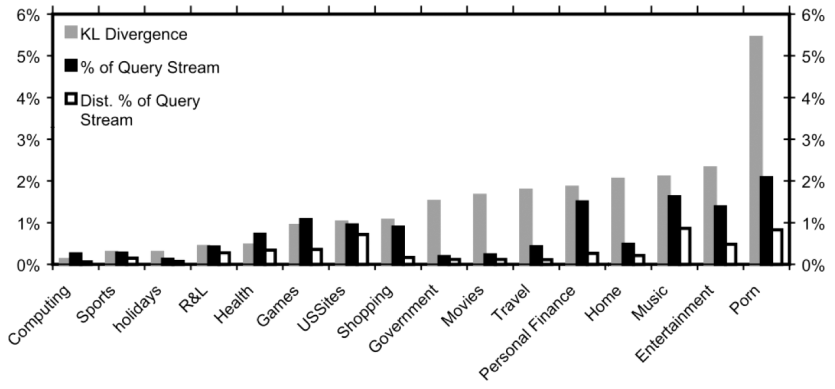


Fig. 3.10. Average percentage of the query stream coverage and KL-divergence for each category over hours in a day [27].

A more recent paper shows similar results on a MSN Web search engine query log [176]. Authors present some results that are not detailed with respect to topics, as presented in the previous paper, yet they do not disagree with the overall results presented in [27]. This analysis revealed that the top three categories in terms of popularity are "pornography", "entertainment", and "music". Beitzel *et al.* [27] reach the same conclusion by thoroughly discussing a more through analysis on weekly and monthly basis.

3.3 Search Sessions

After a first characterization of Web search engines' queries, here we focus on studying how users interact with search engine systems. What happen when a user has submitted a query and results are shown? How can be decided if a query has been correctly answered or if a user is satisfied by the search results? How people change queries if those have not produced satisfying users? The answers to these questions are very important in order to enhance the Web search engine performances.

In one of the first paper devoted to discovery user intent behind queries [43] Andrei Broder studies the goal a user wants to reach when submitting a query to a search engine. In the Broder's taxonomy a query can be either a *Navigational* query – were the immediate intent is to reach a particular destination (e.g. *Yahoo.com*, *America Airlines home page*, *Don Knuth's home page*); an *Informational* query – where the intent is to acquire some information assumed to be present on one or more web pages (e.g. *San Francisco* or *normocytic anemia*); a *Transactional* query – where the immediate intent is to perform some Web-mediated activity (e.g. *online music*, or *online flower delivery service*).

Table 3.5 shows the result of a survey presented to AltaVista users to try to determine their intents. Results in Table 3.5 are obtained by means of a series of questions presented to users through a pop-up window opening for some randomly chosen result pages. The survey obtained a response ratio of about 10% consisting of about 3,190 valid returns. The "query log analysis" column in Table 3.5 is obtained by selecting 1,000 queries at random and manually removing: i) non-English queries, and ii) sexually oriented queries. From the remaining set, the first 400 queries are inspected. Queries that are neither transactional, nor informational, are assumed to be informational in intent.

Results from both the survey, and manual inspection confirm what we argue in Section 3.1: in the majority of the cases users surf the web looking for places where to buy goods, or looking for particular sites they *already* know.

In order to evaluate the quality of search results it is interesting to look at how users interact with the search engine. For instance, it is interesting to extract and analyze user *search sessions* from query logs, and to derive *implicit measures* of quality explicitly tailored on search engine users.

Type	Surveyed	Estimated (from Query Log)
Navigational	24.5%	20%
Informational	∼ 39%	48%
Transactional	∼ 36%	30%

Table 3.5. Query classification on the basis of user survey [43].

Queries themselves are not always enough to determine the user intent. Furthermore, one of the key objectives of a search engine is to evaluate the quality of their results. Implicit measures that are available to log analysts are: the *click-through rate* – the number of clicks a query attract, *time-on-page* – the time spent on the result page, and *scrolling behavior* – how users interact with the page in terms of scrolling up and down; are all performance indicators search engines can use to evaluate their quality. How are the data recorded? Toolbars and user profiles surveyed directly from users are the main mean through which search engine companies record usage data diverse from those obtained by query logs.

A series of queries can be part of a single information seeking activity. Efforts have been spent on understanding the effects of request chains on the search engine side. The main goals of this analysis are to show how users interact with the search engine, and how they modify queries depending on what they obtain (in terms of results) from the search engine. Last but not least, efforts have been spent on understanding how users use multitasking and task switching on search sessions.

A first interesting result to highlight is how users interact with the search engine from a *page request* point of view. Many studies point out that users rarely visit result pages beyond the first one. Spink *et at.* analyze the Excite query log showing that about 78% of users do not go beyond the first page of results. Similar analysis on different query logs show the same behaviors [100, 66].

Jansen and Spink [83] show the percentage of single-query session in different query logs. The analysis is conducted on US search engines and it highlights that the complexity of interactions is increasing as indicated by longer sessions (i.e. users submitting more Web queries). In 2002, about 47% of AltaVista users submitted only one query, while in 1998 it was 77%.

An important analysis is conducted by Fagni *et al.* [66]. Authors estimate the probability of clicking the "next" button of a search engine result page. Figure 3.12 shows the behavior of the probability of clicking the "next" button. It increases as the number of result page increases. The rationale of this could be that users issuing informational queries usually go through a higher number of pages. Furthermore, half of the users on page 2 go to page 3, and about 60-70% of users on page 3 go to page 4.

In order to obtain the results they need, users often try to reformulate queries (refine or modify) during their search sessions. Lau and Horvitz [99] study this behavior by categorizing queries. They propose seven categories:

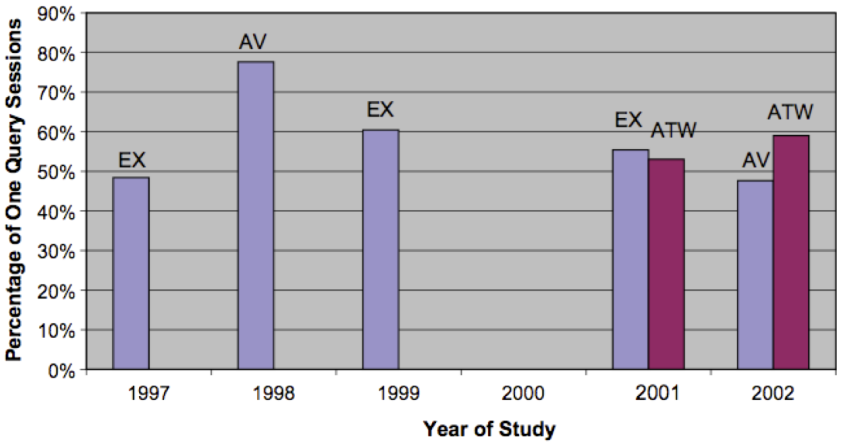


Fig. 3.11. Percentage of single query sessions [83].

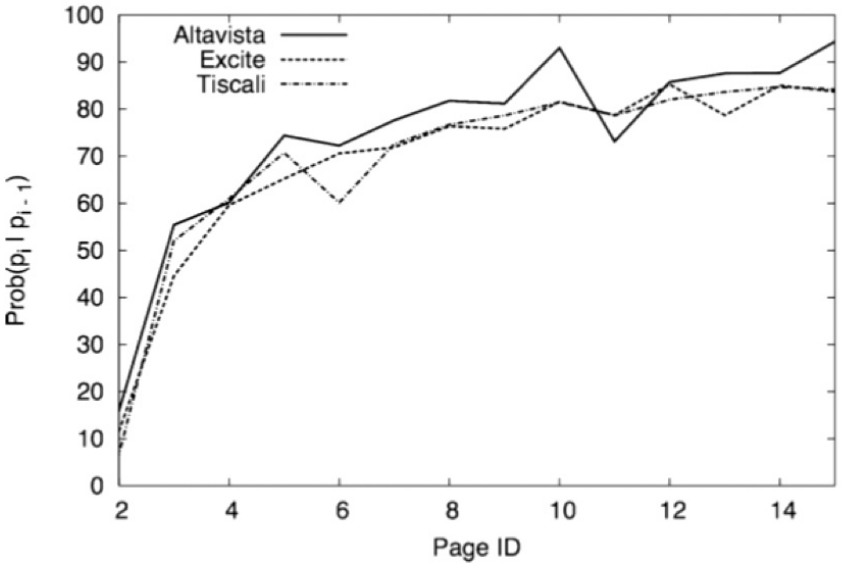


Fig. 3.12. Probability of pushing the next button for three query logs [66].

- *New*: A query for a topic not previously searched for by this user within the scope of the dataset (twenty-four hours);
- *Generalization*: A query on the same topic as the previous one, but seeking more general information than the previous one.
- *Specialization*: A query on the same topic as the previous one, but seeking more specific information than the previous one.
- *Reformulation*: A query on the same topic that can be viewed as neither a generalization nor a specialization, but a reformulation of the prior query.

- *Interruption*: A query on a topic searched on earlier by a user that has been interrupted by a search on another topic.
- *Request for additional results*: A request for another set of results on the same query from the search service.
- *Blank queries*: Log entries containing no query.

Authors apply the proposed categorization scheme within the Excite query log. Figure 3.13 shows the results obtained. In the majority of the cases most actions are either new queries or requests for additional informations. Furthermore, a large percentage of users (about 30%) issue a modification, (refinement, specification, or a reformulation) of a previously submitted query.

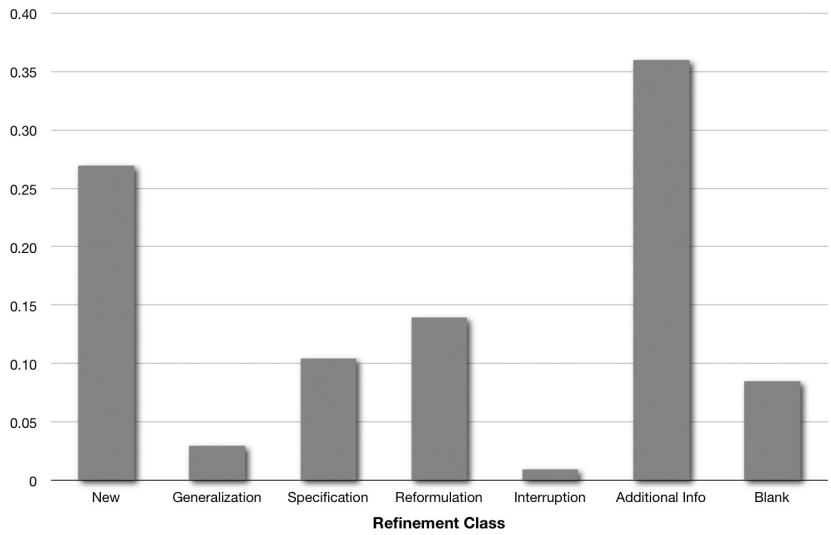


Fig. 3.13. Summary of the categorization of 4,960 queries analyzed in [99].

Jansen *et al.* [87, 88] examine 2,465,145 interactions from 534,507 users of Dogpile.com submitted on May 6, 2005 in order to investigate and identify the patterns of interaction between searchers and search engine during Web searching. They compare query reformulation patterns and investigate the type of query modifications and query modification transitions within sessions. Authors use manually-crafted rules to identify reformulation types (such as "adding one extra word means specializing the current query"). Authors identify three strong query reformulation transition patterns: between i) specialization and generalization, between ii) video and audio, and between iii) content change and system assistance. In addition, results show that Web and images content are the most popular media collections.

Boldi *et al.* [37, 33] build an accurate model for classifying user query reformulations into four broad classes (*generalization*, *specialization*, *parallel move* or *error*

correction) achieving 92% of the accuracy. Conceptually, the proposed taxonomy has two dimensions, depicted in Figure 3.14. One dimension is found along the generalization-specialization axis, and the other dimension along the dissimilarity axis. Moving left to right along the dissimilarity (horizontal) axis, authors find a continuous in which the syntactic and semantic gap between two queries gets larger and larger. Furthermore, moving to the top or to the bottom along the specificity (vertical) axis, authors find respectively reformulations towards more general or more specific queries. Authors apply the proposed model to automatically label two large query logs. Furthermore, authors study the resulting reformulation patterns, finding results consistent with previous studies done on smaller manually annotated datasets, and discovering new interesting patterns, including connections between reformulation types and topical categories. Finally, applying their findings to a third query log that is publicly available for research purposes, authors demonstrate that their reformulation classifier leads to improved recommendations in a query recommendation system.

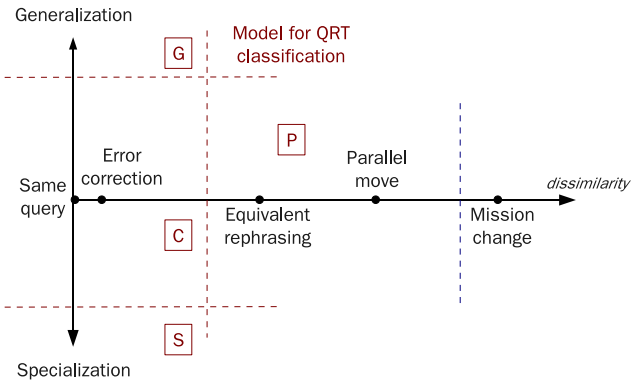


Fig. 3.14. Graphical depiction of transition types defined in [37].

A different type of analysis can be carried out by studying *Multitasking* and *Task Switching* in query sessions. Multitasking sessions are performed by users looking for information on multiple topics at the same time. Latest studies show that users tend to carry on multi-tasking queries. Ozmutlu *et al.* [113] show that in the 11.4% of the cases uses are pursuing multitasking sessions. This percentage increases to 31.8% in the case of users of another popular (at that time) search engine, AllTheWeb.com. In the same paper, the mean number of topic changes per session has been estimated to be around 2.2, and it raises to 3.2 when considering only multi-topic sessions.

Another interesting research field study methods to detect query re-submission, or information *re-retrieval* [155]. The behavior analyzed is how often users search for the same information he searched before. It is a quite common behavior nowa-

days to use search instead of bookmarking an interesting page. An example of this is a user searching for a conference home page by issuing the conference name as a query to a search engine. The user may be interested in looking for news about the conference after two months (for instance to read the call for papers, to check submission deadlines, to check the list of accepted papers, to book an hotel). Some important papers studying this behavior are [155, 154, 130].

Sanderson and Dumais [130] evaluate the re-finding behavior in web search engines. The dataset used by authors covers a shorter period of time, three months from April to June 2006, and contains approximately 3.3 million queries and 7.7 million search result clicks gathered from 324,000 unique users. An important thing authors discover is that repeated queries by the same user are almost the 50% of the total number of queries (1.68 million against 1.62 million unique queries). This result is: i) greater than the one presented in [155], ii) different from a previous result from Teevan *et al.* [154]. The conclusions are the same in all the cases: users do re-submit the same queries over and over and for this reason search engine designers should think solutions to take advantage of this phenomenon, for example, by designing interfaces able to present users resubmitting again the same query with a list of changes in the rank of the results with respect to those returned in answer to the same query but previously submitted.

Many other works deal with the identification of users' search sessions boundaries. Work on session identification can be classified into: i) *time-based*, ii) *content-based*, and iii) *mixed-heuristics*, which usually combine both i) and ii).

Time-based techniques have been extensively proposed for detecting meaningful search sessions due to their simplicity and ease of implementation. Indeed, these approaches are based on the assumption that time between adjacent issued queries is the predominant factor for determining a topic shift in user search activities. Roughly, if the time gap between two issued queries is lower than a certain threshold then they are also likely to be related.

Silverstein *et al.* [140] present a broad analysis of a very large query log data set collected by the AltaVista search engine and firstly define a concept of *session* as follows: two consecutive queries are part of the same session if they are issued at most within a 5-minutes time window. According to this definition, they find that the average number of queries per session in the data they analyzed was 2.02. He and Göker [76] use different timeouts to split user sessions of Excite query log, ranging from 1 to 50 minutes.

Radlinski and Joachims [122] observe that users often perform a sequence of queries with a similar information need, and they refer to those sequences of reformulated queries as *query chains*. Their paper presents a method for automatically detecting query chains in query and click-through logs using 30 minutes threshold for determining if two consecutive queries belong to the same search session.

Jansen and Spink [83] make a comparison of nine Web search engines transaction logs from the perspectives of *session length*, *query length*, *query complexity*,

and *content viewed*. Here, they provide another definition of session, i.e. *search episode*, describing it as the period of time occurring from the first to the last recorded timestamp on the WSE server from a particular user in a single day, so that session length might vary from less than a minute to a few hours. Moreover, using the same concept of search episode, Spink *et al.* [149] investigate also *multitasking* behaviors while users interacting with a WSE. Multitasking during Web searches involves the *seek-and-switch* process among several topics within a single user session. Again, a user session is defined to be the entire series of queries submitted by a user during one interaction with the WSE, so that session length might vary from less than a minute to a few hours. The results of this analysis performed on a AltaVista query log show that multitasking is a growing element in Web searching.

Finally, Richardson [126] shows the value of long-term WSE query logs with respect to short-term, i.e., within-session, query information. He claims that long-term query logs can be used to better understand the world where we live, showing that query effects are long-lasting. Basically, in his work Richardson does not look at term co-occurrences just within a search session that he agree to be a 30 minutes time-window, but rather across entire query histories.

Content-based approaches suggest to exploit the lexical content of the query themselves for determining a possible topic shift in the stream of issued queries and, thus, a session boundary [99, 77, 111].

To this extent, several *search patterns* have been proposed by means of lexical comparison, using different string similarity scores (e.g., *Levenstein*, *Jaccard*, etc.). However, approaches relying only on content features suffer of the so-called *vocabulary-mismatch problem*, namely the existence of topically-related queries without any shared terms (e.g., the queries "nba" and "kobe bryant" are completely different from a lexical content perspective but they are undoubtedly related). In order to overcome this issue, Shen *et al.* [138] compare *expanded representation* of queries, instead of the actual queries themselves. Each individual expanded query was obtained by concatenating the titles and the Web snippets for the top 50 results provided by a WSE for the specific query. Thus, the relatedness between query pairs was computed using cosine similarity between the corresponding expanded queries.

Mixed heuristics propose to combine the two previous approaches. Jansen *et al.* [85] assume that a new search pattern always identifies the start of a new session. Moreover, He *et al.* [77] show that statistical information collected from query logs could be used for finding out the probability that a search pattern actually implies a session boundary. In particular, they extend their previous work [76] to consider both temporal and lexical information.

Boldi *et al.* [35] introduce the *Query Flow Graph* as a model for representing data collected in WSE query logs. They exploited this model for segmenting the

query stream into sets of related information-seeking queries, leveraging on an instance of the Asymmetric Traveling Salesman Problem (ATSP).

Jones and Klinkner [92] argue that within a user's query stream it is possible to recognize particular hierarchical units, i.e., *search missions*, which are in turn subdivided into disjoint *search goals*. A search goal is defined as an atomic information need, resulting in one or more queries, while a search mission is a set of topically-related information needs, resulting in one or more goals. Given a manually generated ground-truth, Jones and Klinkner [92] investigate how to *learn* a suitable binary classifier, which is aimed to precisely detect whether two queries belong to the same task or not. Among various results, they realize that timeouts, whatever their lengths, are of limited utility in predicting whether two queries belong to the same goal, and thus to identify session boundaries. Indeed, authors do not explore how such binary classifier could be exploited for actually segmenting users' query streams into goals and missions.

Lucchese *et al.* [101] devise effective techniques for identifying *task-based sessions*, i.e. sets of possibly non contiguous queries issued by the user of a Web search engine for carrying out a given task. In order to evaluate and compare different approaches the authors built, by means of a manual labeling process, a *ground-truth* where the queries of a given query log have been grouped in tasks. The analysis of this ground-truth shows that users tend to perform more than one task at the same time, since about 75% of the submitted queries involve a multi-tasking activity. Furthermore, authors formally define the *Task-based Session Discovery Problem* (TSDP) as the problem of best approximating the manually annotated tasks, and propose several variants of well-known clustering algorithms, as well as a novel efficient heuristic algorithm, specifically tuned for solving the TSDP. These algorithms also exploit the collaborative knowledge collected by *Wiktionary* and *Wikipedia* for detecting query pairs that are not similar from a lexical content point of view, but actually *semantically* related. The proposed algorithms is evaluated on the above ground-truth. Results show that it perform better than state-of-the-art approaches, because it effectively take into account the multi-tasking behavior of users.

3.4 Time-series Analysis of the Query Log

Queries can be viewed as signals in the domain of time. In each time unit it is possible to record the occurrences of the query. The result is a sort of signal to which standard temporal series techniques could be applied [162, 160, 72, 55, 178]. These papers introduce techniques allowing the discovery of peculiar query features such as being periodic or bursty.

Adar *et al.* [2] use time series to predict i) why users issue queries and ii) how users react and cause new spreading. In particular, a novel way to compare signals coming from different sources of information. Dynamic Time Warping (DTW) is a

way to compare two time series also capturing behavioral properties by mapping inflection points and behavior such as the rise in one curve to the rise in the second, peak to peak, run to run. Figure 3.15 shows an example of DTW, against a simple linear mapping. Lines going from a curve to the other show how events are mapped within each line.

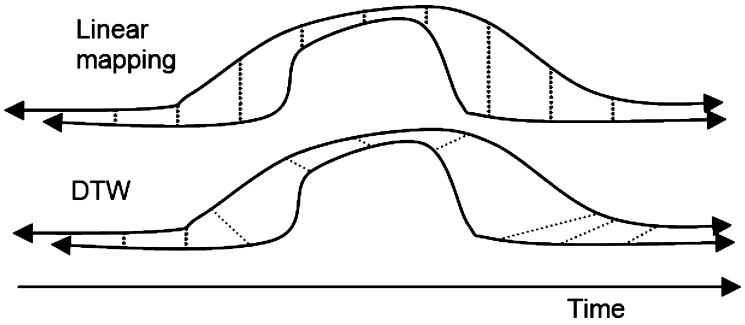


Fig. 3.15. The difference of using DTW against a simple linear mapping for comparing two time series. [2].

DTW is obtained by applying a dynamical programming algorithm minimizing the distance in terms of euclidean distance between two time series points. The algorithm produces a two dimensional array, DTW, containing how the two time series maps. The *best* warping path is simply obtained by crawling the array from the extreme corner in a backward fashion along the minimum gradient direction.

The goal of the paper is to discover how time-series are correlated in order to be able to use events in one information source to predict those in another. The datasets used are two query logs (MSN, and AOL), a blog dataset, a NEWS dataset. By using human-based tests authors devise five different general trends in time-series-based behavior prediction.

News of the weird – Events that are so weird and/or strange to be able to virally spread over a huge amount of people. *Anticipated events* – Events that produce a lot of queries but only few blog posts. *Familiarity breed contempt* – Events that are newsworthy but not searched by users. *Filtering behaviors* – Events that have the capability of deepening the filtering of categories. *Elimination of noise* – Events that combined reduces the noise that might have been generated around a topic, for instance a set of blog posts discussing a news article.

The models describing the aggregated and social behavior of users studied by Adar *et al.* [2] can be used in practice, for instance, to analyze the market, or to make search engines more reactive to changing user needs.

Zhang *et al.* [175] study time series analysis to evaluate predictive scenarios using search engine transactional logs. Authors deal with developing models for the analysis of searchers' behaviors over time and investigate if time series analy-

sis is a valid method for predicting relationships between searchers actions. Time series analysis is a method often used to understand the underlying characteristics of temporal data in order to make forecasts. The study uses a Web search engine transactional log and time series analysis to investigate users' actions. Authors conducted their analysis in two main phases. The initial phase employed a basic analysis and found that 10% of searchers clicked on sponsored links. Furthermore, the analysis reveals that from 22:00 to 24:00, searchers almost exclusively clicked on organic links with almost no clicks on sponsored links. The second, and more extensive phase deals with using a one-step prediction time series analysis method along with a transfer function method. Authors show that the period rarely affects navigational and transactional queries, while rates for transactional queries vary during different periods. Results also show that the average length of a searcher session consist of about 2.9 interactions and that this average is consistent across time periods. Moreover, results show that searchers who submit the shortest queries (i.e. in number of terms) click on highest ranked results.

Some other recent papers deal with applying time series to query log analysis and predicting users behavior.

Vlachos *et al.* [162] present several methods for mining knowledge from the MSN query logs. They analyze time series built on each query of the log. They demonstrate how to efficiently and accurately discover the important periods in a time series. Authors also propose a simple but effective method for identification of bursts (both long or short-term). Later, Vlachos *et al.* [160] motivate the need for more flexible structural similarity measures between time-series sequences, which are based on the extraction of important periodic features. Authors present non-parametric methods for accurate periodicity detection and introduce new periodic distance measures for time-series sequences. Furthermore, they combine these new measures with an effective metric tree index structure for efficiently answering *k-nearest-neighbor* queries. The goal of these tools and techniques is to assist in detecting, monitoring and visualizing structural periodic changes. Authors claim that these methods can be directly applicable in the manufacturing industry for preventive maintenance and in the medical sciences for accurate classification and anomaly detection.

Vlachos *et al.* [161] investigate lower and upper distance bounds on time-series logs when working directly in a compressed form. Optimal distance estimation means tighter bounds, leading to better candidate selection/elimination and ultimately faster search performance. Their derivation of the optimal distance bounds is based on the careful analysis of the problem using optimization principles. The experimental evaluation shows a clear performance advantage of the proposed method, compared to previous compression/search techniques. The presented method results in a 10–30% improvement on distance estimations, which in turn leads to 25–80% improvement on the search performance.

Yizhou *et al.* [152] study the problem of mining causal relation of queries in search engine query logs. They firstly detect events in query logs by applying a statistical frequency threshold and then, the causal relation of queries is mined by the geometric features of the events. The experimental results demonstrate that this approach can accurately detect the events in temporal query logs and effectively identify the causal relation of queries.

Chien *et al.* [56] try to find semantically related search engine queries based on their temporal correlation. They figure out that two queries are related if their popularity behave similarly over time. They also define a new measure of the temporal correlation of two queries based on the correlation coefficient of their frequency functions. Finally, authors provide a method for efficiently finding the highest correlated queries for a given input query.

3.5 Some Applications of Query Log Mining

Query log mining is useful for enhancing the search experience of web search engine users in terms of effectiveness and efficiency. Effectiveness in search systems refers to the quality of returned results. To this aim, five main applications can be highlighted: i) *query expansion*, ii) *query recommendation*, iii) *personalized query results*, iv) *learning to rank*, and v) *query spelling correction*. Furthermore, efficiency in search systems refers to the speed of which results are returned. It basically consists of two main topics that gain a lot of attention in the latest years: i) *caching*, ii) *index partitioning and querying in distributed web search systems*.

In the following we will investigate the major contributions that are mainly related to the results presented throughout the thesis. We will analyze *query expansion* and *query recommendation*. Interested readers could investigate the topics that are not covered in this chapter by reading [141]. It is a good starting point.

3.5.1 Query Expansion

As already showed in Section 3.1, Web search engine users tend to submit short, poorly expressive and sometimes mistyped queries. As a consequence of that search engines tend to return to users too many useless results.

One of the techniques that Web search engines use to improve precision is *query expansion*. Query expansion involves evaluating the user query and expanding it with other semantically correlated terms in order to recall additional and potentially useful documents.

Cui *et al.* [64] show that queries and documents are poorly correlated. Using a two-months query log from the Encarta search engine[2] and 41,942 documents from the Encarta website, they measure the gap between the *document space* (all the

[2] `http://encarta.msn.com`

terms contained in the documents) and the *query space* (all the terms contained in queries). Furthermore, authors define the concept of *virtual document* in the query space that is made up by all the queries for which a document has been *clicked* on. For each document in the document space, they build the corresponding *virtual document* represented as a vector where the weight of each term is defined by the $tf \times idf$ measure. The similarity between a query and the relative clicked documents is measured by computing the cosine similarity between the two corresponding vectors. Results show that in most cases the similarity values of term usages between user queries and documents are between 0.1 and 0.4. Only a small percentage of documents have similarity 0.8. The average similarity value of the whole collection is 0.28. It means that there is a large gap between document space and query space.

For the reasons above, query expansion help in reducing the gap between document space and query space. One of the first contributions on query expansion (sometimes referred also as *pseudo-relevance feedback* [168]) is [68]. In this work, Fitzpatrick and Dent propose *past-query feedback* and test its effectiveness against the TREC-5 dataset. Authors show that past-query feedback improves of 38.3% in average precision if compared to the non-query-expansion case.

Cui *et al.* [64] propose a method that exploits correlations among terms in *clicked* documents and user queries. The exploitation of click-through data is due to the assumption that clicked documents are relevant to a query. The proposed method starts by extracting a set of *query sessions* from the query log. A query session consists of a query and a list of clicked results. As an example, the query session "britney spears" – 4,982 – 2,212 – 8,412 means that a user types the query "britney spears" and she then clicks on three documents: 4,982, 2,212, 8,412. For each one of the clicked documents in each session a list of terms is extracted. While the set of all terms contained within each clicked documents form the *Document Terms* set, the set of all terms contained within all the queries make up the *Query Terms* set. Given a term in the *Document Terms* set (t_d) and a term in the *Query Terms* set (t_q), a link between t_d and t_q is created if and only if for at least one query containing t_q there exists a clicked document containing the term t_d. Each link is weighed by using the degree of *term correlation* between its endpoints. The correlation is given by the conditional probability that term t_d appears given that term t_q already appeared, $P(t_d|t_q)$. The term correlation measure is then used to devise a query expansion method relying on a function measuring the *cohesion* between a query Q and a document term t_d. The cohesion weight is given by:

$$\text{CoWeight}(Q, t_d) = log\left(\prod_{t_d \in Q} P(t_d|t_q) + 1 \right) \tag{3.1}$$

The productory in the Equation (3.1) remarks that authors assume that terms within the query are independent. The cohesion weight is used to build a list of weighted candidate expansion terms. The top-k terms with the highest weights are

selected as expansion terms for Q. The evaluation of the method is performed on 30 queries extracted randomly from a combination of the Encarta query logs, from the TREC query set, and from a small subset of manually evaluated queries. The method is compared against a baseline consisting of not using query expansion, and the *local context* method proposed by Xu and Croft [168]. The local context method has been set to use 30 expanded terms for each query, while the log-based uses 40 terms. The average precision for the baseline was 17.46% while the local context method scored 22.04%. The log-based method scored an average precision of 30.63% corresponding to an improvement of 75.42% on the baseline.

The technique proposed by Xu and Croft [168] is obsolete if compared to today's search engine technology. Moreover, expanding a query longer than 30 terms is not viable due to the overhead it causes in the corresponding query processing phase.

Billerbeck *et al.* [32] use the concept of *Query Association*. User queries are associated with a document if they are similar with the document. The idea is thus to enrich documents with an associated *Surrogate Document*. Surrogate documents are used as a source of terms for query expansion. Furthermore, to make the system more scalable, instead of keeping all of the queries associated with the documents only the M closest queries are kept. Similarity is computed using the Okapi BM25 [128] scoring function. Depending on two parameters, the methods has to be fine tuned with respect to the different values of K, and M. Scholer *et al.* empirically set $K = 5$, and $M = 3$. Billerbeck *et al.* [32] use the Robertson and Walker's term selection value formula [127] to select the e expanding terms on different combinations of ranking/expansion on documents/surrogates. Experiments conducted on a TREC collection shows the superiority of both retrieval/expansion on surrogates with respect to retrieval/expansion on the original documents (full-text).

Collins-Thompson and Callan [61] describe a Markov chain framework that combines multiple sources of knowledge on term associations. The stationary distribution of the model is used to estimate the probability that a potential expansion term reflects aspects of the original query. Authors use this model for query expansion and evaluate the effectiveness of the model by examining the accuracy and robustness of the expansion methods. Authors also investigate the relative effectiveness of various sources of term evidence. Statistically significant differences in accuracy are observed depending on the weighting of evidence in the random walk.

Chirita *et al.* [57] propose to improve Web queries by expanding them with terms collected from each user's Personal Information Repository, thus implicitly personalizing the search output. Authors introduce five broad techniques for generating the additional query keywords by analyzing user data at increasing granularity levels, ranging from term and compound level analysis up to global co-occurrence statistics, as well as to using external thesauri. The extensive empirical analysis under four different scenarios presented shows that some of these

approaches perform very well, especially on ambiguous queries, producing a very strong increase in the quality of the output rankings. Subsequently, authors move this personalized search framework one step further and propose to make the expansion process adaptive to various features of each query. A separate set of experiments indicates the adaptive algorithms to bring an additional statistically significant improvement over the best static expansion approach.

Fonseca *et al.* [69] propose a concept-based query expansion technique, which allows disambiguating queries submitted to search engines. The concepts are extracted by analyzing and locating cycles in a special type of query relations graph. This is a directed graph built from query relations mined using association rules. The concepts related to the current query are then shown to the user who selects the one concept that she interprets is most related to his query. This concept is used to expand the original query and the expanded query is processed instead. Using a Web test collection, authors show that our approach leads to gains in average precision figures of roughly 32%. Furthermore, if the user also provides information on the type of relation between her query and the selected concept, the gains in average precision go up to roughly 52%.

3.5.2 Query Recommendation

Query recommendation is an important application of query log mining. It deals with increasing precision of Web search engines by suggesting users a set of possible queries they might be interested in during their search. The idea is to help users in better formulating their information needs in order to give them the possibility to reach their needs in a shorter time, thus decreasing the number of queries a search engine should resolve.

Differently from query expansion, query suggestion aims at producing hints for users, thus giving them the possibility to select the best similar query to refine their search, instead of having the query automatically expanded with a lot of different terms.

The activity of producing suggestions from queries submitted in the past can be seen as a way of "exploiting queries submitted by experts to help non-expert users" [18]. Therefore, the majority of query suggestion techniques detect related queries by selecting those that are mostly similar to the ones submitted in the past by other users.

A naïve approach to find queries similar to another one consists of simply looking for those queries sharing terms in commons. As an example, "Pisa Vini" and "Dottorato Pisa" would be considered to some extent similar as both the two queries share the term "Pisa". Obviously, this example shows that the naïve approach might result misleading for users.

Lot of efforts have been spent on query recommendation producing a lot of contributions in the field. Starting from papers selecting queries to be suggested

from those appearing frequently in query sessions [70], to use clustering to devise similar queries on the basis of clustering membership [18, 11, 12], to use click-through data information to devise query similarity [177, 63].

Query sessions can be an important source of information for devising potentially related queries to be suggested to a user. One important idea is that if a lot of previous users when issuing the query q_1 also issue q_2 afterwards, query q_2 could be a suggestion for query q_1.

Fonseca *et al.* [70] exploit this idea above by mining association rules from query logs. Authors run an association rules mining algorithm on a query log. As computing association rules on query logs could be very computationally expensive (due to the dimensions of the data), the approach presented by Fonseca *et al.* [70] mines only rules of the form $q_i \Rightarrow q_j$ and, thus reducing the total effort needed for the overall computation. Basically, for each query q_i, all the associations above a given support σ are sorted by confidence level and saved in the form of rules $q_i \Rightarrow q_1$, $q_i \Rightarrow q_2$, ..., $q_i \Rightarrow q_m$. Authors test the proposed technique on a real-world query log coming from a Brazilian search engine and consisting of 2.312.586 queries. Experiments use a support $\sigma = 3$. The produced recommendations are evaluated by means of a survey involving five assessors. Results are encouraging as the technique scores 90.5% of precision (with five queries suggested) measured as the number of suggestions retained relevant for the five assessors. By augmenting the number of suggestions provided to users, precisions drops to 89.5% when ten queries are suggested down to 81.4% when 20 queries are suggested.

Zaïane and Strilets [170] use a *Query Memory* model to store past queries and retrieve them according to the one submitted. The method computes associations on-the-fly at query resolution time. A *Query Memory* is a set of queries represented by six different features: i) a bag of word representation of the query (*BagTerms*); ii) the frequency with which the query has been submitted (*Count*); iii) the timestamp recording the last time the query was submitted (*LDate*); iv) the timestamp recording the first time the query was submitted (*FDate*); v) the query result list stored as records made up of: URL, Title and Snippet of each result page (*QResult*); vi) the timestamp recording the date at which results were obtained. In order to produce query recommendations, seven different methods can be used:

- *Naïve query-based method*: returning queries having in common at least one term;
- *Naïve simplified URL-based method*: returning queries having in common at least one URL in the result lists;
- *Naïve simplified URL-based method*: returning queries having in common a large portion of URLs in the result list;
- *Query-Title-based method*: returning queries where terms in their result titles are contained in the submitted query;

- *Query-Content-based method*: it is the same as the above only considering snippet terms instead of title terms;
- *Common query title method*: returning queries whose results share title terms;
- *Common query text method*: it is the same of the previous one only considering snippet terms instead of the title terms.

The evaluation of the technique is conducted by means of an user study. Authors claim that it is difficult to find a winner strategy. Furthermore, authors investigated the scalability of the method. They claimed that the use of the *Query Model* in a real-world search engine needs the use of an index depicted to the search of queries containing a keyword. This implies that the method performs a double index access for each submitted query. Furthermore, this two accesses can be done in parallel on a distributed platform.

Baeza-Yates *et al.* [18] use a clustering approach to query recommendation. The query recommendation technique works by following a two level approach. An offline process clusters past queries using text from queries and clicked URLs. The online process then follows a two-stage approach: i) given an input (a query), the most representative cluster is found; ii) each query in the cluster is ranked according to two criteria: the similarity and the *attractiveness* of query answer, i.e. how much the answers of the query have attracted the attention of users. In the paper, authors refer to this as *support*. Moreover it should not be confused with the more famous support of association rules [4]. The offline query clustering algorithm operates over queries that are previously enriched by a selection of terms extracted from the clicked documents. Clusters are computed by applying a k-means algorithm. The similarity between queries is computed according to a vector-space approach. Therefore, each query q is represented as a vector whose i-th component q_i is computed as

$$q_i = \sum_{u \in URLs} \frac{Clicks(q,u) \times tf(t_i,u)}{max_t tf(t,u)}$$

where $Clicks(q,u)$ is the percentage of clicks the URL u receives when answered in response to the query q, and $tf(t,u)$ is the number of occurrences of the term t in the document pointed to URL u. All the clicked URLs are considered in computing the sum. The distance between two queries is computed by the cosine similarity of their relative vectors. The approach is evaluated on the TodoCL query log containing 6,042 unique queries with the associated click-through information. 22,190 clicks are present in the log referring to 18,527 different URLs. The evaluation is performed on ten queries by means of an user study. Results show that presenting query suggestions ranked by support (the frequency of the query in the query log) yields to more precise and high quality suggestions.

Jones *et al.* [93] propose a model for suggesting queries based on the concept of query rewriting. Basically a query is rewritten into a new one by means of query

or phrase substitution. The rewriting process is based on the Log-Likelihood Ratio (LLR) [104] measure to assess interdependencies between terms of queries. Given two terms, the more the LLR is high, the more the likelihood of the two words of being dependent is higher. Thus, query pairs with high LLR are identified as *substitutables*. Authors also propose a systemization of possible suggestions into four classes going from the most precise to the less precise class: *precise rewriting*, *approximate rewriting*, *possible rewriting*, and *clear mismatch*. *Precise rewriting* means that the query suggested has exactly the same semantic of the one to be replaced. *Approximate rewriting* is the class containing narrowed or broadened queries of the initial one. *Possible rewriting* is a still less precise query suggestion methodology: queries are in some categorical relationship. The last class, *clear mismatch* is the less precise and contains query pairs where no relationships can be found among them. Authors use the four classes for defining the two general tasks of query suggestion: *specific rewriting* (precise and approximate rewriting), and *broad rewriting* (precise, approximate, and possible rewriting). Given these two tasks, the problem of generating query recommendations can be seen as a classification problem. Authors thus adopt and test different classifiers. Training is performed on a set of manually annotated set of substitutions extracted from queries of a query log. The assessment of the method is performed by using an automatic evaluation. For the log used in [93], the precision of the method has been measured to be 76% with a recall figure of 92%.

Boldi *et al.* [35] introduce the Query Flow Graph (QFG), a graph representation of the interesting knowledge about latent querying behavior. Intuitively, in the QFG a directed edge from query q_i to query q_j means that the two queries are likely to be part of the same *search mission*. Any path over the QFG may be seen as a searching behavior, whose likelihood is given by the strength of the edges along the path. Figure 3.16 shows an example of the users' search behavior represented by means of a Query Flow Graph. Authors propose a methodology that builds a QFG by mining time and textual information as well as aggregating queries from different users. Using this approach, authors build a real-world QFG from a large-scale query log and demonstrate its utility in two concrete applications: i) finding logical sessions, and ii) query recommendation. In a later study, Boldi *et al.* [36] propose, and experimentally study, query recommendations based on short random walks on the query flow graph. The conducted experiments show that these methods can match in precision, and often improve, recommendations based on query-click graphs, without using click-through data. Experiments also show that it is important to consider transition-type labels on edges for having good quality recommendations.

Baeza-Yates and Tiberi [17] use click-through data as a way to provide recommendations. The method is based on the concept of *cover graph*. A *cover graph* is a bipartite graph of queries and URLs, where a query and a URL are connected if the URL was returned as a result for the query and a user clicked on it. To catch

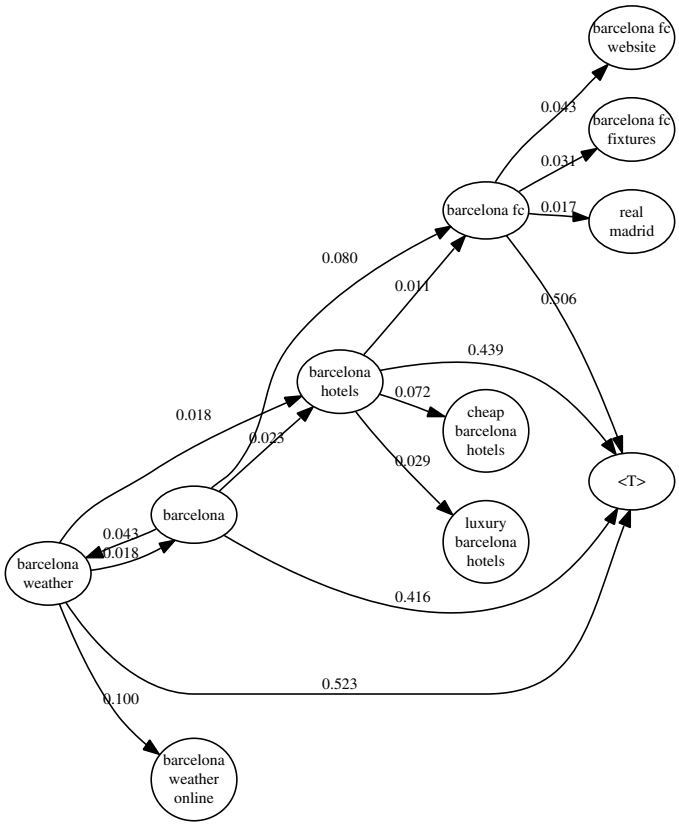


Fig. 3.16. An example of the users' search behavior represented by means of a Query Flow Graph [35].

the relations between queries, a graph is built out of a vectorial representation for queries. In such a vector-space, queries are points in a high-dimensional space where each dimension corresponds to a unique URL u that was, at some point, clicked by some user. Each component of the vector is weighted according to the number of times the corresponding URL has been clicked when returned for that query. For instance, suppose we have five different URLs, namely, $u_1, u_2, \ldots, u_5$, suppose also that for query q users have clicked three times URL u_2 and four times URL u_4, the corresponding vector is $(0, 3, 0, 4, 0)$. Queries are then arranged as a graph with two queries being connected by an edge if and only if the two queries share a non-zero entry, that is, if for two different queries the same URL received at least one click. Furthermore, edges are weighted according to the cosine similarity of the queries they connect. More formally, the weight of an edge $e = (q, q')$ is computed according to Equation (3.2). In the formula, D is the number of dimensions, i.e., the number of distinct clicked URLs, of the space.

$$W(q, q') = \frac{q \cdot q'}{|q| \cdot |q'|} = \frac{\sum_{i \leq D} q_i \cdot q'_i}{\sqrt{\sum_{i \leq D} q_i^2} \sqrt{\sum_{i \leq D} {q'_i}^2}} \tag{3.2}$$

Suggestions for a query q are obtained by accessing the corresponding node in the cover graph and extracting the queries at the end of the top scoring edges.

Baraglia *et al.* [22] propose a new model for query recommendation, the Search Shortcut Problem (SSP), that consists in recommending *successful* queries that allowed other users to satisfy, in the past, similar information needs. This new model present several advantages with respect to traditional query suggestion approaches. Firstly, it allows a straightforward evaluation of algorithms from available query log data. Moreover, it simplifies the application of several recommendation techniques from other domains. In particular, authors apply collaborative filtering techniques to this problem. The proposed query recommendation technique is evaluated on two large query logs (AOL and MSN). Different techniques for analyzing and extracting information from query logs, as well as new metrics and techniques for measuring the effectiveness of recommendations are proposed and evaluated. Results show notably accurate results, which demonstrate that collaborative filtering techniques can be useful in recommending queries.

Query suggestion has been an effective approach to help users narrow down to the information they need. However, most of the existing studies focus on only popular queries. Since rare queries possess much less information (e.g., clicks) than popular queries in the query logs, it is much more difficult to efficiently suggest relevant queries to a rare query.

Yang *et al.* [145] propose an optimal rare query suggestion framework by leveraging implicit feedbacks from users in the query logs. The model resembles the principle of *pseudo-relevance feedback* which assumes that top-returned results by search engines are relevant. However, authors argue that the clicked URLs and skipped URLs contain different levels of information and thus should be treated differently. Hence, the query suggestion framework optimally combines both the click and skip information from users and uses a random walk model to optimize the query correlation. The proposed model specifically optimizes two parameters: i) the restarting (jumping) rate of random walk, and ii) the combination ratio of click and skip information. Unlike the Rocchio algorithm, the proposed learning process does not involve the content of the URLs but simply leverages the click and skip counts in the query-URL bipartite graphs. Consequently, the model is capable of scaling up to the need of commercial search engines. Experimental results on one-month query logs from a large commercial search engine with over 40 million rare queries demonstrate the superiority of our framework, with statistical significance, over the traditional random walk models and pseudo-relevance feedback models.

Broder *et al.* propose to leverage the results from search engines as an external knowledge base for building the word features for rare queries [45]. The authors train a classifier on a commercial taxonomy consisting of 6,000 nodes for categorization. Results show a significant boost in term of precision with respect to the baseline query expansion methods. Lately, Broder *et al.* propose an efficient and effective approach for matching ads against rare queries [44]. The approach builds an expanded query representation by leveraging offline processing done for related popular queries. Experimental results show that the proposed technique significantly improves the effectiveness of advertising on rare queries with only a negligible increase in computational cost.

Mei *et al.* propose a novel query suggestion algorithm based on ranking queries with the hitting time on a large scale bipartite graph [108]. The rationale of the method is to capture semantic consistency between the suggested queries and the original query. Empirical results on a query log from a real world search engine show that hitting time is effective to generate semantically consistent query suggestions. Authors show that the proposed method and its variations are able to boost long tail queries, and personalized query suggestion.

3.6 Privacy Issues in Query Logs

Privacy in query logs becomes an hot topic in 2006 when AOL compiled a statistical sampling of more than twenty million queries entered by 650,000 users of their customers. Such enormous quantity of data was released for research purposes. Users names were replaced by numbers. Furthermore, these numbers provide a sort of *thread* by which queries by a given user could be identified. The identification process is easy if users entered some pieces of information which permits their identity to be discerned. Doing so, all other queries they made during the sampling period could be identified as theirs.

Many commercial search engines overcome the problem by simply not publishing their logs. However, lot of efforts have been spent in order to find good techniques for sanitizing query logs. As an example, Bar-Ilan in [21] state that "interesting results can be obtained from query logs without jeopardizing the privacy of users".

A seminal solution to the *anonymity preservation* challenge has been proposed by Sweeney in [153]. Sweeney introduced the concept of *k-anonymity*, which ensures that each information request contains at least k (anonymized) individuals with the same values, so that it is not possible to identify one individual in particular.

Jones *et al.* [91] provide a detailed description of a data analysis process that leads to information disclosure in a query log. They show how the combination of simple classifiers can be used to map a series of user queries into a gender, age and location showing that this approach remains very accurate even after personally

identifying information has been removed from the log. Authors emphasize that a user can be identified by a real-life acquaintance; this type of person has background knowledge on the user (e.g. location, age, gender, or even access the user's browser) and can use it to disclose the activities of the user in the log.

Adar [1] elaborates on vulnerabilities in the AOL log and shows that traditional privacy preservation methods can not be transferred directly to query logs. Adar also points out that *k-anonymity* is too costly for query log anonymization, because this type of dataset changes very rapidly. Two user anonymization methods are proposed, whose goal is to balance the achieved privacy and the retained *utility*, i.e. the usability of the anonymized log for statistical analysis. The term *utility* refers to the *data utility* of the anonymized log for the purposes of non-adversarial information acquisition. Verykios *et at.* [159] count utility as one of the important features for the evaluation of privacy preserving algorithms, next to the performance of the algorithm, the level of uncertainty with which the sensitive information can be predicted and the resistance to different data mining techniques.

Xiong and Agichtein [167] describe some important applications of query log analysis and discuss requirements on the degree of granularity of query logs. Authors then analyze the sensitive information in query logs and classify them from the privacy perspective. Two orthogonal dimensions are described for anonymizing query logs and a spectrum of approaches along those dimensions is presented. Furthermore, the authors discuss whether existing privacy guidelines such as HIPAA can apply to query logs directly.

Cooper [62] assesses seven state-of-the-art privacy-preserving techniques against three main criteria: i) how well the technique protects privacy, ii) how well the technique preserves the utility of the query logs, and iii) how well the technique might be implemented as a user control. The author highlights that achieving the right balance between protecting privacy and promoting the utility of the query logs is thus difficult but necessary to ensure that Web users can continue to rely on search engines without fear of adverse privacy consequences.

Poblete *et al.* [119] look into the privacy related implications of query log analysis and in particular, they analyze a new concern, the privacy of *businesses*. This include both institutions (such as companies) and people in the public eye (such as political leaders). Authors provide a definition of *confidential* information and analyze attacks that lead to confidentiality breaches, including methods to prevent information disclosure.

Kumar *et al.* [98] study the privacy preservation properties of a specific technique for query log anonymization: token-based hashing. In this approach, each query is tokenized, and then a secure hash function is applied to each token. Authors show that statistical techniques may be applied to partially compromise the anonymization. Authors then analyze the specific risks that arise from these partial compromises, focused on revelation of identity from unambiguous names,

addresses, and so forth, and the revelation of facts associated with an identity that are deemed to be highly sensitive. The goal of the study is twofold: to show that token-based hashing is unsuitable for anonymization, and to present a concrete analysis of specific techniques that may be effective in breaching privacy, against which other anonymization schemes should be measured.

4

The Effects of Time on Query Flow Graph-based Models for Query Suggestion

A recent query-log mining approach for query recommendation is based on *Query Flow Graphs*, a markov-chain representation of the query reformulation process followed by users of Web Search Engines trying to satisfy their information needs. In this chapter we aim at extending this model by providing methods for dealing with evolving data. In fact, users' interests change over time, and the knowledge extracted from query logs may suffer an aging effect as new interesting topics appear. Starting from this observation validated experimentally, we introduce a novel algorithm for updating an existing query flow graph. The proposed solution allows the recommendation model to be kept always updated without reconstructing it from scratch every time, by incrementally merging efficiently the past and present data.

4.1 Introduction

All popular Web search engines provide users with query suggestions to help them to formulate better queries and to quickly satisfy their information needs. As introduced in Chapter 3, query suggestion techniques are typically based on the behavior of past users of the search engine recorded in query logs. In this chapter we are interested particularly in query recommendation: the automatic generation of *interesting queries* that are related in some non trivial way to the current user information need.

A successfully query-log mining approach for generating useful query recommendation based on *Query Flow Graphs* (QFGs) [35], was recently proposed in [36]. The QFG model aggregates information in a query log by providing a markov-chain representation of the query reformulation process followed by users trying to satisfy the same information need. This chapter aims at extending the QFG model by providing a methodology for dealing efficiently with evolving data. The interests of search engine users change in fact over time. New topics may suddenly become popular (this is described as a *query burst*), while others that

attracted for some time the attention of users can lose importance. The knowledge extracted from query logs can thus suffer an aging effect, and the models used for recommendation rapidly becoming unable to generate useful and interesting queries. Unfortunately, building a new fresh QFG from scratch as soon as we discover the effect of aging is very expensive. We thus deal with this problem by introducing an *incremental* algorithm for updating an existing QFG. The solution proposed allows the recommendation model to be kept always updated by incrementally adding fresh knowledge and deleting the aged one.

In order to validate our claims and assess our methodology, we build different query flow graphs from the queries found on a large query log of a real-world search engine, and we analyze the quality of the recommendation model devised from these graphs to show that it inexorably ages. Then, we show that our algorithm for merging QFGs allows the recommendation model to be kept updated and we propose a general methodology for dealing with aging QFG models. Finally, we show that the computational time needed to merge QFGs is remarkably lower than the time required for building it from scratch, and we propose a distributed solution allowing to shorten further the time for the QFG creation/update.

The results we present here show that the model built over a QFG inexorably ages over time [23], to assess the aging effect and also to find effective anti-aging strategies to combat time effects over QFG-based models [24].

The chapter is organized as follows. Section 4.2 discusses related works, while Section 4.3 introduces the concept of query flow graph, and provides readers with some useful notations. The data used for the experiments are described in Section 4.4, while their analysis finalized to the evaluation of aging effects on the recommendation models is discussed in Section 4.5. The incremental algorithm for updating query flow graphs with fresh data is described in Section 4.6. Section 4.7 discusses its parallel implementation. Finally, Section 4.8 draws some conclusions and outlines future work.

4.2 Related Work

Different approaches have been proposed in recent years that use query logs to mine wisdom of the crowds for query suggestion.

Bruno *et al.* in [69] use an association rule mining algorithm to devise query patterns frequently co-occurring in user sessions, and a query relations graph including all the extracted patterns is built. A click-through bipartite graph is then used to identify the concepts (synonym, specialization, generalization, etc.) used to expand the original query.

Jones *et al.* in [93] introduce the notion of query substitution or query rewriting, and propose a solution for sponsored search. Such solution relies on the fact that in about half sessions the user modifies a query with another which is closely

related. Such pairs of reformulated queries are mined from the log and used for query suggestion.

Baeza-Yates *et al.* [18] use a k-means algorithm to cluster queries by considering both topics and text from clicked URLs. Then the cluster most similar to user query is identified, and the queries in the cluster with the highest similarity and attractiveness (i.e. how much the answers of the query have attracted the attention of past users) are suggested. The solution is evaluated by using a query log containing only 6,042 unique queries from the TodoCL search engine, and the suggestions to 10 different queries are evaluated by means of a user study.

Beeferman and Berger [26] apply a hierarchical agglomerative clustering technique to click-through data to find clusters of similar queries and similar URLs in a Lycos log. A bipartite graph is created from queries and related URLs which is iteratively clustered by choosing at each iteration the two pairs of most similar queries and URLs. The conducted experimental evaluation shows that the proposed solution is able to enhance the quality of the Lycos's query recommender which was used as baseline.

QFGs were introduced by Boldi *et al.* [35]. A QFG is an aggregated representation of the interesting information contained in query logs. Authors define a QFG as a directed graph in which nodes are queries, and edges are weighted by the probability of being traversed. Authors propose two weighting schemes. The first one represents the probability that two queries are part of the same search mission given that they appear in the same session, and the other one represents the probability that query q_j follows query q_i. Authors show the utility of the model in two concrete applications, namely, *finding logical sessions* and *query recommendation*. Boldi *et al.* in [36], [37] refine the previous study and propose a query suggestion scheme based on a random walk with restart model. The query recommendation process is based on reformulations of search mission. Each reformulation is classified into *query reformulation types*. Authors use four main reformulations: *generalization*, *specialization*, *error correction*, and *parallel move*. An automatic classifier was trained on manually human-labeled query log data to automatically classify reformulations. Authors showed improvements on the recommendations based on QFG models.

4.3 The Query Flow Graph

A *Query Flow Graph* is a compact representation of the information contained in a query log. It has been applied successfully to model user interactions with a web search engine and for a number of practical applications as segmenting physical sessions into logical sessions [35] or query recommendation [35, 36].

As presented in [35] a *Query Flow Graph* is a directed graph $G = (V, E, w)$ where:

- $V = Q \cup \{s, t\}$, is the set of distinct queries Q submitted to the search engine enriched with two special nodes s and t, representing a *starting state* and a *terminal state* which can be seen as the begin and the end of all the chains;
- $E \subseteq V \times V$ is the set of *directed edges*;
- $w : E \to (0..1]$ is a weighting function that assigns to every pair of queries $(q, q') \in E$ a weight $w(q, q')$.

Each distinct query is represented by a single node independently of its frequency, and the number of users who issued it. In order to build the QFG representing a given query log, we need to preprocess the data, sorting the queries by userid and by timestamp, and splitting them into physical sessions using a fixed time interval. In a second step we connect two queries q, q' with an edge if there is at least one session of the query log in which q and q' are consecutive. The third step of the QFG construction consists in weighting directed edges (q, q') on the basis of a function $w : E \to (0..1]$ that measures the probability of transition from query q to query q'. In [35], two weighting schemes are proposed. A first one based on *chaining probability* and the second one based on *relative frequencies*. For edge weighting we adopted the *chaining probability* scheme. To estimate such chaining probability, we extract for each edge (q, q') a set of features aggregated over all sessions that contain the queries q and q' appearing consecutively.

This classification step produces a set of so called *chain graphs*. Each chain graph is represented by a set of queries (i.e. nodes) interconnected by edges weighted by the probability of moving from a query to another. *Noisy* edges (i.e. those edges having a low probability of being traversed) are removed on the basis of a filtering process by means of a threshold value t.

4.4 Experimental Framework

Our experiments have been conducted on the AOL query log and on a hardware consisting of a cluster of machines equipped with G5 PowerPCs and 5 Gb of RAM each.

The AOL data-set contains about 20 million queries issued by about $650,000$ different users, submitted to the AOL search portal over a period of three months from 1st March, 2006 to 31st May, 2006. Each query record comes with the user ID, timestamp, and the list of results returned to the user. After the controversial discussion followed to its initial public delivery, AOL has withdrawn the query log from their servers and is not offering it for download anymore. We decided to run experiments on that log anyways, because of the following reasons. First of all, the log spans a long period of time and this allows us to show how models for query suggestions degrade in a more realistic way. Second, we are not disclosing any sensitive information neither about users nor about usage behavior. Therefore, we are not breaching into the privacy of any specific user. Last, but not least, the

query log is still available on the web. Everybody can easily find and download it. Indeed we consider this a strong point in favor of its use: the availability of data allows the repeatability of experiments which is an important requirement for any scientific work.

To assess the aging effects on QFG models we conducted several experiments to evaluate the impact of different factors. The log has been split into three different *segments*. Two of them have been used for training and the third one for testing. The three segments correspond to the three different months of users activities recorded in the query log. We fixed the test set – i.e. the set of queries from which we generate recommendations – to be the queries submitted in the last month. We also have conducted experiments with different training granularities, based on weekly and biweekly training sets. Results on those shorter training segments are consistent with those presented in the following, and we are omitting them for brevity.

The QFGs over the two monthly training segments have been constructed according to the algorithm presented by Boldi *et al.* in [35]. This method uses *chaining probabilities* measured by means of a machine learning method. The initial step was thus to extract those features from each training log, and storing them into a compressed graph representation. In particular we extracted 25 different features (time-related, session and textual features) for each pair of queries (q, q') that are consecutive in at least one session of the query log.

Table 4.1 shows the number of nodes and edges of the different graphs corresponding to each query log segment used for training.

Time window	Id	Nodes	Edges
March 06	$\mathcal{M}_1$	3,814,748	6,129,629
April 06	$\mathcal{M}_2$	3,832,973	6,266,648

Table 4.1. Number of nodes and edges for the graphs corresponding to the two different training segments.

It is important to remark that we have not re-trained the classification model for the assignment of weights associated with QFG edges. We reuse the one that has been used in [35] for segmenting users sessions into query chains[1]. This is another point in favor of QFG-based models. Once you train the classifier to assign weights to QFG edges, you can reuse it on different data-sets without losing in effectiveness. We want to point out, indeed, that what we are evaluating in this work is that QFGs themselves age much faster than the model used to build them. This is a subtle difference we want to state clearly.

Once the QFG has been built, the query recommendation methods are based on the probability of being at a certain node after performing a random walk over

[1] We thank the authors of [35] for providing us their model.

the query graph. This random walk starts at the node corresponding to the query for which we want to generate a suggestion. At each step, the random walker either remains in the same node with a probability α, or it follows one of the out-links with probability equal to $1 - \alpha$; in the latter case, out-links are followed proportionally to $w(i, j)$. In all the experiments we computed the stable vector of the random walk on each QFG by using $\alpha = 0.15$. Actually, the stable vector is computed according to a Random Walk with Restart model [156]. Instead of restarting the random walk from a query chosen uniformly at random, we restart the random walk only from a given set of nodes. This is done by using a preference vector v, much in the spirit of the Topic-based PageRank computation [75], defined as follows. Let $q_1, \ldots, q_n$ be a query chain (q_1 is the most recently submitted query). The preference vector v is defined in the following way: $v_q = 0$ for all $q \notin q_1, \ldots, q_n$ and $v_{q_i} \propto \beta^i$. β is a weighting factor that we set in all of our experiments to be $\beta = 0.90$.

4.5 Evaluating the Aging Effect

One of the main goals of this chapter is to show that time has some negative effects on the quality of query suggestions generated by QFG-based models. It is also worth remarking that we can safely extend the discussion that follows also to suggestion models different from QFG-based ones. As a matter of fact, the presence of "*bursty*" [95] topics could require frequent model updates whatever model we are using. To validate our hypothesis about the aging of QFG-based models we have conducted experiments on models built on the two different training segments described in the above section.

In order to assess the various reasons why a QFG-based model ages we have considered, for each segment, two classes of queries, namely $\mathcal{F}_1$, and $\mathcal{F}_3$, which respectively correspond to queries having a strong decrease and a strong increase in frequency. $\mathcal{F}_1$ is the set of the 30 queries that are among the 1,000 most frequent queries in the first month ($\mathcal{M}_1$) but whose frequency has had the greater drop in the last month covered by the query log ($\mathcal{M}_3$). Conversely, $\mathcal{F}_3$ is the set of the 30 queries among the 1,000 most frequent queries in the test log $\mathcal{M}_3$ whose frequency has the greater drop in the first part of the log $\mathcal{M}_1$. Actually, to make the assessment more significant, we do not include queries that are too similar, and we do not include queries containing domain names within the query string. Figure 4.1 graphically show where the selected queries for each class fall when we plot the popularity of the top-1000 most frequent queries in $\mathcal{M}_1$ ($\mathcal{M}_3$) by considering query ids assigned according to frequencies in $\mathcal{M}_3$ ($\mathcal{M}_1$).

Some examples of queries in $\mathcal{F}_1$ are: "*shakira*", "*americanidol*", "*ecards*", "*nfl*". Such queries are related to particular events in March 2006, for instance singer Shakira in March 2006 released a new album. Some examples of queries in $\mathcal{F}_3$ are: "*mothers day gift*", "*mothers day poems*", "*memorial day*", "*da vinci code*". As

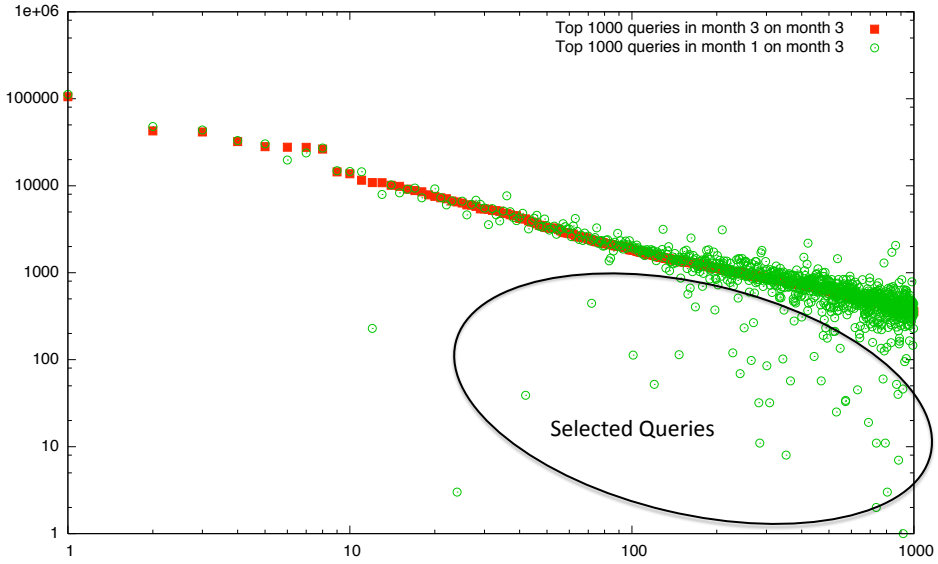


Fig. 4.1. Queries in $\mathcal{F}_3$. The set of top 1,000 queries in $\mathcal{M}_3$ compared with the same set projected on $\mathcal{M}_1$. Query identifiers are assigned according to frequencies in $\mathcal{M}_3$. The circled area in the plot highlights the zone from where $\mathcal{F}_3$ was drawn.

in the previous case, $\mathcal{F}_3$ queries are strongly related to that particular period of time. For instance, in May 2006 the movie adaptation of the popular book "Da Vinci Code" was released.

We selected two distinct sets because we want to assess the effectiveness of recommendations for both new or emerging query topics in the test log (i.e. queries in $\mathcal{F}_3$), and for queries that are frequent in the first month but poorly represented (or absent) in the test month (i.e. queries in $\mathcal{F}_1$).

The first evaluation we perform is a *human-based assessment* of the quality of query suggestions generated by models trained on the two different segments. From each query in $\mathcal{F}_1$ and $\mathcal{F}_3$ we generated the top 20 recommendations using four different sets of QFG-based models: three of them are filtered with different values of the threshold t (0.5, 0.65, and 0.75), one is generated without filtering ($t = 0$). Each set consists of QFGs built on either $\mathcal{M}_1$ or $\mathcal{M}_2$.

The generated recommendations were manually evaluated and classified as **useful** and **not useful**. We consider useful a recommendation that undoubtedly interprets the possible intent of the user better than the original query.

Table 4.3 shows the results of the human assessment performed by counting, for each query and the three different threshold levels, the number of useful suggestions. We averaged the counts over all the queries evaluated. For each training period we show the average number of useful suggestion for queries in the three different groups, i.e. $\mathcal{F}_1$, $\mathcal{F}_3$, and $\mathcal{F}_1 \cup \mathcal{F}_3$.

Set	Query	$\mathcal{M}_1$		$\mathcal{M}_2$	
		Score	Suggestions	Score	Suggestions
$\mathcal{F}_3$	da vinci	49743	da vinci's self portrait black and white	73219	da vinci and math
		47294	the vitruvian man	33769	da vinci biography
		35362	last supper da vinci	31383	da vinci code on portrait
		31307	leonardo da vinci	29565	flying machines
		30234	post it	28432	inventions by leonardo da vinci
		30234	handshape 20stories	26003	leonardo da vinci paintings
				23343	friends church
				23343	jerry c website
	survivor	8533	watch survivor episodes	7392	survivor preview
		8083	survivor island	7298	watch survivor episodes
		4391	2006 survivor	7110	survivor island album
		4310	bubba gump hat	4578	survivor edition 2006
		4310	big shorts	3801	cbs the great race
				3801	chicken and broccoli
	lost	16522	lost fan site	5138	lost season 2
		3634	abcpreview.go.com	3742	lost update
		2341	altricious	2162	lost easter eggs
		2341	5 year treasury rate	1913	abcpreview.go.com
		2341	1-800-sos-radon	1913	antique curio cabinets
				1913	4440
$\mathcal{F}_1$	anna nicole smith	11113	anna nicole smith nude	23497	anna nicole smith recent news
		11101	anna nicole smith - supreme court	18694	anna nicole smith and supreme court
		11054	anna nicole smith diet	18546	anna nicole smith and playboy
		10274	anna nicole smith with liposuction	16836	anna nicole smith pictures
		4677	cameron diaz video	15289	anna nicole smith free nude pics
		4677	calcination	13436	anna nicole smith diet
				6642	branson tractors
				6642	bandlinoblu jeans
	harley davidson	5097	harley davidson ny	5749	harley davidson sound system manual
		2652	american harley davidson	3859	automatic motorcycles
		2615	2002 harley davidson ultra classic	3635	harley davidson credit
		2602	adamec harley davidson	3618	cherokee harley davidson
		2341	air fight	2103	harley davidson sporster
		2341	928 zip code	1965	2002 harley davidson classic
		2341	antispy ware	1394	regions banking
				1394	aol email only
				1394	adultactioncamcom
	shakira	10989	shakira video	3281	hips don't lie
		7522	shakira albums	3281	shakira hips don't lie video
		7522	shakira my hips don't lie	3175	shakira video
		5836	shakira biography	3042	shakira wallpaper
		3864	70s music funk	2811	shakira album
		3864	97.1zht	2592	shakira nude
				1868	cant fight the moonlight
				1868	free video downloads

Table 4.2. Some examples of recommendations generated on different QFG models. Queries used to generate recommendations are taken from different query sets. For each query we present the most important recommendations with their assigned relative scores.

Filtering threshold	Average number of useful suggestions on $\mathcal{M}_1$			Average number of useful suggestions on $\mathcal{M}_2$		
	$\mathcal{F}_1$	$\mathcal{F}_3$	$\mathcal{F}_1 \cup \mathcal{F}_3$	$\mathcal{F}_1$	$\mathcal{F}_3$	$\mathcal{F}_1 \cup \mathcal{F}_3$
0	2.51	2.02	2.26	2.12	2.46	2.29
0.5	3.11	2.69	2.9	2.88	2.87	2.87
0.65	3.02	2.66	2.84	2.8	2.71	2.76
0.75	3	2.64	2.82	2.72	2.68	2.7

Table 4.3. Model aging statistics varying the model type and the temporal window. Results were manually assessed.

From the table we can draw some interesting conclusions. First, the performance of the models built from $\mathcal{M}_1$ and $\mathcal{M}_2$ are quite similar (column $\mathcal{F}_1 \cup \mathcal{F}_3$). This might seem a counterexample to the hypothesis that the models age. Actually, by breaking down the overall figure into separate figures for $\mathcal{F}_1$ and $\mathcal{F}_3$ we can observe that for all the queries in $\mathcal{F}_3$ the suggestions built from $\mathcal{M}_2$ are more useful than those built on $\mathcal{M}_1$.

Furthermore, by inspecting some of the suggestions generated for the queries shown in Table 4.2, it is evident that some of the suggestions are *fresher* (i.e. more up-to-date) in the case of a model built on $\mathcal{M}_2$ than those obtained on models built on $\mathcal{M}_1$. This is particularly true for queries in $\mathcal{F}_3$. For instance, for the query "*lost*" suggestions computed by a model trained on $\mathcal{M}_2$ appear to be more meaningful than those suggested using an old model on that particular period of time. Furthermore, another interesting observation is that filtering (i.e. removing noisy edges) works pretty well since it increases the average number of useful suggestions.

When we performed the assessment of the suggestions we noted a phenomenon regarding the scores computed on the different QFGs by the random walk-based method. Let us consider again the results shown in Table 4.2 and let us look at the suggestions, with the relative scores, computed for 6 queries (3 queries from $\mathcal{F}_1$ and 3 queries from $\mathcal{F}_3$) on $\mathcal{M}_1$ and $\mathcal{M}_2$.

As we go further down the list sorted by score, when the quality of the suggestions starts to degrade, we often observe that the useless suggestions are associated with the same low score values, e.g. "*regions banking*", "*aol email only*", "*adultactioncamcom*" are three different (and useless) query suggestions for the query "*harley davidson*" whose QFG computed score is always 1394.

From the above observation we make the following hypothesis that we will use to derive a second automatic evaluation methodology to assess the "usefulness" of suggestions:

> *when a QFG-based query recommender system gives the same score to consecutive suggestions, these recommendations and the following ones having a lower score are very likely to be useless.*

A QFG-based recommender system recommends queries by computing a random walk with restart on the model. At each step, the random walker either remains in the same node with a probability α, or it follows one of the out-links with probability equal to $1 - \alpha$. Out-links are followed proportionally to $w(i, j)$. Let us suppose the recommender system starts recommending more than k queries sharing the same score for the given query q. On the QFG model it means that the query q has more than k out-links sharing the same probability ($w(i, j)$). Due to the lack of information the system is not able to assign a priority to the k recommended queries. This is the reason why we consider these recommendations as "useless".

This heuristic considers useful k query recommendations if the suggestions following the top-k recommended queries have equal scores associated with them. Consider again the case of the query "*harley davidson*", we have six queries with different scores and then the remaining queries (for which the associated scores are equal) are clearly useless.

We perform the automatic analysis described above to the 400 most frequent queries in the third month for which recommendations were generated on models built on either $\mathcal{M}_1$ or $\mathcal{M}_2$. For all the experiments we set $k = 3$. Table 4.4 shows that according to this measure of quality filtered models works better than unfiltered ones. The filtering process reduces the "noise" on the data and generates more precise knowledge on which recommendations are computed. Furthermore, the increase is quite independent from the threshold level, i.e. by increasing the threshold from 0.5 to 0.75 the overall quality is, roughly, constant.

Filtering threshold	Average number of useful suggestions on $\mathcal{M}_1$	Average number of useful suggestions on $\mathcal{M}_2$
0	2.84	2.91
0.5	5.85	6.23
0.65	5.85	6.23
0.75	5.85	6.18

Table 4.4. Recommendation statistics obtained by using the automatic evaluation method on a set of 400 queries drawn from the most frequent in the third month.

We further break down the overall results shown in Table 4.4 to show the number of queries on which the QFG-based model generated a given number of useful suggestions. We plot this histogram to compare those numbers on $\mathcal{M}_1$ and $\mathcal{M}_2$ in Figure 4.2. To highlight more the effect of incremental updates we show in Figure 4.3 the total number of queries having at least a certain number of useful recommendation. For example, the third bucket shows how many queries have at least three useful suggestions. For each bucket, results for $\mathcal{M}_2$ are always better than the ones for $\mathcal{M}_1$.

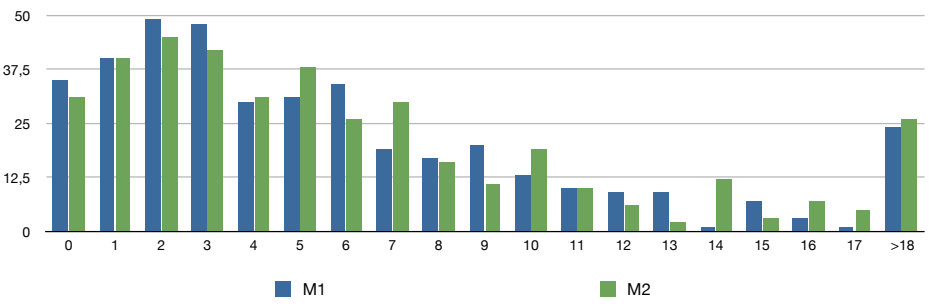


Fig. 4.2. Histogram showing the number of queries (on the y axis) having a certain number of useful recommendations (on the x axis). Results are evaluated automatically.

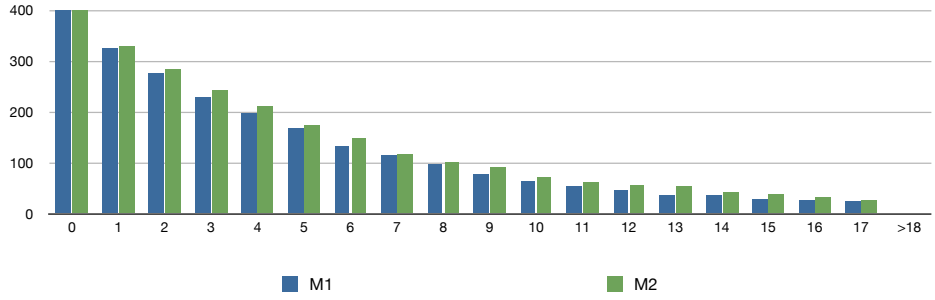


Fig. 4.3. Histogram showing the total number of queries (on the y axis) having at least a certain number of useful recommendations (on the x axis). For instance the third bucket shows how many queries have at least three useful suggestions.

First of all, when we train a QFG-based model on $\mathcal{M}_2$ the percentage of queries having 0 useful results is remarkably lower than those measured on the model trained on $\mathcal{M}_1$. Furthermore, for Figure 4.3 we can observe that a model trained on $\mathcal{M}_2$ has a larger percentage of queries for which the number of useful suggestions is at least 4.

This confirms our hypothesis that QFG-based recommendation models age and have to be updated in order to always generate useful suggestions.

4.6 Combating Aging in Query-Flow Graphs

Models based on Query Flow Graphs age quite rapidly in terms of their performance for generating useful recommendations.

The pool of queries we use to generate recommendation models ($\mathcal{M}_1$) contains both frequent and time-sensitive queries. We consider time-sensitive those queries that are both frequent and have a large variation with respect to their value in the previous month(s) (e.g. $> 50\%$ of positive variation). Time-sensitive queries are related for instance to movies, new products being launched, fashion, and in general with news events generating query bursts. In Figure 4.1 it is easy to identify time-sensitive queries by looking at those with the greater variation between their frequencies over the two months.

As Tables 4.3 and 4.8 show, if we compute recommendations on older models the average recommendation quality degrades. Indeed, *stable* queries, i.e. those queries that are (relatively) frequent in each period of the year, also have to be taken into account by the model. Therefore, we must be able to give suggestions that are not too much influenced by time or, in other words, to update the recommendation model instead of rebuilding a new model from scratch disregarding older knowledge.

There are two ways of building an updated QFG: i) rebuilding the model by considering the whole set of queries from scratch, or ii) using the old QFG-based model and adding fresh queries to it.

A straightforward option is to compute a new QFG representing the two months. The new part of the query log ($\mathcal{M}_2$) is merged with the old one ($\mathcal{M}_1$) obtaining a new data-set to be used for building the new model. The merged data-set can be processed from scratch by performing all the steps necessary to build a QFG (i.e. preprocessing, features generation, compression, normalization, chain graph generation, random walk computation). The old model is simply discarded.

The other option is computing recommendations on an *incremental model* that is built by merging the old QFGs with the one obtained from the new queries. In this way: i) we refresh the model with new data covering time-related queries; ii) the *old* part of the model contributes to maintain quality on frequent and time-unrelated queries. The methodology we are proposing incrementally extends the recommendation model with a sub-model built on more recent data. A sound incremental approach has to consistently update the old model with fresh data continuously or after fixed periods of time. Another advantage of this approach is that the new model can be built by spending only the time needed to build a QFG from a relatively small set of new queries, plus the cost of the merging process.

Let us introduce an example to show the main differences among the two approaches in terms of computational time. Table 4.5 shows the total elapsed times to create different QFGs. Suppose the model used to generate recommendations consists of a portion of data representing one month (for $\mathcal{M}_1$ and $\mathcal{M}_2$) or two months (for $\mathcal{M}_{12}$) of the query log. The model is being updated every 15 days (for $\mathcal{M}_1$ and $\mathcal{M}_2$) or every 30 days (for $\mathcal{M}_{12}$). By using the first approach, we pay 22 (44) minutes every 15 (30) days to rebuild the new model from scratch on a new set of data obtained from the last two months of the query log. Instead, by using the second approach, we need to pay only 15 (32) minutes for updating the one-month (two-months) QFG.

Dataset	*From scratch* strategy [min.]	*Incremental* strategy [min.]
$\mathcal{M}_1$ (March 2006)	21	14
$\mathcal{M}_2$ (April 2006)	22	15
$\mathcal{M}_{12}$ (March and April)	44	32

Table 4.5. Time needed to build a Query Flow Graph from scratch and using our *incremental* approach (from merging two QFG representing an half of data).

The time spent in updating incrementally the model is, in practice, shorter than the time needed to build the model from scratch (in our case it is almost two third (i.e. 32%) that time). The process of merging two QFGs can be performed using an open-source Java tool [39] that implements a graph algebra giving users

the possibility to perform some operations on WebGraph [34] encoded graphs. In our experiments we used three different QFGs built on $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_{12}$ defined as in the first column of Table 4.5.

Query	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_{12}$
mothers day	2	3	3
da vinci	4	6	7
lost	2	4	6
philippines	2	2	3

Table 4.6. Manual assessment of the number of useful recommendations generated for some time-related queries on the three different models.

Table 4.6 shows the importance of having an incremental approach for time-related queries. It is evident from the table that the model built on $\mathcal{M}_{12}$ always gives the best recommendations, in terms of quality, with respect to the two separate models $\mathcal{M}_1$ and $\mathcal{M}_2$.

Table 4.7 shows example query suggestions generated by the system, to demonstrate the improved quality of the recommendations. These results suggest a positive effect of the incremental-update method on the recommendation quality.

As in the previous section we evaluated the quality of the recommendations, also by using the automatic procedure above described. The results are shown on Table 4.8. We have used the same set of 400 queries for which recommendations were generated using the QFG built on $\mathcal{M}_{12}$.

Again, the results suggested by the anecdotal evidence, are confirmed by the assessment procedure. The model built on the two merged train segments is better than the single $\mathcal{M}_2$ model (which was, in turn, better than the model built on $\mathcal{M}_1$). Improvements, in this case, are quite significant and range from 25% to 28% in accuracy.

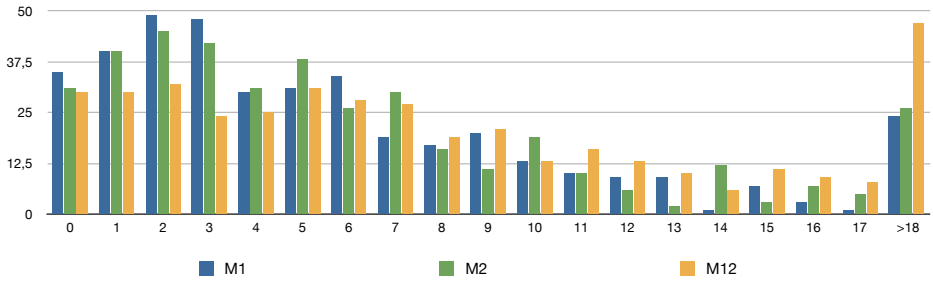


Fig. 4.4. Histogram showing the number of queries (on the y axis) having a certain number of useful recommendations (on the x axis). Results are evaluated automatically.

Set	Query	$\mathcal{M}_{12}$	
$\mathcal{F}_3$	da vinci	86251	da vinci and math
		85436	da vinci biography
		83427	da vinci code on portrait
		82119	da vinci's self portrait black and white
		80945	flying machines
		79563	inventions by leonardo da vinci
		74346	leonardo da vinci paintings
		30426	friends church
	survivor	7250	survivor preview
		7250	watch survivor episodes
		7236	survivor island panama
		7110	survivor island exile
		4578	survivor edition 2006
		3980	survivor games
		3717	big shorts
		3717	baltimore bulk trash
	lost	13258	lost fan site
		3716	lost season 2
		3640	abcpreview.go.com
		3640	lost chat room
		3582	lost update
		3272	lost easter eggs
		1913	henry gale
		1858	4440
		1858	1-800-sos-radon
		1858	4440
$\mathcal{F}_1$	anna nicole smith	15174	anna nicole smith recent news
		14876	anna nicole smith and supreme court
		13567	anna nicole smith court appeal
		12768	anna nicorle smith and playboy
		10509	anna nicole smith pictures
		9832	anna nicole smith free nude pics
		9411	anna nicole smith show
		8971	anna nicole smith diet
		8880	branson tractors
		8880	bandlinoblu jeans
	harley davidson	5969	harley davidson premium sound system owners manual
		4073	harley davidson ny
		4001	automatic motorcycles
		3738	cherokee harley davidson
		3038	harley davidson credit
		2562	custom harley davidson
		2494	harley davidson sporster
		2166	2002 harley davidson classic
		2085	regions banking
		2085	1998 dodge ram ground effects kits
		2085	adultactioncamcom
	shakira	4174	hips don't lie
		4174	shakira albums
		4140	shakira hips don't lie video
		3698	shakira video
		3135	shakira nude
		3099	shakira wallpaper
		3020	shakira biography
		3018	shakira aol music
		2015	free video downloads

Table 4.7. Some examples of recommendations generated on different QFG models. Queries used to generate recommendations are taken from different query sets.

Filtering threshold	Average number of useful suggestions on $\mathcal{M}_2$	Average number of useful suggestions on $\mathcal{M}_{12}$
0	2.91	3.64
0.5	6.23	7.95
0.65	6.23	7.94
0.75	6.18	7.9

Table 4.8. Recommendation statistics obtained by using the automatic evaluation method on a relatively large set of 400 queries drawn from the most frequent in the third month.

Using the automatic evaluation method we have investigated the main reasons why we obtain such an improvement. Looking at the different bars in Figure 4.4 we can observe that values that have had the greatest improvement are those corresponding to a number of suggestions larger than 8 (with the only exceptions of

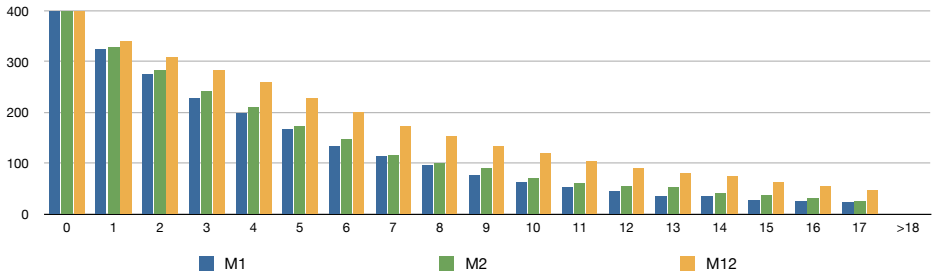


Fig. 4.5. Histogram showing the total number of queries (on the y axis) having at least a certain number of useful recommendations (on the x axis). For instance the third bucket shows how many queries have at least three useful suggestions.

the cases of 10 and 14 suggestions). In particular the improvement is remarkable in the case of "more than 18" useful suggestions given. Figure 4.5 also shows the total number of queries having at least a certain number of useful recommendations. Again, results for $\mathcal{M}_{12}$ are remarkably better than those referring to $\mathcal{M}_1$, and $\mathcal{M}_2$.

To conclude, we have shown that recommendations given using QFG-based models are sensitive to the aging of the query base on which they are built. We have also shown the superiority of QFG-based models built on queries drawn from larger period of time and we have shown how to build such a models without the need of retraining them from scratch.

4.7 Distributed QFG Building

In this section we present a method to build QFGs that exploit the observation made above on the feasibility of an incremental approach to QFG-based model update. We present an approach for building a QFG-based model on a distributed architecture.

4.7.1 Divide-and-Conquer Approach

The first approach exploits a parallel divide-and-conquer computation that proceeds by dividing the query log into m distinct segments, building the QFG in parallel on each processor available and the iteratively merging the different segments until a single QFG is obtained.

The process is depicted in Figure 4.6. Our algorithm builds a QFG as follows:

1. the query log is split into m parts. In the example represented in Figure 4.6 files were split into 15 days intervals;

2. the features requested to build the QFG are extracted from the data contained in each interval. Each interval, is processed in parallel on the different machines of the cloud;
3. each part is compressed using the WebGraph framework, obtaining a partial data-graph;
4. using the graph algebra described in [39], each partial graph is iteratively merged. Each iteration is done in parallel on the different available nodes of the cloud;
5. the final resulting data-graph is now processed with other steps [35] (normalization, chain extraction, random walk) to obtain the complete and usable QFG.

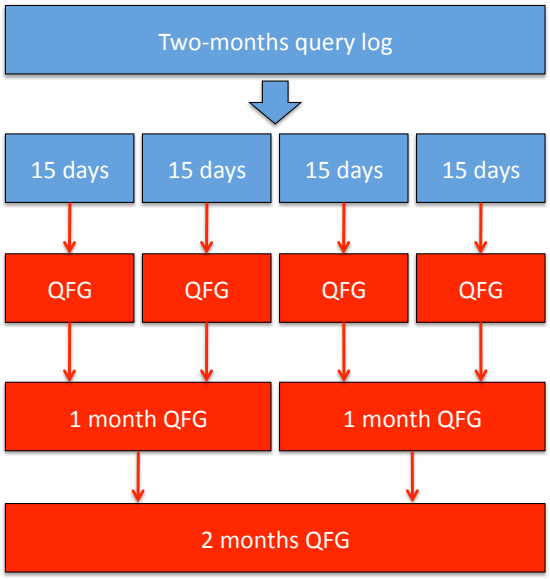


Fig. 4.6. Example of the building of a two months query flow graph with a parallel approach.

Table 4.9 summarizes the computational costs of building a QFG in a distributed way. The main benefit of this approach is to significantly reduce the time needed to perform the preliminary generation step.

4.8 Summary

In this chapter we studied the effect of time on recommendations generated using *Query Flow Graphs* [35] (QFGs). These models aggregate information in a query log by providing a markov-chain representation of the query reformulation process

generation of 15-days data-graph	6 min 15 sec.
merge of two 15-days data-graphs	5 min 23 sec.
merge of two one-month data-graphs	11 min 04 sec.
Total time	22 min 42 sec.

Table 4.9. Time needed to build a two-months data-graph using our *incremental* approach and splitting the query log in four parts.

followed by multiple users. In this chapter we showed how to extend QFG-based recommendation models to evolving data. We showed that the interests of search engine users change over time and new topics may become popular, while other interests that focused for some time the attention of the crowds can suddenly loose importance. The knowledge extracted from query logs can thus suffer from an aging effect, and the models used for recommendations rapidly become unable to generate useful and interesting suggestions. We showed that the building of a new fresh QFG from scratch is expensive. To overcome this problem we introduced an *incremental* algorithm for updating an existing QFG. The solution proposed allows the recommendation model to be kept always updated by incrementally adding fresh knowledge and deleting the aged one.

In order to validate our claims and assess our methodology, we built different QFGs from the query log of a real-world search engine, and we analyze the quality of the recommendation models obtained from these graphs to show that they inexorably age. It is worth noticing that a comprehensive user-study is (almost) impossible on this kind of task. To assess the effect of aging with a survey, we would need to make users aware of the context and news events happened in the period to which the query log is referred (March-May 2006). Then, we proposed a general methodology for dealing with aging QFG models that allows the recommendation model to be kept up-to-dated in a remarkably lower time than that required for building a new model from scratch. As a side result we proposed a parallel/distributed solution allowing to make QFG creation/update operations scalable.

5

Incremental Algorithms for Effective and Efficient Query Recommendation

Query recommender systems give users hints on possible *interesting queries* relative to their information needs. Most query recommenders are based on static knowledge models built on the basis of past user behaviors recorded in query logs. These models should be periodically updated, or rebuilt from scratch, to keep up with the possible variations in the interests of users. In this chapter we study query recommender algorithms that generate suggestions on the basis of models that are updated continuously, each time a new query is submitted. We extend two state-of-the-art query recommendation algorithms and evaluate the effects of continuous model updates on their effectiveness and efficiency. Tests conducted on an actual query log show that contrasting model aging by continuously updating the recommendation model is a viable and effective solution.

5.1 Introduction

In the latest years, web search engines have started to provide users with query recommendations to help them refine queries and to quickly satisfy their needs. Query suggestions are generated according to a model built on the basis of the knowledge extracted from query logs. The model usually contains information on relationships between queries that are used to generate suggestions (see Section 3.5.2).

In Chapter 4 we show that as the model is built on a previously collected snapshot of a query stream, its effectiveness decreases due to interest shifts [54]. To reduce the effect of aging, query recommendation models must be periodically re-built or updated.

In this chapter we thus propose two novel algorithms, based on previously proposed, state-of-the-art query recommendation solutions, that update their model continuously on the basis of each new query processed. We name the new class of query recommender algorithms proposed here *incrementally updating* query recommender systems to point out that this kind of systems update the model on which recommendations are drawn without the need for rebuilding it from scratch.

Designing an effective method to update a recommendation model poses interesting challenges due to: i) *Limited memory availability* – queries are potentially infinite, and we should keep in memory only those queries "really" useful for recommendation purposes, ii) *Low response time* – recommendations and updates must be performed efficiently without degrading user experience.

Some of the approaches considered in related works are not suitable for continuous updates because modifying a portion of the model requires, in general, the modification of the whole structure. Therefore, the update operation would be too expensive to be of practical relevance. Other solutions exploit models which can be built incrementally. The two algorithms we propose use two different approaches to generate recommendations. The first uses association rules for generating recommendations, and it is based on the *static* query suggestion algorithm proposed in [70], while the second uses click-through data, and its *static* version is described in [17].

We conduct multiple tests on a large real-world query log to evaluate the effects of continuous model updates on the effectiveness and the efficiency of the query recommendation process. Result assessment used an evaluation methodology that measures the effectiveness of query recommendation algorithms by means of different metrics. Experiments show the superiority of incrementally updating algorithms with respect to their static counterparts. Moreover, the tests conducted demonstrate that our solution to update the model each time a new query is processed has a limited impact on system response time.

An incremental approach to continuously update indices rather than query recommenders was proposed in [71]. Similar efficiency and effectiveness trends were noted, and hence, that approach forms a motivation for the described incremental recommender algorithm approach.

5.1.1 Main Contributions

This chapter extends and groups together, in a complete framework, several major contributions:

- a novel class of query recommendation algorithms whose models are continuously updated as user queries are processed (previously introduced in [41]);
- four new metrics to evaluate the quality of the recommendations computed (two of them previously introduced in [41]);
- an analysis of the effect of time on the quality and coverage of the suggestions provided by the algorithms presented and by their static counterparts using both the four new metrics defined and two syntax-based metrics.

Novel, and unpublished contributions presented in this chapter are:

- a deeper analysis of the quality of recommendations provided by the algorithms presented and by their static counterparts using two new quality metrics and two syntax-based metrics;

- an analysis of the correlation of the effectiveness of recommendations between results from syntax-based and quality metrics;

The chapter is organized as follow: Section 5.2 discusses related works, while Section 5.3.1 discusses two state-of-the-art algorithms in their original formulation and afterward, in Section 5.3.2 we discuss the main issues concerning their incrementally updating versions. The definition of four novel metrics used to assess the results are presented in Section 5.4, while the evaluation of the effectiveness of the algorithms for query recommendation is discussed in Section 5.5. Finally, Section 5.6 draws some conclusions and discusses future work.

5.2 Related Work

The wisdom of the crowds, i.e., the behavior of many individuals is smarter than the behavior of few intelligent people, is the key to query recommenders and to many other web 2.0 applications. We now present a review of state-of-the-art techniques for query recommendation.

Document-based. Baeza-Yates *et al.* in [18] propose to compute groups of related queries by running a clustering algorithm over the queries and their associated information recorded in the logs. Semantically similar queries may even not share query-terms if they share relevant terms in the documents clicked by users. Thus, the algorithm avoids the problem of comparing and clustering a sparse collection of vectors, in which semantically similar queries are difficult to find. Query suggestions are ranked according to two principles: i) the similarity of the queries to the input query, and ii) the support, which measures how much the answers of the query have attracted the attention of users. The method uses a modified version of the TF-IDF and a ranking method based on the relevance of a query with respect to its own cluster. The solution is evaluated by using a query log containing 6,042 unique queries from the TodoCL search engine, and the suggestions to 10 different queries are evaluated by means of a user study.

Click-through-based. Beeferman and Berger in [26] apply a hierarchical agglomerative clustering technique to click-through data to find clusters of similar queries and similar URLs in a Lycos log. A bipartite graph is created from queries and related URLs which is iteratively clustered by choosing at each iteration the two pairs of most similar queries and URLs. The experimental evaluation shows that the proposed solution is able to enhance the quality of the Lycos's query recommender which was used as baseline.

Cao *et al.* propose a query suggestion approach based on contexts [49]. A query context consists of recent queries issued by a user. The query suggestion process is structured according to two steps. An offline phase summarizes user queries into concepts (i.e., a small set of similar queries) by clustering a click-through

bipartite graph, and extracts query sessions, and builds a concept sequence suffix tree, representing a query suggestion model, by using session data. The concept sequence suffix tree is built bottom-up starting from sequence of concepts (i.e., list of candidate suggestions), which constitute the leaves. A parent of two nodes is the longest proper suffix of the concept sequences associated to such nodes. The root node corresponds to the empty sequence. The on-line step finds the context of the submitted query, and its related concepts suggesting associated queries to the user. This solution was experimented with a large query log containing $151,869,102$ unique queries and $114,882,486$ unique URLs.

Session-based. Boldi *et al.* introduce the concept of *Query Flow Graph* (QFG) [35]. A QFG is an aggregated representation of the interesting information contained in query logs. Authors define a QFG as a directed graph in which nodes are queries, and edges are weighted by the probability $w(q_i, q_j)$ of being traversed. Authors propose two weighting schemes. The first one represents the probability that two queries are part of the same search mission given that they appear in the same session, and the other one represents the probability that query q_j follows query q_i. Furthermore, authors highlight the utility of the model in two concrete applications, namely, *finding logical sessions* and *query recommendation*. Boldi *et al.* refine the previous study in [36], [37] proposing a query suggestion scheme based on a random walk with restart model. The query recommendation process is based on reformulations of search missions. Each reformulation is classified into *query reformulation types*. Authors use four main reformulations: *generalization*, *specialization*, *error correction*, and *parallel move*. An automatic classifier was trained on manually human-labeled query log data to automatically classify reformulations. Baraglia *et al.* show that the QFG model ages [24] and propose strategies for updating it efficiently.

Fonseca *et al.* use an association rule mining algorithm to devise query patterns frequently co-occurring in user sessions, and a query relations graph including all the extracted patterns is built [69]. A click-through bipartite graph is then used to identify the concepts (synonym, specialization, generalization, etc.) used to expand the original query.

Jones *et al.* introduce the notion of query substitution or query rewriting, and propose a solution for sponsored search [93]. Such a solution relies on the fact that in about half sessions the user modifies a query with another which is closely related. Such pairs of reformulated queries are mined from the log and used for query suggestion.

White *et al.* present a novel web search interaction feature which, for a given query, provides links to websites frequently visited by other users with similar information needs [166]. These links complement traditional search results, allowing direct navigation to authoritative resources for the query topic. Authors propose a user study showing that a search enhanced by destination suggestions outperforms

other systems for exploratory tasks, with best performance obtained from mining past users behavior at query-level granularity.

5.3 Incremental algorithms for query recommendation

Our claim is that continuously updating the query recommendation model is both useful and feasible. The feasibility of developing an *incrementally updating* version of a query recommender system depends on the technique used to compute the suggestions. For efficiency, these algorithms should only store a reduced set of information.

We validate our claims by studying two well-known query recommendation algorithms and modifying them to continuously update the model on which recommendations are computed. It is worth mentioning that not all query recommendation algorithms can be designed to on-line update their recommendation model. For example, some of the approaches presented in Section 5.2 are based on indexing terms of documents selected by users, clustering click-through data, or extracting knowledge from users' sessions. Such operations are too expensive to be performed on-line, and their high computational costs would compromise the efficiency of their recommender system. Furthermore, as shown in [29], the fusion of similar approaches typically yields minimal gains.

The two algorithms considered use different approaches for generating recommendations. The first uses association rules [70] (henceforth *AssociationRules*), while the second exploits click-through data [17] (henceforth *CoverGraph*). Hereinafter, we will refer to the original formulations of the two algorithms as *static*, as opposed to their relative incremental versions which will be called *incremental*. Not all the recommendation models can be efficiently updated incrementally. Indeed, for efficiency reasons, incremental versions of recommending algorithms do not compute exactly the same model. Instead, they compute an approximation that produces good results due to the continuous updating of the model.

5.3.1 Static solutions

Static solutions work by preprocessing historical data (represented by past users' activities on query logs), building an on-line recommendation module that is used to provide suggestions to users.

AssociationRules. Fonseca *et al.* uses association rules as a basis for generating recommendations [70]. The algorithm is based on two main phases. The first uses query log analysis for session extraction, and the second basically extracts association rules and identifies highly related queries. Each session is identified by all queries sent by an user in a specific time interval ($t = 10$ minutes). Let $I = I_1, \ldots, I_m$ be the set of queries and T the set of user sessions t. A session $t \in T$

is represented as a vector where $t_k = 1$ if session t contains query $k \in [1, \ldots, m]$, 0 otherwise.

Let X be a subset of I. A session t satisfies X, if for all items I_k in X, $t_k = 1$. *Association rules* are implications of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds with i) a *confidence* factor of c if $c\%$ of the transactions in T that contains X also contains Y, and ii) a *support* s if $s\%$ of the sessions in T contains $X \cup Y$. The problem of mining associations is to generate all the rules having a support greater than a specified minimum threshold (*minsup*). The rationale is that distinct queries are considered related if they occur in many user sessions.

Suggestions for a query q are simply computed by accessing the list of rules of the form $q \Rightarrow q'$ and by suggesting the q''s corresponding to rules with the highest support values.

CoverGraph. Baeza-Yates et al. use click-through data as a way to provide recommendations [17]. The method is based on the concept of *cover graph*. A *cover graph* is a bipartite graph of queries and URLs, where a query and a URL are connected if the URL was returned as a result for the query and a user clicked on it.

To catch the relations between queries, a graph is built out of a vectorial representation for queries. In such a vector-space, queries are points in a high-dimensional space where each dimension corresponds to a unique URL u that was, at some point, clicked by some user. Each component of the vector is weighted according to the number of times the corresponding URL has been clicked when returned for that query. For instance, suppose we have five different URLs, namely, $u_1, u_2, \ldots, u_5$, suppose also that for query q users have clicked three times URL u_2 and four times URL u_4, the corresponding vector is $(0, 3, 0, 4, 0)$. Queries are then arranged as a graph with two queries being connected by an edge if and only if the two queries share a non-zero entry, that is, if for two different queries the same URL received at least one click. Furthermore, edges are weighted according to the cosine similarity of the queries they connect. More formally, the weight of an edge $e = (q, q')$ is computed according to Equation 5.1. In the formula, D is the number of dimensions, i.e., the number of distinct clicked URLs, of the space.

$$W(q, q') = \frac{q \cdot q'}{|q| \cdot |q'|} = \frac{\sum\limits_{i \leq D} q_i \cdot q'_i}{\sqrt{\sum\limits_{i \leq D} q_i^2} \sqrt{\sum\limits_{i \leq D} {q'_i}^2}} \tag{5.1}$$

Suggestions for a query q are obtained by accessing the corresponding node in the cover graph and extracting the queries at the end of the top scoring edges.

5.3.2 Incremental algorithms

The interests of search-engine users change over time, and new topics may become popular. Consequently, the knowledge extracted from query logs can suffer from an aging effect, and the models used for recommendations rapidly become unable to generate useful and interesting suggestions [24]. Furthermore, the presence of *bursty* [95] topics could require frequent model updates independent of the model used.

The algorithms proposed in Section 5.3.1 use a statically built model to compute recommendations. *Incremental* algorithms are radically different from static methods for the way they build and use recommendation models. While static algorithms need an off-line preprocessing phase to build the model from scratch every time an update of the knowledge base is needed, incremental algorithms consist of a single online module integrating the two functionalities: i) *updating* the model, and ii) providing suggestions for each query.

Starting from the two algorithms presented above, we design two new query recommender methods continuously updating their models as queries are issued. Algorithms 1 and 2 formalize the structure of the two proposed incremental algorithms that are detailed in the following. The two incremental algorithms differ from their static counterparts by the way in which they manage and use data to build the model. Both algorithms exploit *LRU* caches and *Hash* tables to store and retrieve efficiently queries and links during the model update phase.

Our two incremental algorithms are inspired by the *Data Stream Model* [109] in which a stream of queries are processed by a database system. Queries consist modifications of values associated with a set of data. When the dataset fits completely in memory, satisfying queries is straightforward. Turns out that the entire set of data cannot be contained in memory. Therefore, an algorithm in the data stream model must decide, at each time step, which subset of the set of data is worthwhile to maintain in memory. The goal is to attain an approximation of the results we would have had in the case of the non-streaming model. We make a first step towards a data stream model algorithmic framework aimed at building query recommendations. We are aware that there is significant room for improvement, especially in the formalization of the problem in the streaming model. Nonetheless, we show empirically that an incremental formulation of each of the two popular query recommenders maintains the high accuracy of suggestions.

IAssociationRules. Algorithm 1 specifies the operations performed by *IAssociationRules*, the incremental version of AssociationRules.

The data structures storing the model are updated at each iteration. We use the LastQuery auxiliary data structure to record the last query submitted by u. Since the model and the size of LastQuery could grow indefinitely, whenever they are full, the LRUInsert function is performed to keep in both structures only the most recently used entries.

Algorithm 1 IAssociationRules

```
1: loop
2:    (u, q) ← GetNextQuery() {Get the query q and the user u who submitted it.}
3:    ComputeSuggestions (q, σ) {Compute suggestions for query q over σ.}
4:    if ∃LastQuery (u) then
5:       q′ ← LastQuery (u)
6:       LastQuery (u) ← q {Update the last query submitted by u.}
7:       if ∃σ_{q′,q} then
8:          ++σ_{q′,q} {Increment Support for q′ ⇒ q.}
9:       else
10:         LRUInsert (σ, (q′, q)) {Insert an entry for (q′, q) in σ. If σ is full, remove an
            entry according to an LRU policy.}
11:      end if
12:   else
13:      LRUInsert (u, q, LastQuery) {Insert an entry for (u, q) in LastQuery. If
         LastQuery is full, remove an entry according to an LRU policy.}
14:   end if
15: end loop
```

Claim. Keeping up-to-date the AssociationRule-based model is $O(1)$.

The proof of the claim is straightforward. The loop at line 3 of Algorithm 1 is made up of constant-cost operations (whenever we use hash structures for both LastQuery and σ). LRUInsert maintains the most recently submitted queries in the model.

Algorithm 2 ICoverGraph

```
1: Input: A threshold τ.
2: loop
3:    (u, q) ← GetNextQuery() {Get the query q and the user u who submitted it.}
4:    ComputeSuggestions (q, σ) {Compute suggestions for query q over σ.}
5:    c = GetClicks (u, q)
6:    if ∃queryHasAClickOn (c) then
7:       queryHasAClickOn (c) ← q
8:    else
9:       LRUInsert (queryHasAClickOn, c)
10:   end if
11:   for all q′ ≠ q ∈ queryHasAClickOn (c) s.t. W ((q, q′)) > τ do
12:      if w > τ then
13:         if ∃σ_{q′,q} then
14:            σ_{q,q′} = w {For each query that shares at least one clicked url, update the
               model}
15:         else
16:            LRUInsert (σ, (q′, q), w)
17:         end if
18:      end if
19:   end for
20: end loop
```

ICoverGraph. The incremental version of CoverGraph adopts a solution similar to that used by IAssociationRules. It uses a combination of LRU structures and associative arrays to incrementally update the (LRU managed) structure σ. Algorithm 2 shows the description of the algorithm. The hash table queryHasAClickOn is used to retrieve the list of queries having c among their clicked URLs. This data structure is stored in a fixed amount of memory, and whenever its size exceeds the allocated capacity, an entry is removed on the basis of a LRU policy (this justifies the conditional statement at line 6).

Claim. Keeping up-to-date a CoverGraph-based model is $O(1)$.

Actually, the cost depends on the degree of each query/node in the cover graph. As shown in [17], i) the degree of nodes in the cover graph follows a power-law distribution, and ii) the maximal number of URLs between two queries/nodes is constant, on average. The number of iterations needed in the loop at line 11 can be thus considered constant.

From the above methods, it is clear that to effectively produce recommendations, a continuous updating algorithm should have the following characteristics:

- The algorithm must cope with an undefined number of queries. LRU caches keep in memory only the most relevant items for which it is important to produce recommendations. LRU-like structures, like the one used in the previously presented algorithms are suitable enough, have a constant managing complexity and are very good candidates for this purpose.
- The lookup structures used to generate suggestions and maintain the models must be efficient, possibly constant in time. Random-walks on graph-based structures, or distance functions based on comparing portions of texts, etc., are not suitable for our purpose.
- A modification of an item in the model must not involve a modification of the entire model. Otherwise, update operations take too much time and jeopardize the efficiency of the method.

The feasibility of an incremental update of the recommendation model is an important point of our work. The update operations must run in parallel with the query processor. Therefore, those operations must not constitute a bottleneck for the entire system. As analyzed above, we have succeeded in proposing a method for keeping up-to-date two algorithms in constant time. Furthermore, in the incremental algorithms we use efficient data structures, and an optimized implementation of the model update algorithm. For each new query, our algorithms are able to update the model, and to produce suggestions in, on-the-order-of, a few tenth of a second. Such response times guarantee the feasibility of the approach on a real-world search engine where the query recommender and the query processor run in parallel.

5.4 Quality Metrics

Assessing the effectiveness of recommender systems is a tough problem that can be addressed either through *user-studies* or via automatic evaluation mechanisms.

We opted for an automatic evaluation methodology conducted by means of two previously proposed metrics, and four novel metrics based on the analysis of users' traces contained in query logs. The four novel metrics measure the overlap between queries actually submitted by the users and recorded in the tails of users' sessions and suggestions generated starting from the first queries in the same sessions. The more users actually submitted queries suggested, the more the recommender system is considered effective.

The two, basic, syntax-based metrics used are the *TermBased*, and the *ResultsMetric* proposed by Balfe and Smyth in [20]. Let q be the submitted query, and let the set $S = \{s_1, \ldots, s_m\}$ be the m recommendations computed over q. We define the following two metrics:

$$\tau(q, S) = \frac{1}{|S|} \sum_{s_i \in S} \frac{|q \cap s_i|}{|q \cup s_i|} \qquad (TermBased)$$

$$\rho(q, S) = \frac{1}{|S|} \sum_{s_i \in S} \frac{|\mathrm{res}(q) \cap \mathrm{res}(s_i)|}{|\mathrm{res}(q) \cup \mathrm{res}(s_i)|} \qquad (ResultsMetric)$$

where $|q \cap s_i|$ is the number of shared terms, and $\mathrm{res}(s)$ is the set of results retrieved by a search engine for the query s.

The *TermBased* metric considers terms composing the query. It counts how many terms in the original query occur in the suggested queries. For example, suppose q is the query "query recommender system". Suppose, then, that suggestions are: "query suggestion", "query recommendation system", "recommending queries". The TermBased score is, then, equal to 1/4. Note that the TermBased score is equal to 1 only when all the suggested queries are equal to the submitted one. The trivial recommendation algorithm maximizing this score is, thus, the one generating the input query as the only suggestion for each query. It is a baseline for the evaluation and provides a measure of the syntactic distance of two given queries.

The *ResultsMetric* metric aims at catching semantic similarities among queries exploiting web search engine results. For each query the first k results are retrieved by querying a search engine. In our experiments we retrieved the top-10 results returned by the Yahoo! Web Search Engine for each query. Then, the number of results shared by the submitted query and the suggested ones is divided by the total number of results retrieved.

The most important limitation of the previous two metrics is that they consider a single issued query comparing it with all the ones provided by the recommender system. In other words, the metrics do not take into account the history of the

users and how they interact with the search engine. To overcome this limitation we study four new metrics considering users' histories from query logs.

To focus the evaluation on the most recently submitted queries in the user session, we introduce four new metrics. While two of them work by evaluating the effectiveness of recommendations on a per query basis, the remaining two metrics work by measuring the effectiveness of the recommendations provided within the user session. They are defined as follows. Let $\mathbb{S}$ be the set of all users' sessions in the query log, and let $S = \{q_1, \ldots, q_n\}$ be a user session of length n. We define $S_1 = \{q_1, \ldots, q_{\lfloor \frac{n}{2} \rfloor}\}$ to be the set of queries in the first half of the session, and let $R_j = \{r_1, \ldots, r_m\}$ be the set of top-m query recommendations returned for the query $q_j \in S_1$[1]. For each q_j, we now define $S_2 = \{q_{j+1}, \ldots, q_n\}$ to be the $n - j$ most recently submitted queries in the session, $S^* = \{q_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, q_n\}$ to be the set of queries in the second half of the session, and

$$QueryOverlap = \frac{1}{K} \sum_{\substack{r_i \in R_j \\ s_k \in S_2}} [r_i = s_k] f(k) \tag{5.2}$$

$$LinkOverlap = \frac{1}{K} \sum_{\substack{r_i \in \text{findClk}(R_j) \\ s_k \in \text{clk}(S_2)}} [r_i = s_k] f(k) \tag{5.3}$$

$$Omega = \frac{1}{K \cdot |S_1|} \sum_{j \in S_1} \sum_{\substack{r_i \in R_j \\ s_k \in S^*}} [r_i = s_k] f(k) \tag{5.4}$$

$$LinkOmega = \frac{1}{K \cdot |S_1|} \sum_{j \in S_1} \sum_{\substack{r_i \in \text{findClk}(R_j) \\ s_k \in \text{clk}(S^*)}} [r_i = s_k] f(k) \tag{5.5}$$

where $[expr]$ is a boolean function whose result is 1 if $expr$ is *true* or 0 otherwise, $clk(S_2)$ is a function returning the set of clicked URLs by the user for the queries in S_2, $findClk(R_j)$ is a function returning the set of clicked URLs by other users for the queries in R_j, and $f(k)$ is a weighting function allowing us to differentiate the importance of each recommendation depending on the position it occupies in the second part of the session. The value of K is defined as $\sum_{k=1}^{m} f(k)$, where $m = |S_2|$ for the *QueryOverlap*, $m = |clk(S_2)|$ for the *LinkOverlap*, $m = |S^*|$ for the *Omega*, and $m = |clk(S^*)|$ for the *LinkOmega* metric. K normalizes the values in the range $[0, 1]$.

The intuition behind of the metrics presented above is that a suggestion is useful if it let the user reach his/her goal quicker. Here goal is meant to be the user's information need. In other words, suppose Alice wants to have all the information available on training for a marathon. She starts by issuing the query "marathon training". She goes on by issuing "marathon training program", "marathon running training", "marathon training from scratch", and finally, by issuing "running

[1] In our experiments we use $m = 5$.

marathon training program" Alice finds the information she was looking for. The goal of the metrics presented above is to give higher scores to recommenders suggesting the query "running marathon training program" rather than suggesting any preceding query in Alice's session. In fact, link-based metrics, *LinkOverlap* and *LinkOmega*, are more sophisticated and capture the concept of letting a user find the information s/he wanted by comparing the clicked links instead of the submitted query.

Finally, the *Coverage* of a recommendation model is defined as the fraction of queries for which a recommendation can be computed.

5.5 Experiments

5.5.1 Experimental Setup

We conducted our experiments on a collection consisting of the first $3,200,000$ queries from the AOL query log [116]. The AOL data-set contains about 20 million queries issued by about $650,000$ different users, submitted to the AOL search engine over a period of three months from 1st March, 2006 to 31st May, 2006.

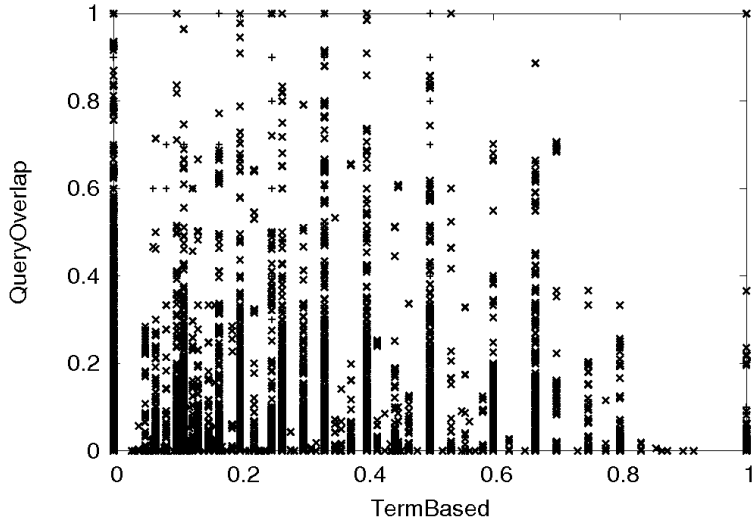
Regardless of the controversy spawned by the public release of this log, it is still readily available on the web. Unlike often used, hidden, proprietary logs, the use of the AOL query log supports experimental repeatability, an important requirement of any scientific study. Hence, we conduct our experiments using this log.

5.5.2 Correlation of Metrics

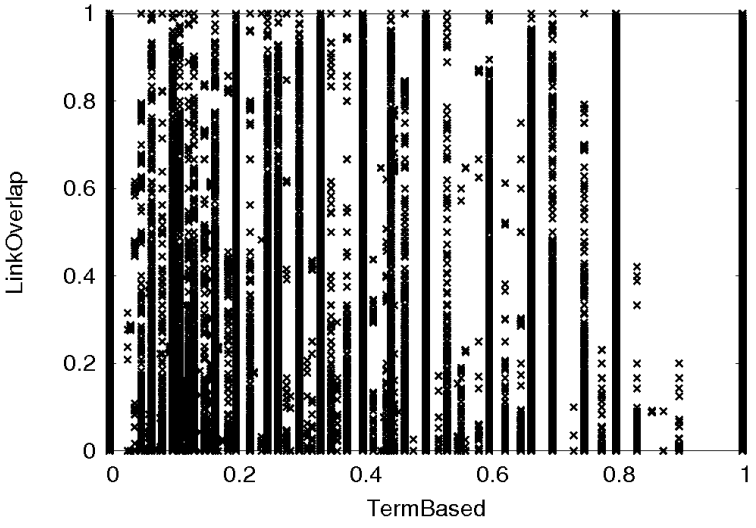
First of all, we measure the correlation between our two classes of metrics (syntax-based and session-based). The correlation is computed using a set of several hundred queries. In Figures 5.1 and 5.2 each point represents a query and shows the correlation between two different instances of syntax-based and session-based metrics for the given query. They show that there is no correlation within the two syntax-based metrics and session-based ones. As an example, in 5.1(b) there are some black vertical line resulting from the overlap of a high number of points. These lines illustrate that for a set of queries LinkOverlap assumes very different values while TermBased present the same behavior. Our two classes of metrics capture two unrelated aspects of the generated suggestions. Syntax-based metrics measure the goodness of the proposed recommendations from a term level point of view, or from how many search results they share. Whereas our new metrics focus on a different aspect: the relatedness of recommendation with respect to the "history" of the user, and how s/he interacts with the search engine.

5.5.3 Results

First, we analyze the effect of time on the static models (Section 5.3.1) showing that this type of models age as time passes.



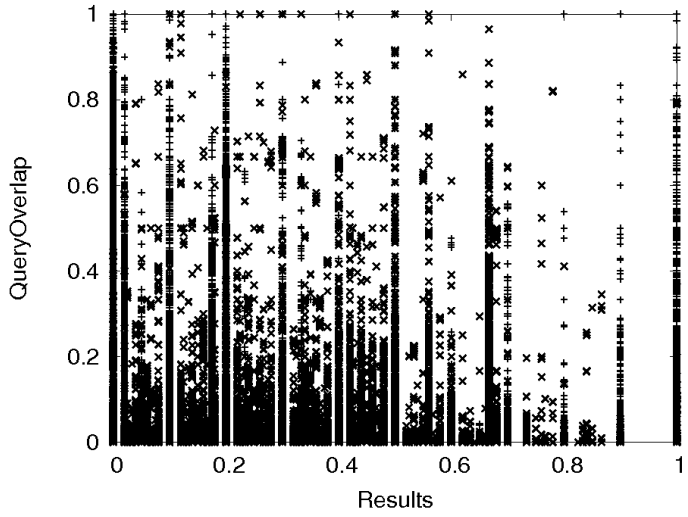
(a)



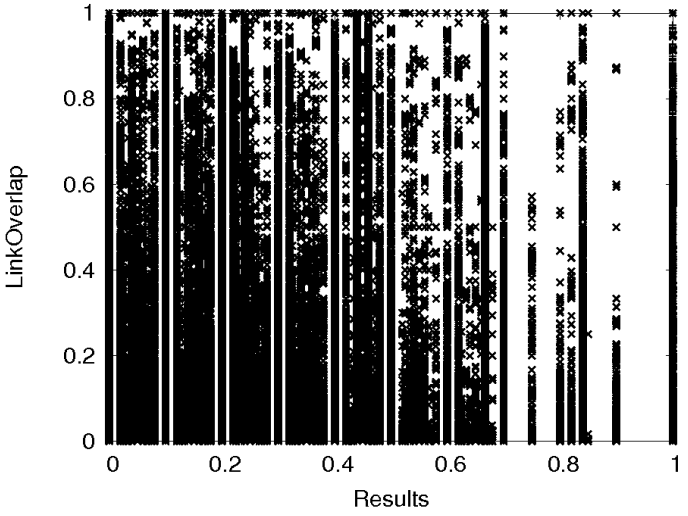
(b)

Fig. 5.1. TermBased vs. QueryOverlap, TermBased vs. LinkOverlap

The plots reported in Figures 5.3, 5.4, and 5.5, show the effectiveness of query suggestions on a per time window basis for both the static and incremental algorithms. We use a "timeline" composed of 10 days of the query log. The "timeline" is divided into ten intervals, each corresponding to one day of queries stored in the query log (about 400,000 queries). The queries in the first time interval were used to train the models used by the algorithms. While static models are trained



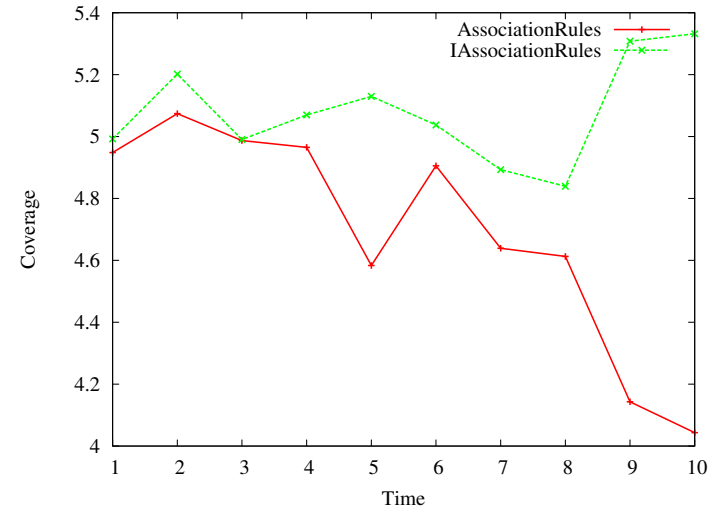
(a)



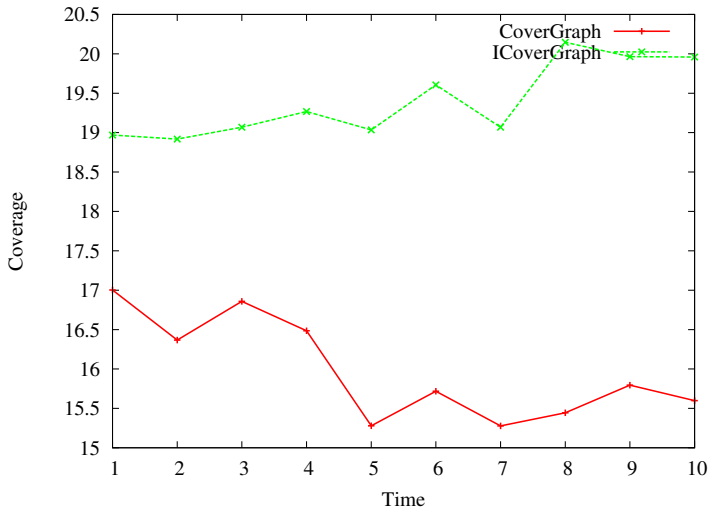
(b)

Fig. 5.2. ResultsMetric vs. QueryOverlap, ResultsMetric vs. LinkOverlap

only on the first interval, the incremental counterparts update their model on the basis of the queries submitted during the entire timeline considered. Effectiveness of recommendations generated by the different algorithms during the remaining nine days considered is measured by means of the *LinkOverlap*, *QueryOverlap*, and *Coverage* metrics.



(a)



(b)

Fig. 5.3. Coverage for AssociationRules, IAssociationRules, CoverGraph, and ICoverGraph as a function of the time.

Our first finding is illustrated in Figure 5.3, where coverage, i.e., the percentage of queries for which the algorithms are able to generate recommendations, is plotted as a function of time. In both plots, the coverage measured for the static versions of the recommendation algorithms decreases as time passes. In particular, at the end of the observed period, AssociationRules and CoverGraph lose 20%, and 9% of their initial coverage, respectively. Even if CoverGraph appears to be

more robust than AssociationRules, both algorithms suffer an aging effect on their models. On the other hand, the coverage measured for the two incremental algorithms is always greater than the one measured for their respective static versions. In particular, at the end of the observed period, the IAssociationRules algorithm covers 23.5% more queries with respect to its static version, while ICoverGraph covers 22% more queries with respect to CoverGraph. This is due to the inclusion in the model of new and "fresh" data.

Figures 5.4 and 5.5 illustrate the effectiveness of recommendations produced by the static and incremental versions of the AssociationRules and CoverGraph algorithms as a function of the time. Both QueryOverlap and LinkOverlap in Figures 5.4 and 5.5 are measured in the above described setting.

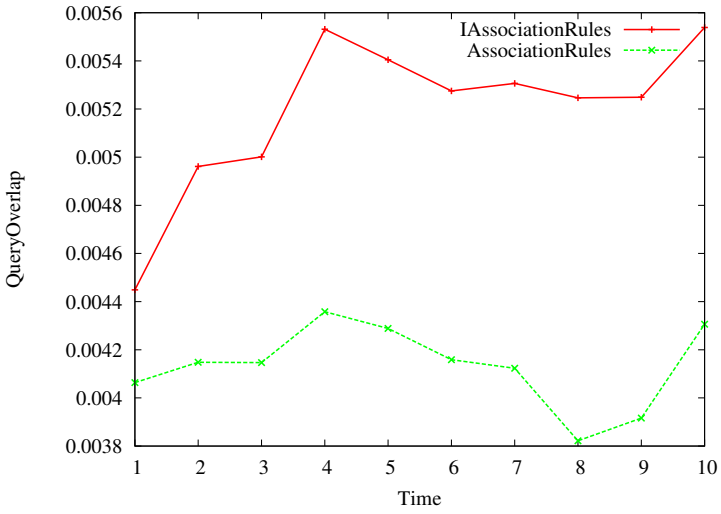
From the plots we can see that the two static algorithms behave in a slightly different way. CoverGraph seems to suffer more than AssociationRules for the aging of its recommendation model.

By considering both QueryOverlap and LinkOverlap metrics, AssociationRules is able to return better quality recommendations than CoverGraph, but, as Figure 5.3 shows, the coverage of queries for which suggestions can be generated is lower. In particular, CoverGraph is able to give suggestions to a number of queries which is three times larger than the one measured with AssociationRules.

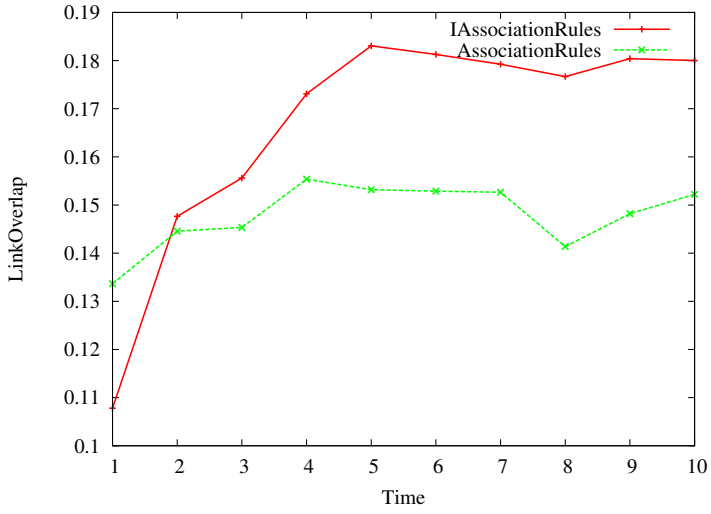
Figures 5.6 and 5.7 show the effectiveness of recommendations produced by the static and incremental versions of AssociationRules and CoverGraph by varying the size of the model used ("training set" for the static algorithms, and "cache dimensions" for the incremental algorithms). The effectiveness of recommendations is measured by using Omega and LinkOmega. Plots in Figure 5.6 (Omega) reveal that AssociationRules and its incremental version (IAssociationRules) are remarkably more sensible to model dimension variations with respect to its competitor, CoverGraph (ICoverGraph). Furthermore, plots in Figure 5.7 (LinkOmega) are obtained by adding click-through data in the evaluation process. They show that CoverGraph is more competitive in both the static and incremental versions, i.e., it produces recommendations with an higher number of clicks associated than AssociationRules, while AssociationRules and IAssociationRules are still more sensible to model dimension variations.

Figure 5.8 shows the relations between model dimensions ("training set" for the static algorithms, and "cache dimensions" for the incremental algorithms) and the syntax-based metrics. Increasing the training set for AssociationRules gives an improvement for the ResultsMetric. Therefore, the recommender model is always more able to suggest queries sharing search engine results with the original submitted query. Furthermore, augmenting the training set for CoverGraph gives bad performances on both syntax-based metrics.

We argue that incremental algorithms for query recommendation can provide better recommendations because they do not suffer from model aging, and can rapidly cover also *bursty* topics. Figure 5.8 clearly shows that for incremental al-



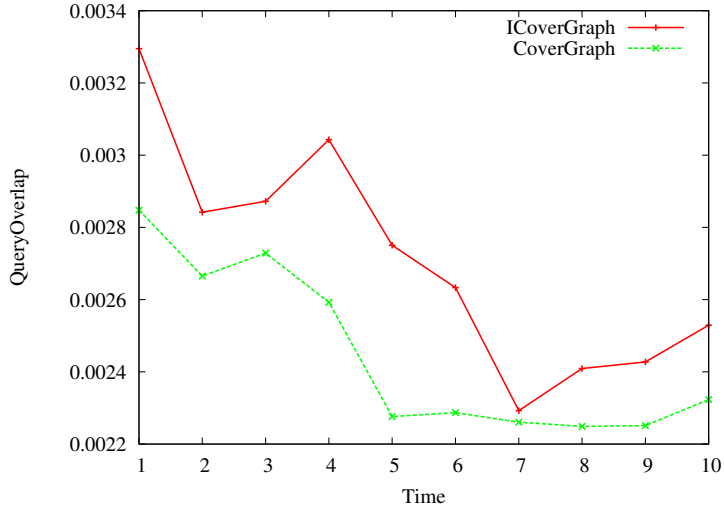
(a)



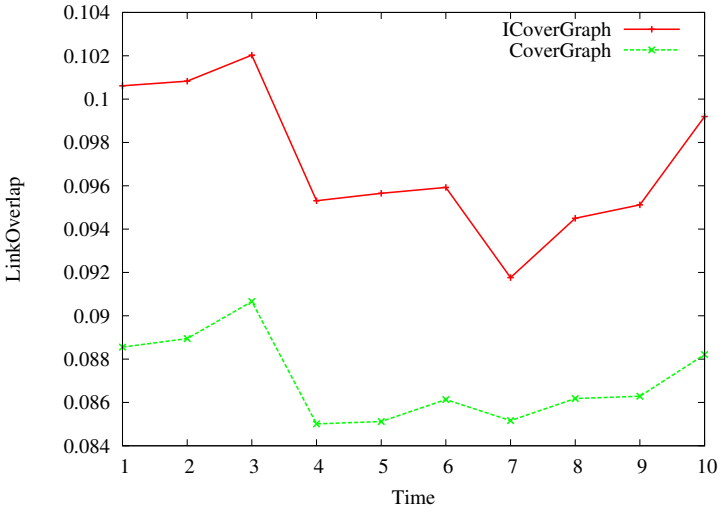
(b)

Fig. 5.4. QueryOverlap, and LinkOverlap for AssociationRules, and IAssociationRules as a function of the time.

gorithms an increment of cache size does not provide sensitive improvements of the values for both syntax-based metrics. From the figures, it is evident that the effectiveness of recommendations provided by both static and incremental models eventually stabilize. Indeed, the proposed incremental algorithms IAssociation-Rules and ICoverGraph produce better recommendations. With the exception of the initialization phase (see Figure 5.4, LinkOverlap) in which the model warms



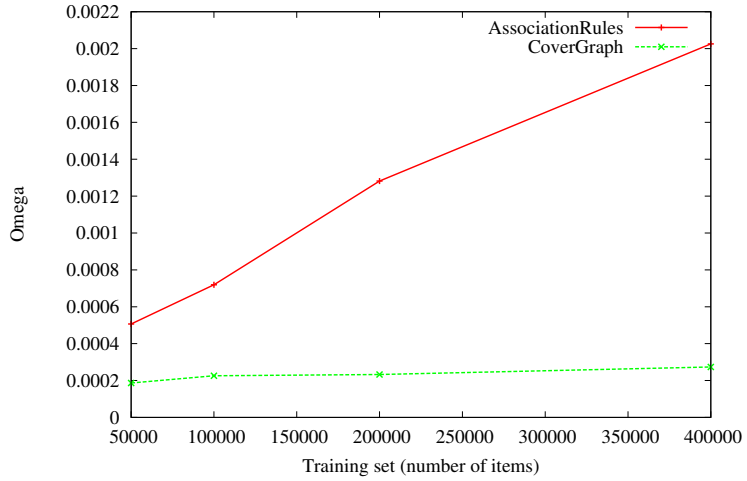
(a)



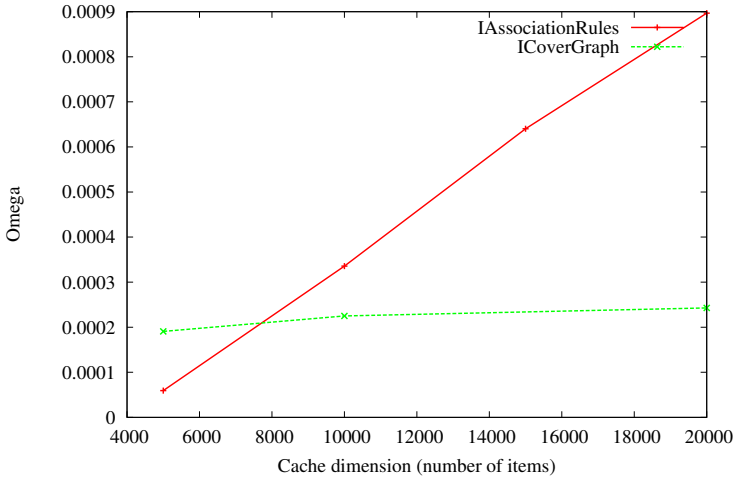
(b)

Fig. 5.5. QueryOverlap, and LinkOverlap for CoverGraph, and ICoverGraph as a function of the time.

up, the percentage of effective suggestions generated by the two incremental algorithms during the entire period observed is larger than those provided by their static counterparts.



(a)

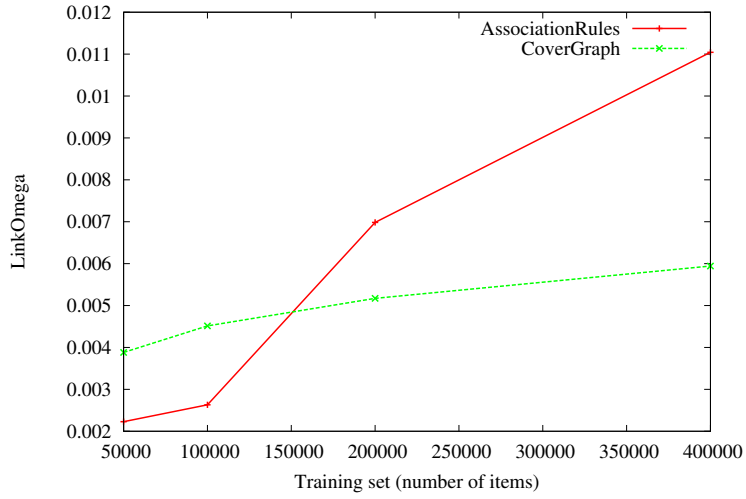


(b)

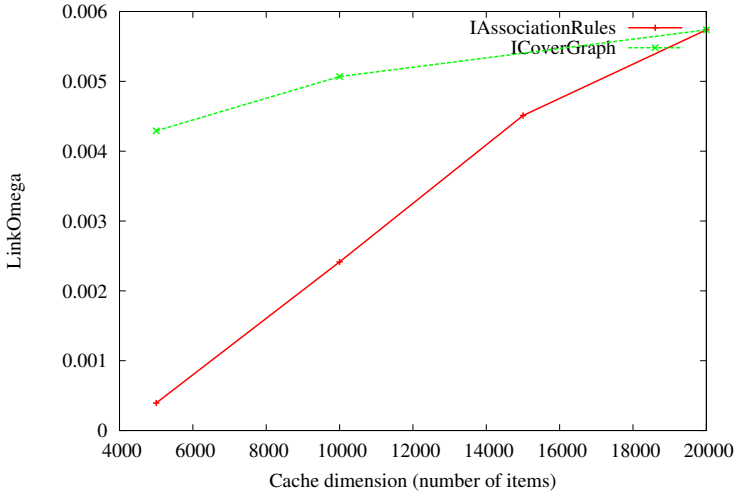
Fig. 5.6. Omega for AssociationRules, CoverGraph, and IAssociationRules, ICoverGraph.

5.6 Summary

We studied the effects of incremental model updates on the effectiveness of two query suggestion algorithms. As the interests of search-engine users change over time and new topics become popular, the knowledge extracted from historical usage data can suffer an aging effect. Consequently, the models used for recommendations may rapidly become unable to generate high-quality and interesting suggestions.

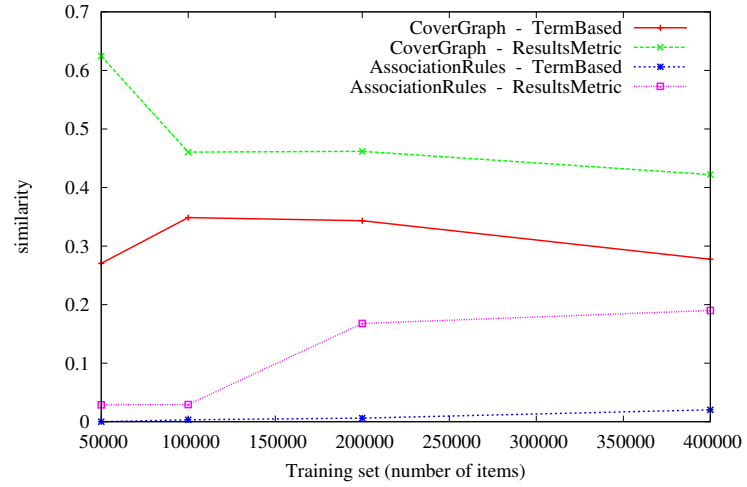


(a)

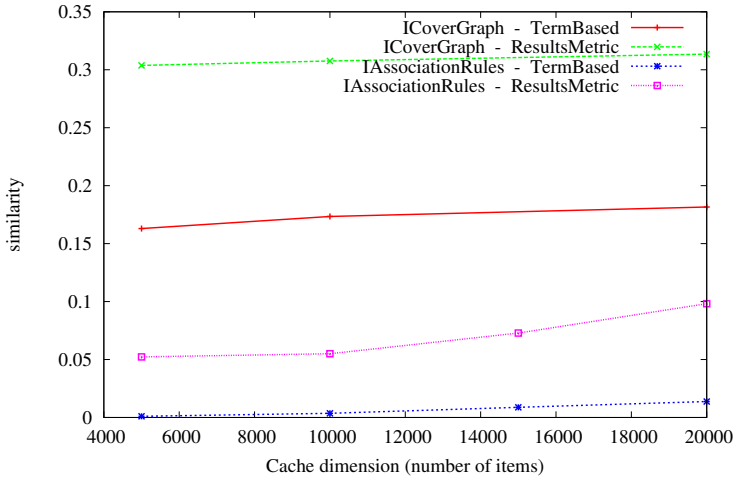


(b)

Fig. 5.7. LinkOmega for AssociationRules, CoverGraph, and IAssociationRules, ICoverGraph.



(a)



(b)

Fig. 5.8. Syntax-based metrics for AssociationRules, CoverGraph, and IAssociationRules, ICoverGraph as a function of the model dimensions.

We introduced a new class of query recommender algorithms that update *incrementally* the model on which recommendations are drawn. Starting from two state-of-the-art algorithms, we designed four new query recommender systems that continuously update their models as queries are issued. The two incremental algorithms differ from their static counterparts by the way in which they manage and use data to build the model. In addition, we proposed an automatic evalu-

ation mechanism based on four new metrics to assess the effectiveness of query recommendation algorithms.

The experimental evaluation conducted by using a large real-world query log shows that the incremental update strategy for the recommendation model yields better results for both coverage (more than 20% queries covered by both IAssociationRules, and ICoverGraph) and effectiveness due to the "fresh" data that are added to the recommendation models. Furthermore, this improved effectiveness is accomplished without compromising the efficiency of the query suggestion process.

6

Generating Suggestions for Queries in the Long Tail with an Inverted Index

This chapter proposes an efficient and effective solution to the problem of choosing the queries to suggest to web search engine users in order to help them in rapidly satisfying their information needs. By exploiting a weak function for assessing the similarity between the current query and the knowledge base built from historical users' sessions, we re-conduct the suggestion generation phase to the processing of a full-text query over an inverted index. The resulting query recommendation technique is very efficient and scalable, and is less affected by the data-sparsity problem than most state-of-the-art proposals. Thus, it is particularly effective in generating suggestions for rare queries occurring in the long tail of the query popularity distribution. The quality of suggestions generated is assessed by evaluating the effectiveness in forecasting the users' behavior recorded in historical query logs, and on the basis of the results of a reproducible user study conducted on publicly-available, human-assessed data. The experimental evaluation conducted shows that our proposal remarkably outperforms two other state-of-the-art solutions, and that it can generate useful suggestions even for rare and never seen queries.

6.1 Introduction

Giving suggestions to users of Web search engines is a common practice aimed at "*driving*" users toward the information bits they may need. Suggestions are usually provided as queries that are, to some extent, related to those recently submitted by the user. The generation process of such queries, basically, exploits the expertise of "skilled" users to help inexperienced ones. Besides, generating effective suggestions for user queries which are rare or have never been seen in the past is an open issue poorly addressed by state-of-the-art query suggestion techniques.

In a previous work, an interesting framework for the query suggestion problem is provided by the *Search Shortcut* model, and an evaluation metric for assessing the effectiveness of suggested queries by exploiting a query log are proposed [22].

Basically, the model formalizes a way of exploiting the knowledge mined from query logs to help users to *rapidly* satisfy their information need. In the same work the use of *Collaborative Filtering* (CF) algorithms is investigated. However, the work highlights some limitations in the query recommendations solutions based on collaborative filtering mainly due to the poor and very sparse scoring information available in query logs. In fact, due to the long-tail distribution of query occurrences, click information for low-frequency queries is rare and very sparse. Since implicit feedback information given by popularity and user clicks is the only source of (positive) query scoring available, most of the queries in the query log cannot be exploited to generate the recommendation model [3]. This issue affects CF-based solutions, but also many other query recommendation techniques discussed in the literature.

In this chapter we propose an efficient and effective query recommendation algorithm that can "*cover*" also queries in the long tail. We adopt the Search Shortcuts model and its terminology, and re-conduct the shortcut generation phase to the processing of a full-text query over an inverted file that indexes satisfactory user sessions recorded in a query log. Differently from most state-of-the art proposals, our shortcut generation algorithm aggregates implicit feedback information present in query logs at the level of single query terms, thus alleviating the data sparseness issue. The contribution of each query terms is then combined during the suggestion generation process in order to provide recommendations also for queries that are rare or even for those that were never seen in the past. Generating suggestions for rare queries is a hot research topic [108, 145, 44], and our suggestion generation technique beyond addressing the data-sparsity problem, is both very efficient and scalable, making it suitable for a large-scale deployment in real-world search engines.

Another contribution of this chapter consists in the methodology adopted for manually assessing the effectiveness of query suggestion techniques. The methodology exploits the query topics and the human judgements provided by the National Institute of Standards and Technology (NIST) for running the TREC Web Track's diversity task. For the purposes of the diversity task, the NIST assessors provide 50 queries, and, for each of them, they identify a representative set of subtopics, based on information extracted from the logs of a commercial search engine. We claim that *given a query topic A with all its subtopics $\{a_1, a_2, \ldots, a_n\}$, and a query suggestion technique $\mathcal{T}$, the more the queries suggested by $\mathcal{T}$ for A cover the human-assessed subtopics $\{a_1, a_2, \ldots, a_n\}$, the more $\mathcal{T}$ is effective.* To assess the effectiveness of a given query suggestion technique, we thus propose to simply ask human editors to count how many subtopics are actually covered by the suggestions generated by $\mathcal{T}$ for the TREC diversity track queries. This methodology is entirely based on a publicly-available data. It can be thus considered fair and constitute a good shared base for testing and comparing query recommendation systems. We shall define the above concept better in Section 6.4.

The experimental evaluation conducted shows that the solution proposed outperforms remarkably two state-of-the-art algorithms chosen for performance comparison purposes (presented in [17] and [35, 36]). Differently from these competitor algorithms, our solution generates in fact relevant suggestions for a vast majority of the 50 TREC queries, and the suggested queries covered a high percentage of possible subtopics. In particular we assessed that it can generate useful suggestions even for queries that are rare or do not occur in the query log used for training the recommendation model. Moreover, the proposed algorithm outperforms the competitor solutions even on the tests measuring the effectiveness in forecasting the users' behavior recorded in historical query according to the metric used in [22].

The main original contributions of this chapter are thus:

- a novel algorithm to efficiently and effectively generate query suggestions that is robust to data sparsity;
- a novel evaluation methodology with which we can compare the effectiveness of suggestion mechanisms;
- an extensive evaluation comparing on the same basis the proposed solution with two state-of-the-art algorithms.

The rest of the chapter is organized as follows. The next Section shows a brief overview of the state of the art in query recommendation. Section 6.3 briefly sketches the shortcuts model and describes the efficient algorithm designed for generating query shortcuts. The evaluation methodology based on the TREC diversification track data is discussed in Section 6.4 which also presents the encouraging results obtained by our solution in the performance comparisons tests conducted. Finally, Section 6.5 draws some conclusions and outlines future work.

6.2 Related Work

The problem addressed in this chapter is related to two related research fields that have been traditionally addressed from different points of view. We are talking about query suggestion algorithms and recommender systems.

Recommender systems are used in several domains, being specially successful in electronic commerce. They can be divided in two broad classes: those based on *content filtering*, and those on *collaborative filtering*. As the name suggests, content filtering approaches base their recommendations on the content of the items to be suggested. They face serious limitations when dealing with multimedia content and, more importantly, their suggestions are not influenced by the human-perceived *quality* of contents. On the other side, collaborative filtering solutions are based on the preferences of other users. There are two main families of collaborative filtering algorithms: memory-based and model-based. Memory-based approaches use the whole past usage data to identify similar users [136], items [134], or both [164]. Generally, memory-based algorithms are quite simple and produce

good recommendations, but they usually face serious scalability problems. On the other hand, model-based algorithms construct in advance a model to represent the behavior of users, allowing to predict more efficiently their preferences. However, the model building phase can be highly time-consuming, and models are generally hard to tune, sensitive to data changes, and highly dependent on the application domain. Different approaches can be adopted based on linear algebra methods [48, 125], clustering [157], latent class models [78], SVD models [117]. An analysis of the use of CF algorithms to the query suggestion problem can be found in [22], where the problem descending from the poor and very sparse scoring information available in query logs is highlighted.

On the other side, query suggestion techniques address specifically the problem of recommending queries to WSE users, and propose specific solutions and specific evaluation metrics tailored to the Web search domain. Techniques proposed during last years are very different, yet they have in common the exploitation of usage information recorded in query logs [142]. Many approaches extract the information used from the plain set of queries recorded in the log, although there are several works that take into account the chains of queries that belong to the same search session [122]. In the first category we have techniques that employ clustering algorithms to determine groups of related queries that lead users to similar documents [165, 18, 26]. The most "representative" queries in the clusters are then returned as suggestions. Others solutions employ the reformulations of the submitted query issued by previous users [93], or propose as suggestions the frequent queries that lead in the past users to retrieve similar results [19].

Baeza-Yates and Tiberi [17] exploit click-through data as a way to provide recommendations. The method is based on the concept of *Cover Graph* (CG). A CG is a bipartite graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u that was an answer for the query. Suggestions for a query q are thus obtained by accessing the corresponding node in the CG and by extracting the related queries sharing more URLs. The sharing of clicked URLs results to be very effective for devising related queries, and the CG solution has been chosen as one of the two query suggestion algorithms considered in this chapter for experimental performance comparison.

Among the proposals exploiting the chains of queries stored in query logs, [69] use an association rule mining algorithm to devise frequent query patterns. These patterns are inserted in a query relation graph which allows "concepts" (queries that are synonyms, specializations, generalizations, etc.) to be identified and suggested.

Boldi *et al.* introduce the concept of *Query Flow Graph* (QFG), an aggregated representation of the information contained in a query log [35]. A QFG is a directed graph in which nodes are queries, and the edge connecting node q_1 to q_2 is weighted by the probability that users issue query q_2 after issuing q_1. Authors highlight the utility of the model in two concrete applications, namely, *devising logical sessions*

and *generating query recommendation*. The authors refine the previous studies in [36] and [37] where a query suggestion scheme based on a random walk with restart model on the QFG is proposed. Such QFG-based suggestion algorithm is the second algorithm considered in this chapter for experimental performance comparison.

A approach similar to our is represented by the query refinement/substitution technique discussed in [93]. The goal of query refinement is to generate a new query to replace a user's original ill-formed search query in order to enhance the relevance of retrieved results. The technique proposed includes a number of tasks such as spelling error correction, word splitting, word merging, phrase segmentation, word stemming, and acronym expansion. Our approach instead aims at suggesting users a set of queries that better specify their information needs.

The importance of rare query classification and suggestion recently attracted a lot of attention from the IR community. Generating suggestions for rare queries is in fact very difficult due to the lack of information in the query logs.

Downey *et al.* [65] describe search log studies aiming at explaining behaviors associated with rare and common queries. They investigate the search behavior following the input of rare and common queries. Results show that search engines perform less well on rare queries. The authors also study transitions between rare and common queries highlighting the difference between the frequency of queries and their related information needs.

Yang *et al.* [145] propose an optimal rare query suggestion framework by leveraging implicit feedbacks from users in the query logs. The proposed model is based on the pseudo-relevance feedback. It assumes that clicked and skipped URLs contain different level of information, and thus, they should be treated differently. Therefore, the framework optimally combines both click and skip information from users, and uses a random walk model to optimize i) the restarting rate of the random walk, and ii) the combination ratio of click and skip information. Experimental results on a log from a commercial search engine show the superiority of the proposed method over the traditional random walk models and pseudo-relevance feedback models.

Mei *et al.* propose a novel query suggestion algorithm based on ranking queries with the hitting time on a large scale bipartite graph [108]. The rationale of the method is to capture semantic consistency between the suggested queries and the original query. Empirical results on a query log from a real world search engine show that hitting time is effective to generate semantically consistent query suggestions. The authors show that the proposed method and its variations are able to boost long tail queries, and personalized query suggestion.

Broder *et al.* propose to leverage the results from search engines as an external knowledge base for building the word features for rare queries [45]. The authors train a classifier on a commercial taxonomy consisting of 6,000 nodes for categorization. Results show a significant boost in term of precision with respect to

the baseline query expansion methods. Lately, Broder *et al.* propose an efficient and effective approach for matching ads against rare queries [44]. The approach builds an expanded query representation by leveraging offline processing done for related popular queries. Experimental results show that the proposed technique significantly improves the effectiveness of advertising on rare queries with only a negligible increase in computational cost.

The idea we present in this chapter follows a completely new approach. First, we infer the relevance of a query based on whether it successfully ended a search session, i.e., the *last query* of the user session allowed the user to find the information she was looking for. "*Successful sessions*" have already been taken into account as a way to evaluate promotions of search results [144, 143]. Similarly, "*satisfactory sessions*" are considered in this chapter as the key factor for generating useful query recommendations. All the queries in the satisfactory sessions stored in the log which terminate with the same final query are considered "related", since it is likely that these queries were issued by different users trying to satisfy a similar information need. Thus, our technique exploit a sort of collaborative clustering of queries inferred from successful user search processes, and suggest users the final queries which are the representatives of the clusters closest to the submitted query.

6.3 An Efficient Algorithm for the Query Shortcuts Problem

In the following we briefly recall the basis of the *Search Shortcuts Problem* (SSP) proposed in [22], and we introduce our novel shortcuts generation algorithm.

6.3.1 The Search Shortcuts Problem

The SSP is formally defined as a problem related to the recommendation of queries in search engines and the potential reductions obtained in the users session length. This problem formulation allows a precise goal for query suggestion to be devised: *recommend queries that allowed "similar" users, i.e., users which in the past followed a similar search process, to successfully find the information they were looking for*. The problem has a nice parallel in computer systems: *prefetching*. Similarly to prefetching, search shortcuts anticipate requests to the search engine with suggestion of queries that a user would have likely issued at the end of her session.

We now introduce the notations and we recap the formal definition of the SSP. Let $\mathcal{U}$ be the set of users of a WSE whose activities are recorded in a query log QL, and $\mathcal{Q}$ be the set of queries in QL. We suppose QL is preprocessed by using some session splitting method (e.g. [92, 101]) in order to extract query *sessions*, i.e., sequences of queries which are related to the same user search task. Formally, we denote by $\mathcal{S}$ the set of all sessions in QL, and σ^u a session issued by user u. Moreover, let us denote with σ^u_i the i-th query of σ^u. For a session σ^u of length

n its **final query** is the query σ_n^u, i.e. the last query issued by u in the session. To simplify the notation, in the following we will drop the superscript u whenever the user u is clear from the context.

We say that a session σ is ***satisfactory*** if and only if the user has clicked on at least one link shown in the result page returned by the WSE for the final query σ_n, ***unsatisfactory*** otherwise.

Finally, given a session σ of length n we denote $\sigma_{t|}$ the **head** of σ, i.e., the sequence of the first t, $t < n$, queries, and $\sigma_{|t}$ the **tail** of σ given by the sequence of the remaining $n - t$ queries.

Definition 1. *We define **k-way shortcut** a function h taking as argument the head of a session $\sigma_{t|}$, and returning as result a set $h\left(\sigma_{t|}\right)$ of k queries belonging to Q.*

Such definition allows a simple ex-post evaluation methodology to be introduced by means of the following similarity function:

Definition 2. *Given a satisfactory session of length n $\sigma \in \mathcal{S}$, and a k-way shortcut function h, the similarity between $h\left(\sigma_{t|}\right)$ and a tail $\sigma_{|t}$ is defined as:*

$$s\left(h\left(\sigma_{t|}\right), \sigma_{|t}\right) = \frac{\sum_{q \in h(\sigma_{t|})} \sum_{m=1}^{n-t} [\![q = \left(\sigma_{|t}\right)_m]\!] f(m)}{|h(\sigma_{t|})|} \tag{6.1}$$

where $f(m)$ is a monotonic increasing function, and function $[\![q = \sigma_m]\!] = 1$ if and only if q is equal to σ_m.

For example, to evaluate the effectiveness of a given shortcut function h, the sum (or average) of the value of s computed on all satisfactory sessions in $\mathcal{S}$ can be computed.

Definition 3. *Given the set of all possible shortcut functions $\mathcal{H}$, we define **Search Shortcut Problem** (SSP) the problem of finding a function $h \in \mathcal{H}$ which maximizes the sum of the values computed by Equation (6.1) on all satisfactory sessions in $\mathcal{S}$.*

A difference between search shortcuts and query suggestion is actually represented by the function $[\![q = \left(\sigma_{|t}\right)_m]\!]$ in Equation (6.1). By relaxing the strict *equality* requirement, and by replacing it with a similarity relation – i.e., $[\![q \sim \left(\sigma_{|t}\right)_m]\!] = 1$ if and only if the similarity between q and σ_m is greater than some threshold – the problem reduces, basically, to query suggestion. By defining appropriate similarity functions, the Equation in (6.1) can be thus used to evaluate query suggestion effectiveness as well.

Finally, we should consider the influence the function $f(m)$ has in the definition of scoring functions. Actually, depending on how f is chosen, different features of

a shortcut generating algorithm may be tested. For instance, by setting $f(m)$ to be the constant function $f(m) = c$, we measure simply the number of queries in common between the query shortcut set and the queries submitted by the user. A non-constant function can be used to give an higher score to queries that a user would have submitted later in the session. An exponential function $f(m) = e^m$ can be exploited instead to assign an higher score to shortcuts suggested early. Smoother f functions can be used to modulate positional effects.

6.3.2 The Search Shortcuts Generation Method

Inspired by the above SSP, we define a novel algorithm that aims to generate suggestions containing *only* those queries appearing as final in satisfactory sessions. The goal is to suggest queries having a high potentiality of being useful for people to reach their initial goal. As hinted by the problem definition, suggesting queries appearing as final in satisfactory sessions, in our view is a good strategy to accomplish this task. In order to validate this hypothesis, we analyzed the Microsoft RFP 2006 dataset, a query log from the MSN Search engine containing about 15 million queries sampled over one month of 2006 (hereinafter QL).

First, we measured that the number of distinct queries that appear as final query in satisfactory sessions of QL is relatively small if compared to the overall number of submitted queries: only about 10% of the total number of distinct queries in QL occur in the last position of satisfactory user sessions. As expected, the distribution of the occurrences of such final queries in satisfactory user sessions is very skewed (as shown in Figure 6.1), thus confirming once more that the set of final queries *actually* used by people is limited.

Queries which are $final$ in some satisfactory sessions may obviously appear also in positions different from the last in other satisfactory sessions. We verified that, when this happens, these queries appear much more frequently in positions very close to the final one. About 60% of the distinct queries appearing in the penultimate position of satisfactory sessions are also among the final queries, about 40% in positions second to the last, 20% as third to the last, and so on. We can thus argue that $final$ queries are usually *close* to the achievement of the user information goal. We consider these queries as highly valued and high quality short pieces of text expressing actual user needs.

The SSP algorithm proposed in this chapter works by computing, efficiently, similarities between partial user sessions (the one currently performed) and historical satisfactory sessions recorded in a query log. Final queries of most similar satisfactory sessions are suggested to users as search shortcuts.

Let σ' be the current session performed by the user, and let us consider the sequence τ of the concatenation of all terms with possible repetitions appearing in $\sigma'_{t|}$, i.e. the head of length t of session σ'. We now compute the value of a scoring function $\delta(\tau, \sigma^s)$, which for each satisfactory session measures the similarity between its queries and the set of terms τ. Intuitively, this similarity measures how

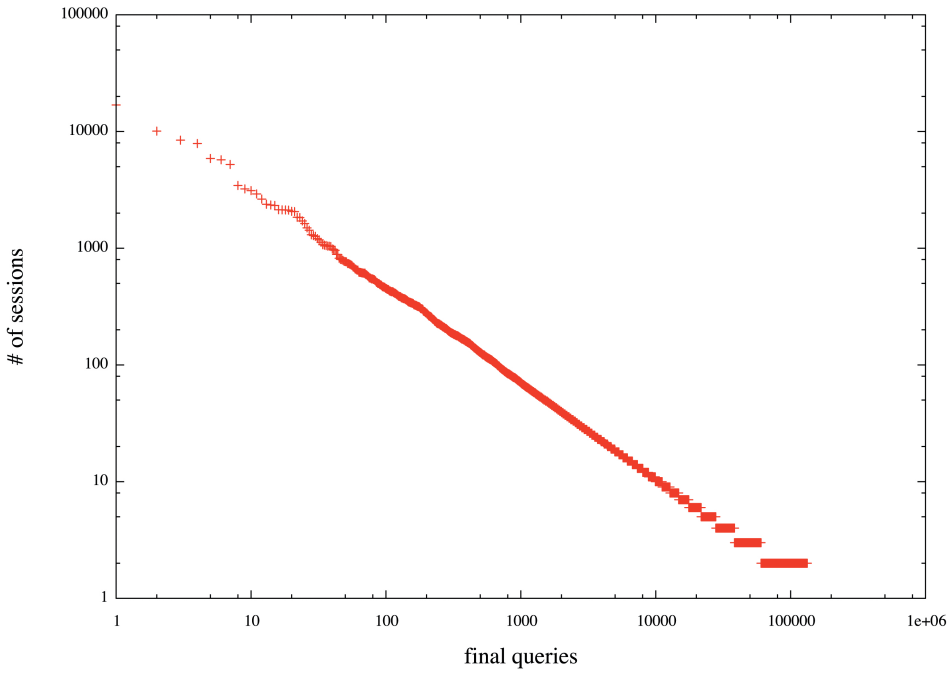


Fig. 6.1. Popularity of final queries in satisfactory sessions.

much a previously seen session overlaps with the user need expressed so far (the concatenation of terms τ serves as a bag-of-words model of user need). Sessions are ranked according to δ scores and from the subset of the top ranked sessions we suggest their final queries. It is obvious that depending on how the function δ is chosen we may have different recommendation methods. In our particular case, we opt for δ to be the similarity computed as in the BM25 metrics [128]. We opt for an IR-like metric because we want to take into much consideration words that are discriminant in the context of the session to which we are comparing. BM25, and other IR-related metrics, have been designed specifically to account for that property in the context of query/documents similarity. We borrow from BM25 the same attitude to adapt to this conditions. The shortcuts generation problem has been, thus, reduced to the information retrieval task of finding highly similar sessions in response to a given sequence of queries. In our experiments, we compute the similarity function δ only on the current query issued by the user instead of using the whole head of the session. We do this in order to be fair with respect to our competitors as they produce recommendations starting from a single query. We leave the study of the use of the whole head of the session for producing query recommendations as a future work.

The idea described above is thus translated into the following process. For each unique "*final query*" q_f contained in satisfactory sessions we define what we have

called a *virtual document* identified by its *title* and its *content*. The title, i.e. the identifier of the document, is exactly query string q_f. The content of the virtual document is instead composed of all the terms that have appeared in queries of all the satisfactory sessions ending with q_f. At the end of this procedure we have a set of virtual documents, one for each distinct final query occurring in some satisfactory sessions. Just to make things more clear, let us consider a toy example. Consider the two following satisfactory sessions: (*gambling*, *gambling places*, *las vegas*, *bellagio*), and (*las vegas*, *strip*, *las vegas hotels*, *bellagio*). We create the virtual document identified by title **bellagio** and whose content is the text (*gambling gambling places las vegas las vegas strip las vegas hotels*). As you can see the virtual document actually contains also repetitions of the same term that are considered in the context of the BM25 metrics. All virtual documents are indexed with the preferred Information Retrieval system, and generating shortcuts for a given user session σ' becomes simply processing the query $\sigma'_{t|}$ over the inverted file indexing such virtual documents. We know that processing queries over inverted indexes is very fast and scalable, and these important characteristics are inherited by our query suggestion technique as well.

The other important feature of our query suggestion technique is its robustness with respect to rare and singleton queries. Singleton queries account for almost 50% of the submitted queries [142], and their presence causes the issue of the sparsity of models [3]. Since we match τ with the text obtained by concatenating all the queries in each session, we are not bound to look for previously submitted queries as in the case of other suggestion algorithms. Therefore we can generate suggestions for queries in the long tail of the distribution whose terms have some context in the query log used to build the model.

6.4 Assessing Search Shortcuts Quality

The effectiveness of a query recommender systems can be evaluated by means of *user-studies* or through the adoption of some performance metrics. Unfortunately, both these methodologies may lack of generality and incur in the risk of being over-fitted on the system object of the evaluation. The evaluation methodology used in this chapter tries to address pragmatically the above issues.

For what concerns the methodology based on a performance metrics, we used the one defined in Equation (6.1), and we computed the average value of similarity over a set of satisfactory sessions. This performance index *objectively* measures the effectiveness of a query suggestion algorithm in foreseeing the satisfactory query for the session.

In particular, we measured the values of this performance index over suggestions generated by using our *Search Shortcuts* (SS) solution and by using in exactly the same conditions two other state-of-the-art algorithms: *Cover Graph* (CG) proposed in [17], and *Query Flow Graph* (QFG), proposed in [36]. These algorithms

are recent and highly reputed representatives of the best practice in the field of query recommendation. To test QFG-based query suggestion we used the original implementation kindly provided us by the authors. In the case of CG, instead, we evaluate our own implementation of the technique.

For what concerns the methodology based on user-studies, we propose a approach that measures *coverage* and the *effectiveness* of suggestions against a manually assessed and publicly available dataset.

To this purpose, we exploited the query topics and the human judgements provided by NIST for running the TREC 2009 Web Track's Diversity Task[1]. For the purposes of the TREC diversity track, NIST provided 50 queries to a group of human assessors. Assuming each TREC query as a topic, assessors were asked to identify a representative set of subtopics covering the whole spectrum of different user needs/intentions. Subtopics are based on information extracted from the logs of a commercial search engine, and are roughly balanced in terms of popularity. Obviously the queries chosen are very different and from different categories: difficult, ambiguous, and/or faceted in order to allow the overall performance of diversification methods to be evaluated and compared. Since diversity and topic coverage are key issues also for the query recommendation task [102], we propose to use the same third-party dataset for evaluating query suggestion effectiveness as well.

Let's now introduce the definitions of *coverage*, and *effectiveness*.

Definition 4 (Coverage). *Given a query topic A with subtopics $\{a_1, a_2, \ldots, a_n\}$, and a query suggestion technique $\mathcal{T}$, we say that $\mathcal{T}$ has coverage equal to c if $n \cdot c$ subtopics match suggestions generated by $\mathcal{T}$.*

In other words, a coverage of 0.8 for the top-10 suggestions generated for a query q having 5 subtopics means that 4 subtopics of q are covered by at least one suggestion.

Definition 5 (Effectiveness). *Given a query topic A with subtopics $\{a_1, a_2, \ldots, a_n\}$, and a query suggestion technique $\mathcal{T}$ generating k suggestions, we say that $\mathcal{T}$ has effectiveness equal to e if $k \cdot e$ suggestions cover at least one subtopic.*

In other words, an effectiveness of 0.1 on the top-10 suggestions generated for a query q means that only one suggestion is relevant for one of the subtopics of q.

The methodology just described has some net advantages. It is based on a publicly-available test collection which is provided by a well reputed third-party organization. Moreover, it grants to all the researchers the possibility of measuring the performance of their solution under exactly the same conditions, with the same dataset and the same reproducible evaluation criterium. In fact, even though the matching between suggestions and topics is still human-driven the process has a very low ambiguity as we shall discuss in the next section.

[1] `http://trec.nist.gov/data/web09.html`

Query and its subtopics
TREC query (n. 8):***appraisal***
S1: What companies can give an appraisal of my home's value?
S2: I'm looking for companies that appraise jewelry.
S3: Find examples of employee performance appraisals.
S4: I'm looking for web sites that do antique appraisals.
SS
performance appraisal (**S3**)
hernando county property appraiser (**S1**)
antique appraisal (**S4**)
appraisers in colorado (**S1**)
appraisals etc (**S1**)
appraisers.com (**S4**)
find appraiser (**S1**)
wachovia bank appraisals (**S1**)
appraisersdotcom (**S4**)
QFG
online appraisals (**S4**)
CG
appraisersdotcom (**S4**)
employee appraisals (**S3**)
real estate appraisals (**S1**)
appraisers (**S1**)
employee appraisals forms (**S3**)
appraisers.com (**S4**)
gmac
appraisers beverly wv (**S1**)
picket fence appraisal (**S1**)
fossillo creek san antonio

Table 6.1. An example of the coverage evaluating process involving the TREC dataset. For the 8^{th} TREC query ***appraisal***, one of the assessors evaluates the coverage of suggestions generated by SS, QGF, and CG. The subtopics covered by each suggestion are reported in bold between parentheses. Suggestions not covering any of the subtopics are emphasized.

6.4.1 Experimental Settings

The experiments were conducted using the Microsoft RFP 2006 query log which was preliminary preprocessed by converting all queries to lowercase, and by removing stop-words and punctation/control characters.

The queries in the log were then sorted by user and timestamp, and segmented into sessions on the basis of a splitting algorithm which simply groups in the same session all the queries issued by the same users in a time span of 30 minutes. We tested also the session splitting technique based on the Query Flow Graph proposed in [35], but for the purpose of our technique, we did not observe a significant variation in terms of quality of the generated suggestions.

Noisy sessions, likely performed by software robots, were removed. The remaining entries correspond to approximately 9M sessions. These were split into two subsets: training set with 6M sessions and a test set with the remaining 3M sessions. The training log was used to build the recommendation models needed by CG and QFG and used for performance comparison.

Instead, to implement our SS solution we extracted satisfactory sessions present in the training log and grouped them on the basis of the final query. Then, for each distinct final query its corresponding *virtual document* was built with the terms (with possible repetitions) belonging to all the queries of all the associated satisfactory sessions. Finally, by means of the Terrier search engine[2], we indexed the resulting $1,191,143$ virtual documents. The possibility of processing queries on such index is provided to interested readers through a simple web interface available at the address `http://searchshortcuts.isti.cnr.it`. The web-based wrapper accepts user queries, interact with Terrier to get the list of final queries (id of virtual documents) provided as top-k results, and retrieves and visualizes the associated query strings.

6.4.2 TREC queries statistics

We measured the popularity of the 50 TREC 2009 Web Track's Diversity Task queries in the training log obtained by the Microsoft RFP 2006 dataset as described in the previous section. Figure 6.2 shows the cumulative frequency distribution of the 50 TREC queries. While 8/50 queries are not present in the training log, 2/50 queries occur only one time. Furthermore, 23/50 queries have a frequency lower than 10 and 33/50 queries occur lower than 100 times. The TREC dataset thus contains a valid set of queries for evaluating the effectiveness of our method as it includes several examples of unseen and rare queries, while popular queries are represented as well. Table 6.2 shows some queries with their popularity measured in the training log.

TREC query	Frequency
wedding budget calculator	0
flame designs	1
dog heat	2
the music man	5
diversity	27
map of the united states	170
cell phones	568
starbucks	705

Table 6.2. An example of eight TREC queries with their relative frequency in the training log.

[2] `http://terrier.org/`

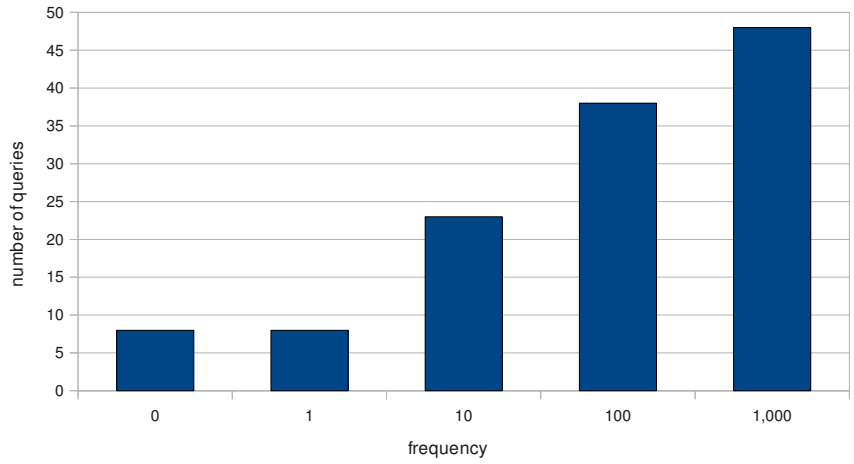


Fig. 6.2. Histogram showing the total number of TREC queries (on the y axis) having at most a certain frequency (on the x axis) in the training log. For instance, the third bar shows that 23 TREC queries out of 50 occur at most ten times in the training log.

6.4.3 Search Shortcuts metric

We used Equation (6.1) to measure the similarity between the suggestions generated by SS, CG, and QFG for the first queries issued by a user during a satisfactory session belonging to the test set, and the final queries actually submitted by the same user during the same session. We conducted experiments by setting the number k of suggestions generated to 10, and, as in [22], we chose the exponential function $f(m) = e^m$ to assign an higher score to shortcuts suggested early. Moreover, the length t of the head of the session was set to $\lceil n/2 \rceil$, where n is the length of the session considered. Finally, the metric used to assess the similarity between two queries was the Jaccard index computed over the set of tri-grams of characters contained in the queries [89], while the similarity threshold used was 0.9.

Due to the long execution times required by CG, and QFG for generating suggestions, it was not possible to evaluate suggestion effectiveness by processing all the satisfactory sessions in the test set. We thus considered a sample of the test set constituted by a randomly selected group of 100 satisfactory sessions having a length strictly greater than 3. The histogram in Figure 6.3 plots the distribution of the number of sessions vs. the quality of the top-10 recommendations produced by the three algorithms. Results in the plot are grouped by quality range. As an example, the second group of bars shows the number of sessions for which the three algorithms generated suggestions having a quality (measured using Equation (6.1)) ranging from from 0.1 to 0.2. Results show that recommendations produced for the majority of sessions by QFG and CG obtains a quite low score (in the interval

between 0 to 0.1), while SS produces recommendations whose quality is better distributed among all the range.

In particular, SS produces recommendations having a quality score greater than 60% for 18 sessions out of 100. Moreover, in 36 cases out of 100, SS generates useful suggestions when its competitors CG and QFG fails to produce even a single effective suggestion. On average, over the 100 sessions considered, SS obtains an average quality score equal to 0.32, while QFG and CG achieves 0.15 and 0.10, respectively.

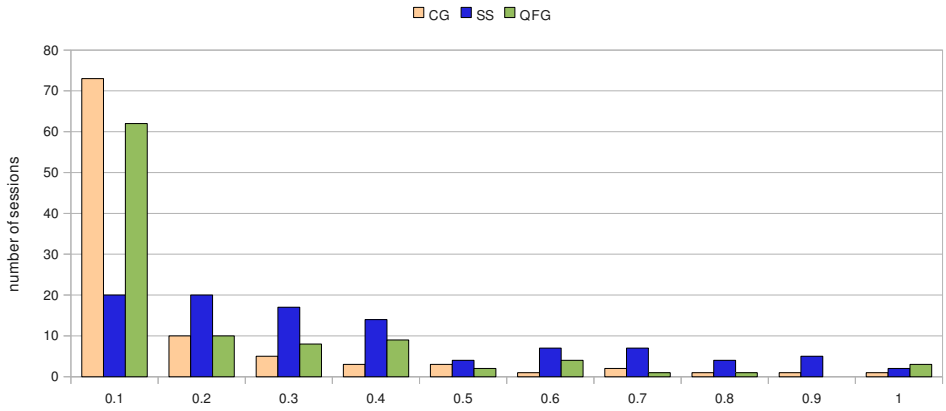


Fig. 6.3. Distribution of the number of sessions vs. the quality of the top-10 recommendations produced by the three algorithms.

6.4.4 Suggestions Quality on TREC topics

The relevance of the suggestions generated by SS, CG, and QFG w.r.t. the TREC query subtopics was assessed manually[3]. Seven volunteers were chosen among CS researchers working in different research groups of our Institute. The evaluation consisted in asking assessors to assign, for any given TREC query, the top-10 suggestions returned by SS, CG, and QFG to their related subtopic. Editors were also able to explicitly highlight that no subtopic can be associated with a particular recommendation. The evaluation process was blind, in the sense that all the suggestions produced by the three methods were presented to editors in a single, lexicographically ordered sequence where the algorithm which generated any specific suggestion was hidden. Given the limited number of queries and the precise definition of subtopics provided by NIST assessors, the task was not particularly

[3] All the queries suggested by the three algorithms for the 50 TREC queries are available to the interested reader along with the associated subtopic lists at the address `http://searchshortcuts.isti.cnr.it/TREC_results.html`.

cumbersome, and the evaluations generated by the assessors largely agree. Table 6.1 shows the outcome of one of the editors for the TREC query n. 8. The note in bold after each suggestion indicates the subtopic to which the particular suggestion was assigned (e.g. "*employee appraisals*" in the CG column matches subtopic S3). Thus for this topic this editor gave to both SS and CG a coverage of 3/4 (3 subtopics covered out of 4), while QFG was rated 1/4. Moreover, suggestions in italic, e.g. *gmac* in the CG column, were considered by the editor not relevant for any of the subtopics. Thus, for topic "*appraisals*" SS and QFG score an effectiveness equal to 1 (all suggestions generated were considered relevant), whereas CG score was 4/5 (2 non relevant suggestions out of 10).

The histogram shown in Figure 6.4 plots, for each of the 50 TREC topics, the average coverage (Definition 4) of the associated subtopics measured on the basis of assessor's evaluations for the top-10 suggestions returned by SS, CG, and QFG. By looking at the Figure, we can see that SS outperforms remarkably its competitors. On 36 queries out of 50 SS was able to cover at least half of the subtopics, while CG only in two cases reached the 50% of coverage, and QFG on 8 queries out of 50. Moreover, SS covers the same number or more subtopics than its competitors in all the cases but 6. Only in 5 cases QFG outperforms SS in subtopic coverage (query topics 12, 15, 19, 25, 45), while in one case (query topic 22) CG outperforms SS. Furthermore, while SS is always able to cover one or some subtopics for all the cases, in 15 (27) cases for QFG (CG) the two methods are not able to cover any of the subtopics. The average fraction of subtopics covered by the three methods is: 0.49, 0.24, and 0.12 for SS, QFG, and CG, respectively.

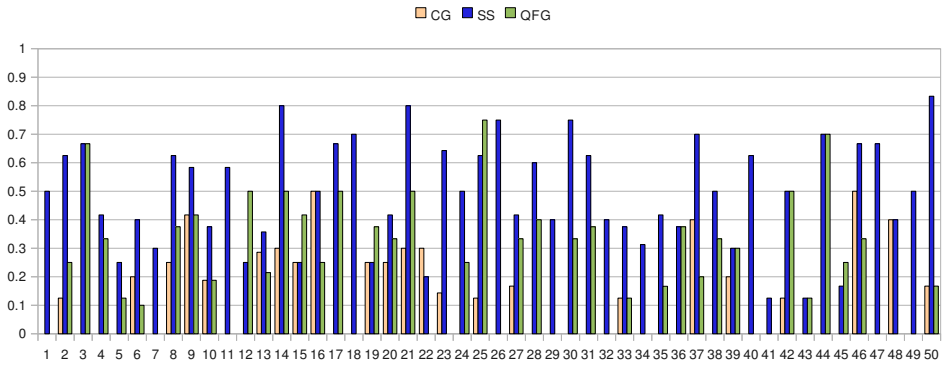


Fig. 6.4. Coverage of the subtopics associated with the 50 TREC diversity-track queries measured by means of an user-study on the top-10 suggestions provided by the Cover Graph (CG), Search Shortcuts (SS), and Query Flow Graph (QFG) algorithms.

Figure 6.5 reports the effectiveness (Definition 5) of the top-10 suggestions generated by SS, QFG, and CG. Also considering this performance metric our Search Shortcuts solution results the clear winner. SS outperforms its competitors

in 31 cases out of 50. The average effectiveness is 0.83, 0.43, and 0.42 for SS, QFG, and CG, respectively. The large difference measured is mainly due to the fact that both CG and QFG are not able to generate good suggestions for queries that are not popular in the training log.

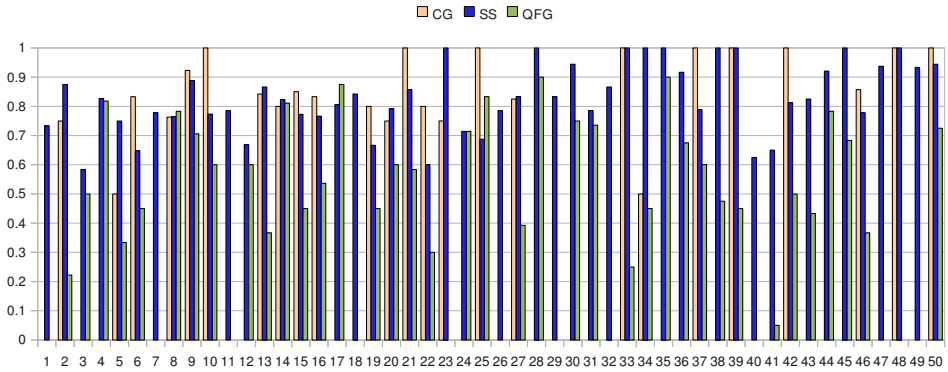


Fig. 6.5. Effectiveness measured on the TREC query subtopics among the top-10 suggestions returned by the Cover Graph (CG), Search Shortcuts (SS), and Query Flow Graph (QFG) algorithms.

Regarding this aspect, the histogram in Figure 6.6 shows the average effectiveness of the top-10 suggestions returned by SS, CG and QFG measured for groups of TREC queries arranged by their frequency in the training log. SS remarkably outperforms its competitors. SS is in fact able to produce high-quality recommendations for all the categories of query analyzed, while CG and QFG can not produce recommendations for unseen queries. Furthermore, while SS produce constant quality recommendations with respect to the frequency of the TREC queries, CG and QFG show an increasing trend in the quality of recommendations as the frequency of the TREC queries increases.

For this reason, we can assert that *the SS method is very robust to data sparsity which strongly penalizes the other two algorithms, and is able to effectively produce significant suggestions also for singleton queries which were not previously submitted to the WSE*. We recall that singleton queries account for almost half of the whole volume of unique queries submitted to a WSE, and are often the hardest to answer since they ask for "rare" or badly expressed information needs. The possibility of suggesting relevant alternatives to these queries is more valuable than the one of suggesting relevant alternatives to frequent queries, which express common and often easier to satisfy needs.

Just to give an example of the results we obtained and the data on which we evaluated the quality, Table 6.3 reports the top-10 suggestions provided by SS, CG, and QFG for some TREC Web Track's diversity task query topics. For each query

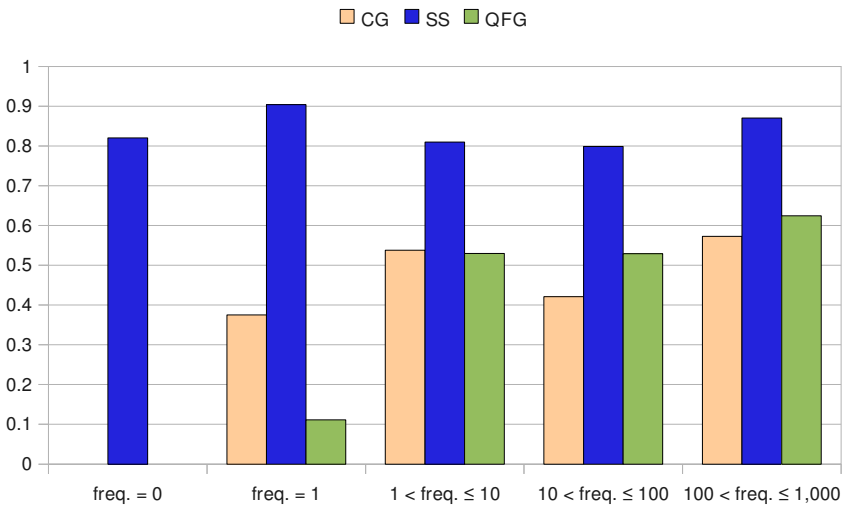


Fig. 6.6. Average effectiveness of the top-10 suggestions provided by the Cover Graph (CG), Search Shortcuts (SS), and Query Flow Graph (QFG) algorithms for groups of TREC queries arranged by their frequency (freq.) in the training log.

topic, the first column of the table lists the associated subtopics. These examples are representative of the figures above discussed: SS computed mostly relevant suggestions covering a significant subset of the subtopics. CG, on the other hand, performed worst and returned three suggestions only for a single query among the five reported in the table, and one single suggestion in another case. QFG returned instead 10 suggestions for three topics, and no suggestions in two cases.

Queries and their subtopics	SS	CG	QFG
TREC query (n. 18): ***wedding budget calculator*** (faceted); Not present in the training log;			
1. I want to find online guides, tips, and checklists for planning a wedding. 2. I am looking for spreadsheets or templates to help me tabulate a budget for a wedding. 3. I want to find some example wedding budgets. 4. I'm looking for information on planning a wedding shower, like theme ideas and budget guidelines. 5. How can I plan an inexpensive wedding?	budget wedding budget sheet sample wedding budgets budget calculator budget outside wedding wedding planning budget wedding bouquets how to have a celebrity wedding on a budget planning a wedding on a budget wedding costs	*no suggestion provided*	*no suggestion provided*
TREC query (n. 49): ***flame designs*** (ambiguous); Frequency in the training log: 1;			
1. Find free flame design clipart I can use on a website. 2. How do I make realistic flame images using Photoshop? 3. I'm looking for good flame tattoo designs. 4. Find pictures of flames and fire. 5. I want to find flame design decals I can put on my car or motorcycle. 6. I'm looking for flame design stencils.	flames flame fonts penny flame flame illustration flaming text flaming fonts tattoo flame designs flame pattern comforters in flames band flame art	*no suggestion provided*	*no suggestion provided*
TREC query (n. 50): ***dog heat*** (ambiguous); Frequency in the training log: 2;			
1. What is the effect of excessive heat on dogs? 2. What are symptoms of heat stroke and other heat-related illnesses in dogs? 3. Find information on dogs' reproductive cycle. What does it mean when a dog is in heat?	heat heat cycle of a dog can a dog be spayed in heat dogs heat dog in heat symptoms do female dogs howl in heat cycle dog heat exhaustion signs of a female dog in heat heat cycle of dogs when do dauschound go into heat	female dogs in heat how long how often do dogs come in heat female dogs in heat	dogs in heat do female dogs howl in heat cycle how to tell if your dog is pregnant can a dog be spayed in heat memphis spaying pet vacs pet vax myspace yspace myspac
TREC query (n. 42): ***the music man*** (faceted); Frequency in the training log: 5;			
1. Find lyrics for songs from The Music Man. 2. Find current performances of The Music Man. 3. Find recordings of songs from The Music Man. 4. I'm looking for the script for The Music Man.	till there was you musical music man lyrics the music man soundtrack the music man summary elephant man music 70s music rubberband man encino man songs music man lyrics free music on msn music man trouble in river city	the music man movie	the music man on broadway the music man summary state fair musical till there was you dizzy gilelespee oysters rockefeller recipe archnid female whale brewski fats dominos first name
TREC query (n. 24): ***diversity*** (faceted); Frequency in the training log: 27;			
1. How is workplace diversity achieved and managed? 2. Find free activities and materials for running a diversity training program in my office. 3. What is cultural diversity? What is prejudice? 4. Find quotes, poems, and/or artwork illustrating and promoting diversity.	diversity in education diversity inclusion cultural diversity diversity test accepting diversity diversity poem diversity skills diverse learners presentation picture of diverse childern advantages of diversity	*no suggestion provided*	accepting diversity dispariging remarks diverse world diversity director diversity poem diversity test minority & women civil liberties inclusion gender and racial bias

Table 6.3. Query suggestions provided by Search Shortcuts, Cover Graph, and Query Flow Graph for some TREC diversity-track query topics.

6.5 Summary

We proposed a very efficient solution for generating effective suggestions to WSE users based on the model of *Search Shortcut*. Our original formulation of the problem allows the query suggestion generation phase to be re-conducted to the processing of a full-text query over an inverted index. The current query issued by the user is matched over the inverted index, and final queries of the most similar satisfactory sessions are efficiently selected to be proposed to the user as query shortcuts. The way a satisfactory session is represented as a virtual document, and the IR-based technique exploited, allow our technique to generate in many cases effective suggestions even to rare or not previously seen queries. The presence of *at least* one query term in at least a satisfactory session used to build our model, permits in fact SS to be able to generate *at least* a suggestion.

By using the automatic evaluation approach based on the metric defined in Equation (6.1), SS outperforms QFG in quality of a 0.17, while the improvement over CG was even greater (0.22). In 36 evaluated sessions out of 100, SS generates useful suggestions when its competitors CG and QFG fails to produce even a single useful recommendation.

An additional contribution of the chapter regards the evaluation methodology used, based on a publicly-available test collection provided by a highly reputed organization such as the NIST. The proposed methodology is objective and very general, and, if accepted in the query recommendation scientific community, it would grant researchers the possibility of measuring the performance of their solution under exactly the same conditions, with the same dataset and the same evaluation criterium.

On the basis of the evaluation conducted by means of the user-study, SS remarkably outperforms both QFG and CG in almost all the tests conducted. In particular, suggestions generated by SS covers the same number or more TREC subtopics than its two counterparts in 44 cases out of 50. In 36 cases the number of subtopics covered by SS suggestions is strictly greater. Only in 5 cases QFG outperforms SS, while this never happens with CG. Also when considering effectiveness, i.e. the number of relevant suggestions among the top-10 returned, SS results the clear winner with an average number of relevant suggestions equal to 8.3, versus 4.3, and 4.2 for QFG, and CG, respectively. Moreover, differently from competitors SS results to be very robust w.r.t. data sparsity, and can produce relevant suggestions also to queries which are rare or not present in the query log used for training.

All the queries suggested by the three algorithms for the 50 TREC queries given as input to assessors are available along with the associated subtopic lists at `http://searchshortcuts.isti.cnr.it/TREC_results.html`. Moreover, a simple web-based wrapper that accepts user queries and computes the associated top-20 SS recommendations is available at `http://searchshortcuts.isti.cnr.it`.

6.6 Acknowledgements

We acknowledge the authors of [35, 36] and the Yahoo! Research Lab of Barcelona for providing us the possibility of using their Query Flow Graph implementation for evaluation purposes.

7

Efficient Diversification of Web Search Results

In this chapter we analyze the efficiency of various search results diversification methods. While efficacy of diversification approaches has been deeply investigated in the past, response time and scalability issues have been rarely addressed. A unified framework for studying performance and feasibility of result diversification solutions is thus proposed. First we define a new methodology for detecting when, and how, query results need to be diversified. To this purpose, we rely on the concept of "query refinement" to estimate the probability of a query to be *ambiguous*. Then, relying on this novel ambiguity detection method, we deploy and compare on a standard test set, three different diversification methods: IASelect, xQuAD, and OptSelect. While the first two are recent state-of-the-art proposals, the latter is an original algorithm introduced here. We evaluate both the efficiency and the effectiveness of our approach against its competitors by using the standard TREC Web diversification track testbed. Results show that OptSelect is able to run two orders of magnitude faster than the two other state-of-the-art approaches and to obtain comparable figures in diversification effectiveness. Finally, we extend a Web search architecture based on additive Machine Learned Ranking (MLR) systems with a new module computing the diversity score of each retrieved document. Our proposed solution is designed to be used with other techniques, (e.g. early termination of rank computation, etc.).

7.1 Introduction

Web Search Engines (WSEs) are nowadays the most popular mean of interaction with the Web. Users interact with them by usually typing a few keywords representing their information need. Queries, however, are often ambiguous and have more than one possible interpretation [7, 121].

Consider, for example, the popular single-term query "apple". It might refer to Apple Corp., to the fruit, or to a tour operator which is very popular in the US. Without any further information that may help to disambiguate the user intent,

search engines should produce a set of results possibly *covering* all (the majority of) the different interpretations of the query. To help users in finding the right information they are looking for, many different interfaces have been proposed to present search results. The first, and naïve, solution has been paging the ranked results list. Instead of presenting the whole list of results, the search engine presents results divided in pages (also known with the term SERP, i.e. Search Engine Result Page) containing ten results each.

Some alternative interfaces have been proposed. *Results clustering* approaches, for instance, organize search results into folders that group similar items together [171, 53]. On the other side of the same coin, *results diversification* [5] follows an intermediate approach that aims at "packing" the highest possible number of diverse, relevant results within the first SERP.

Given that web search engines' mission of satisfying their users is of paramount importance, diversification of web search results is a hot research topic nowadays. Nevertheless, the majority of research efforts have been put on studying effective diversification methods able to satisfy web users. In this chapter we take a different turn and consider the problem from the *efficiency* perspective. As Google's cofounder Larry Page declares[1]: "Speed is a major search priority, which is why in general we do not turn on new features if they will slow our services down." This chapter is, to the best of our knowledge, the first work discussing efficiency in SERP diversification. The approach we follow to achieve an efficient and viable solution is based on analyzing query log information to infer knowledge about when diversification actions have to be taken, and what users expectations are. Indeed, the original contributions presented are several:

- we define a methodology for detecting when, and how, query results need to be diversified. We rely on the well-known concept of query refinement to estimate the probability of a query to be *ambiguous*. In addition, we show how to derive the most likely refinements, and how to use them to diversify the list of results;
- we define a novel *utility* measure to evaluate how useful is a result for a diversified result list;
- we propose OptSelect, an original algorithm allowing the diversification task to be accomplished effectively and very efficiently;
- relying on our diversification framework, we deploy OptSelect and two other recent state-of-the-art diversification methods in order to evaluate on the standard TREC Web diversification track testbed both the efficiency and the effectiveness of our approach against its competitors;
- we extend a search architecture based on additive Machine Learned Ranking (MLR) systems with a new module computing the diversity score of each retrieved document. Our proposed solution is designed to be used with other techniques, (e.g. early termination of rank computation, etc.).

[1] `http://www.google.com/corporate/tech.html`

The chapter is organized as follows: Section 7.2 discusses related works. Section 7.3 presents a formalization of the problem, the specialization extraction method, and the algorithm we propose. Section 7.4 shows the efficiency of our solutions. Section 7.5 discusses some experimental results. In Section 7.7 we present our conclusions and we outline possible future work.

7.2 Related Work

Result diversification has recently attracted a lot of interest. A very important pioneering work on diversification is [52]. In this paper, the authors present the Maximal Marginal Relevance (MMR) problem, and they show how a trade-off between novelty and relevance of search results can be made explicit through the use of two different functions, the first measuring the similarity among documents, and the other the similarity between documents and the query. Zhai *et al.* [172] stated that in general it is not sufficient to return a set of relevant results as the correlation among the returned results is also very important. In a later work, Zhai *et al.* [174] formalize and propose a risk minimization approach that allow an arbitrary *loss* function over a set of returned documents to be defined. *Loss* functions aim at determining the *dissatisfaction* of the user with respect to a particular set of results. Such *loss* function depends on the language models rather than on categorical information about two documents [173].

Diversification has also been studied for purposes different from search engine result diversification. Ziegler *et al.* [180] study the diversification problem from a "recommendation" point of view. Radlinski *et al.* [123] propose a learning algorithm to compute an ordering of search results from a diverse set of orderings. Vee *et al.* [158] study the diversification problem in the context of structured databases with applications to online shopping. Clarke *et al.* [59] study diversification in question answering.

Agrawal *et al.* [5] present a systematic approach to diversify results that aims to minimize the risk of dissatisfaction of the web search engine users. Furthermore, authors generalize some classical IR metrics, including NDCG, MRR, and MAP, to explicitly account for the value of diversification. They show empirically that their algorithm scores higher in these generalized metrics compared to results produced by commercial search engines.

Gollapudi *et al.* [73] use the axiomatic approach to characterize and design diversification systems. They develop a set of axioms that a diversification system is expected to satisfy, and show that no diversification function can satisfy all these axioms simultaneously. Finally, they propose an evaluation methodology to characterize the objectives and the underlying axioms. They conduct a large scale evaluation based on data derived from Wikipedia and a product database.

Rafiei *et al.* [124] model the diversity problem as expectation maximization and study the challenges of estimating the model parameters and reaching an equilib-

rium. One model parameter, for example, is the correlation between pages which authors estimate using textual contents of pages and click data (when available). They conduct experiments on diversifying randomly selected queries from a query log and the queries chosen from the disambiguation topics of Wikipedia.

Clough *et al.* [60] examine user queries with respect to diversity. They analyze 14.9 million queries from the MSN query log by using two query log analysis techniques (click entropy and reformulated queries). Authors found that a broad range of query types may benefit from diversification. They also found that, although there is a correlation between word ambiguity and the need for diversity, the range of results users may wish to see for an ambiguous query stretches well beyond traditional notions of word sense.

Santos *et al.* [132, 131, 133] introduce a novel probabilistic framework (xQuAD) for Web search result diversification, which explicitly accounts for the various aspects associated to an underspecified query. In particular, they diversify a document ranking by estimating how well a given document satisfies each uncovered aspect and the extent to which different aspects are satisfied by the ranking as a whole. Authors evaluate the xQuAD framework in the context of the diversity task of the TREC 2009 Web track. They exploit query reformulations provided by three major WSEs to uncover different query aspects.

Similarly to [131], we exploit related queries as a mean of achieving diversification of query results. Nevertheless, our approach is very different from the above two. In [131], the authors exploit query reformulations provided by commercial Web search engines. All these reformulations are then taken into account during query processing in order to achieve a result set covering all the facets. In our case, the different meanings and facets of queries are instead disclosed by analyzing user behaviors recorded in query logs. During the analysis also the popularity of the different specializations is derived. Popularity distribution is then used to maximize the "usefulness" of the final set of documents returned. An approach orthogonal to our is instead investigated by Radlinski and Dumais in [121], where the problem of generating queries that can yield to a more diverse results set is studied starting from the observation that the top-k results retrieved for a query might not contain representative documents for all of its interpretations.

7.3 Diversification using Query Logs

Users generally query a search engine by submitting a sequence of requests. Splitting the chronologically ordered sequence of queries submitted by a given user into *sessions*, is a challenging research topic [118, 13, 92]. Since our approach exploits user session information, but session splitting methodologies are out of the scope of this work, we resort to adopt a state-of-the-art technique based on Query-Flow Graph [35, 37]. It consists of building a Markov Chain model of the query log and subsequently finding paths in the graph which are more likely to be followed

by random surfers. As a result, by processing a query log Q we obtain the set of logical user sessions exploited by our result diversification solution. Both the query topics possibly benefiting from diversification, and the probability of each distinct specialization among the spectrum of possibilities, are in fact mined from logs storing historical information about the interaction of users with the WSE.

As an example, let us assume that in a given query log the queries *leopard mac OS X*, *leopard tank*, and *leopard pictures*, are three specializations of query *leopard* that commonly occur in logical query sessions. The presence of the same query refinements in several sessions issued by different users gives us evidence that a query is ambiguous, while the relative popularity of its specializations allow us to compute the probabilities of the different meanings. On the basis of this information learned from historical data, once a query q is encountered by the WSE, we: (a) check if q is ambiguous or faceted, and if so, (b) exploit the knowledge about the different specializations of q submitted in the past to retrieve documents relevant for all of them. Finally, (c) use the relative frequencies of these specializations to build a final result set that maximize the probability of satisfying the user. In the following, we describe more precisely how ambiguous/faceted queries are detected and managed.

7.3.1 Mining Specializations from Query Logs

We assume that a query log Q is composed by a set of records $\langle q_i, u_i, t_i, V_i, G_i \rangle$ storing, for each submitted query q_i: (i) the anonymized user u_i; (ii) the timestamp t_i at which u_i issued q_i; (iii) the set V_i of URLs of documents returned as top-k results of the query, and, (iv), the set C_i of URLs corresponding to results clicked by u_i. Let q and q' be two queries submitted by the same user during the same logical session recorded in Q. We adopt the terminology proposed in [37], and we say that a query q' is a "specialization" of q if the user information need is stated more precisely in q' than in q (i.e., q' is more specific than q). Let us call S_q the set of specializations of an ambiguous/faceted query q mined from the query log.

Given the above generic definition, any algorithm that exploits the knowledge present in query log sessions to provide users with useful suggestions of related queries, can be easily adapted to the purpose of devising specializations of submitted queries. Given the popularity function $f()$ that computes the frequency of a query topic in Q, and a query recommendation algorithm $\mathcal{A}$ trained with Q, Algorithm 3 can be used to detect efficiently and effectively queries that can benefit from result diversification, and to compute for them the set of most common specializations along with their probabilities.

In this work, we experimented the use of a very efficient query recommendation algorithm [42] (see Chapter 6) for computing the possible specializations of queries. The algorithm used learns the suggestion model from the query log, and returns as related specializations, only queries that are present in Q, and for which related

Algorithm 3 AmbiguousQueryDetect$(q, \mathcal{A}, f(), s)$

{*Given the submitted query q, a query recommendation algorithm $\mathcal{A}$, and an integer s compute the set $\widehat{S}_q \subseteq Q$ of possible specializations of q.*}

1: $\widehat{S}_q \leftarrow \mathcal{A}(q)$; {*Select from $\widehat{S}_q$ the most popular specializations.*}
2: $S_q \leftarrow \{q' \in \widehat{S}_q | f(q') \geq \frac{f(q)}{s}\}$;
3: **if** $|S_q| \geq 2$ **then return** (S_q); **else return** $(\emptyset)$;

probabilities can be, thus, easily computed. Note that any other approach for deriving user intents from query logs, (as an example, [120, 129]), could be used and easily integrated in our diversification framework.

Definition 6 (Probability of Specialization). *Let $\widehat{Q} = \{q \in Q,\ s.t.\ |S_q| > 1\}$ be the set of ambiguous queries in Q, and $P(q'|q)$ the probability for $q \in \widehat{Q}$ to be specialized from $q' \in S_q$.*

We assume that the distribution underlying the possible specialization of an ambiguous query is known and complete, i.e., $\sum_{q' \in S_q} P(q'|q) = 1$, and $P(q'|q) = 0, \forall q' \notin S_q,\ \forall q \in \widehat{Q}$. To our purposes these probability distributions are simply estimated by dividing the frequency returned by Algorithm 3 using the following formula:

$$P(q'|q) = f(q') / \sum_{q' \in S_q} f(q')$$

Obviously, query logs can not give the complete knowledge about all the possible specializations for a given ambiguous query, but we can expect that the most popular interpretations are present in a large query log covering a long time period. Now, let us give some additional assumptions and notations.

$\mathcal{D}$ is the collection of documents indexed by the search engine which returns, for each submitted query q, an ordered list R_q of documents. The rank of document $d \in \mathcal{D}$ within R_q is indicated with $rank(d, R_q)$.

Moreover, let d_1 and d_2 be two documents of $\mathcal{D}$, and $\delta : \mathcal{D} \times \mathcal{D} \rightarrow [0,1]$ a distance function having the non-negative, and symmetric properties, i.e. (i) $\delta(d_1, d_2) = 0$ iff $d_1 = d_2$, and (ii) $\delta(d_1, d_2) = \delta(d_2, d_1)$.

Definition 7 (Results' Utility). *The* utility *of a result $d \in R_q$ for a specialization q' is defined as:*

$$U(d|R_{q'}) = \sum_{d' \in R_{q'}} \frac{1 - \delta(d, d')}{rank(d', R_{q'})}. \tag{7.1}$$

where $R_{q'}$ is the list of results that the search engine returned for specialized query q'.

Such utility represents how good $d \in R_q$ is for satisfying a user intent that is better represented by specialization q'. The intuition for U is that a result $d \in R_q$ is more useful for specialization q' if it is very similar to a highly ranked item contained in the results list $R_{q'}$.

The utility function specified in Equation (7.1) uses the following function to measure the distance between two documents:

$$\delta(d_1, d_2) = 1 - cosine(d_1, d_2) \tag{7.2}$$

where $cosine(d_1, d_2)$ is the cosine similarity between the two documents.

In the methods presented in the following, we use a normalized version of results' utility, $\widetilde{U}\left(d|R_{q'}\right)$, which is defined as the normalization of $U\left(d|R_{q'}\right)$ in the $[0, 1]$ interval. The normalization factor is computed by assuming that, in the optimal case, result d is at distance $\delta(d, \cdot) = 0$. In this case, the utility function is equal to

$$\sum_{d' \in R_{q'}} \frac{1}{rank(d', R_{q'})} \;=\; \sum_{i=1}^{|R_{q'}|} \frac{1}{i} \;=\; H_{|R_{q'}|}$$

where $H_{|R_{q'}|}$ is the $|R_{q'}|$-th harmonic number. Therefore,

$$\widetilde{U}\left(d|R_{q'}\right) \;=\; \frac{U\left(d|R_{q'}\right)}{H_{|R_{q'}|}}$$

Using the above definitions, we can now define three different query-logs-based approaches to diversification. The first two methods are adaptations of the Agrawal *et al.* [5] algorithm, and the Santos's *et al.* xQuAD framework [131]. The last one refers to our novel formulation detailed in Section 7.3.1.

The QL_Diversify(k) Problem

In a recent paper, Agrawal *et al.* [5] defined the DIVERSIFY(k) problem, a covering-like problem aimed to include the maximum number of possible "categories" into the list of k results that are returned in response to a user's query.

We briefly recall the definition of the problem, as stated in the original paper [5]:

> DIVERSIFY(k): Given query q, a set of documents R_q, a probability distribution of categories for the query $P\left(c|q\right)$, the quality values of the documents $V(d|q, c)$, $\forall d \in \mathcal{D}$ and an integer k. Find a set of documents $S \subseteq R_q$ with $|S| = k$ that maximizes
>
> $$P\left(S|q\right) = \sum_{c} P\left(c|q\right) \left(1 - \prod_{d \in S} \left(1 - V\left(d|q, c\right)\right)\right) \tag{7.3}$$

Equation (7.3) uses two concepts similar to those we have already introduced: the probability of a query to be part of a category is very similar to our concept of probability of specialization (see Definition 6), while quality value $V(d|q,c)$ resembles $\widetilde{U}(d|R_{q'})$.

It is possible to see the set of possible specializations S_q as the set of possible categories for q mined from a query log. The utility, in this case, can be seen as the utility of selecting resulting document d for category/specialization q'. Thus, the problem becomes choosing a subset S of R_q with $|S| = k$ that maximizes:

$$P\left(S|q\right) = \sum_{q' \in S_q} P\left(q'|q\right) \left(1 - \prod_{d \in S} \left(1 - \widetilde{U}\left(d|R_{q'}\right)\right)\right) \tag{7.4}$$

We call this problem QL_DIVERSIFY(k) to differentiate it from the original Agrawal *et al.* formulation [5].

The xQuAD_Diversify(k) Problem

In [131], Santos *et al.* propose a probabilistic framework called xQuAD. Compared to [5], the proposed framework extends the measure with which documents produced for ambiguous query q are iteratively selected. To this aim, xQuAD evaluates also the initial ranking of such documents for q. Formally, the problem is the following:

xQuAD_DIVERSIFY(k): Given a query q, a set of ranked documents R_q retrieved for q, a mixing parameter $\lambda \in [0, 1]$, two probability distributions $P(d|q)$ and $P(d, \bar{S}|q)$ measuring, respectively, the likelihood of document d being observed given q, and the likelihood of observing d but not the documents in the solution S. Find a set of documents $S \subseteq R_q$ with $|S| = k$ that maximizes for each $d \in S$

$$(1 - \lambda) \cdot P(d|q) \; + \; \lambda \cdot P(d, \bar{S}|q) \tag{7.5}$$

xQuAD is a greedy algorithm that iteratively selects a new document, and pushes it into the current solution. The selection process consists in choosing each time the document $d^* \in \bar{S} = R \setminus S$ that maximizes Equation (7.5). Such formula combines two probabilities: the first evaluates the *relevance* of a document d as the expectation for d to be observed given the query q, namely $P(d|q)$. The second probability measures the *diversity* of a candidate document d as the product of two components (see Equation (7.6)). The first component is, thus, the relevance of d with respect to a set of specializations S_q, and it is obtained by multiplying the likelihood of a specialization q' by the likelihood of d considering a particular specialization q'. Furthermore, the second component estimates the coverage degree of the current solution S with respect to each specialization q'.

$$P(d, \bar{S}|q) = \sum_{q' \in S_q} \Big[P(q'|q)\, P(d|q') \prod_{d_j \in S} 1 - P(d_j|q') \Big] \tag{7.6}$$

As for the Agrawal's formulation, $P(d_j|q')$ can be seen as the utility of selecting resulting document d_j for specialization q'. Thus, we measure $P(d_j|q')$ using $\widetilde{U}\,(d|R_{q'})$. Similarly to [5], at each step, the algorithm updates the coverage degree of solution S for each candidate document, then it scans $R \setminus S$ in order to choose the best document.

The MaxUtility_Diversify(k) Problem

The problem addressed in the Agrawal's paper, is actually the maximization of the *weighted* coverage of the categories with pertinent results. The objective function does not consider directly the number of categories covered by the result set; it might be the case that even if the categories are less than $|S_q|$, some of these will not be covered by the results set. This may happen because the objective function considers explicitly how much a document satisfies a given category. Hence, if a category that is a dominant interpretation of the query q is not covered adequately, more documents related to such category will be selected, possibly at the expense of other categories.

We believe, instead, that it is possible to maximize the sum of the various utilities for the chosen subset S of documents by guaranteeing that query specializations are covered proportionally to the associated probabilities $P(q'|q)$. Motivated by the above observation, we define the following problem.

MaxUtility_Diversify(k): Given a query q, the set R_q of results for q, two probability distributions $P(d|q)$ and $P\,(q'|q)\,\forall q' \in S_q$ measuring, respectively, the likelihood of document d being observed given q, and the likelihood of having q' as a specialization of q, the utilities $\widetilde{U}(d|R_{q'})$ of documents, a mixing parameter $\lambda \in [0, 1]$, and an integer k. Find a set of documents $S \subseteq R_q$ with $|S| = k$ that maximizes

$$\widetilde{U}\,(S|q) = \sum_{d \in S} \sum_{q' \in S_q} (1 - \lambda) P(d|q) + \lambda P\,(q'|q)\, \widetilde{U}\,(d|R_{q'}) \tag{7.7}$$

with the constraints that every specialization is covered proportionally to its probability. Formally, let $R_q \bowtie q' = \{d \in R_q | U\,(d|R_{q'}) > 0\}$. We require that for each $q' \in S_q$, $|R_q \bowtie q'| \geq \lfloor k \cdot P\,(q'|q) \rfloor$.

Our technique aims at selecting from R_q the k results that maximize the overall utility of the list of results. When $|S_q| \leq k$ the results are in someway split into $|S_q|$ subsets each one covering a distinct specializations. The more popular a specialization, the greater the number of results relevant for it. Obviously, if $|S_q| > k$ we select from S_q the k specializations with the largest probabilities.

7.4 Efficiency Evaluation

Efficiency of diversification algorithms is an important issue to study. Even the best diversification algorithm can be useless if its high computational cost forbids its actual use in a real-world IR system. In the following discussion, IASelect is the greedy algorithm used to approximate QL_DIVERSIFY(k), xQuAD refers to the greedy algorithm used to approximate XQUAD_DIVERSIFY(k), and eventually OptSelect is our algorithm solving the MAXUTILITY_DIVERSIFY(k) problem. We consider diversification to be done on a set of $|R_q| = n$ results returned by the baseline retrieval algorithm. Furthermore, we consider $|S_q|$, i.e. the number of specifications for a query q to be a constant (indeed, it is usually a small value depending on q).

IASelect. As shown by Agrawal *et al.* the DIVERSIFY(k) problem, and thus also the QL_DIVERSIFY(k) problem, is NP-Hard. Since the problem's objective function is *submodular*, an opportune greedy algorithm yields to a solution whose value is smaller than $(1 - 1/e)$ times the optimal one [110]. The greedy algorithm consists in adding to the results set the documents giving the largest marginal increase to the objective function. Since there is an insertion operation for each result needed in the final result set, the algorithm performs k insertions. For each insertion the algorithm searches for the document with the largest marginal utility that has not yet been selected. Since marginal utility is computed for each candidate document in terms of the current solution and each specialization, its value must be updated at each insertion. Hence, the computational cost of the procedure is linear in the number of categories/specializations multiplied by the number n of candidate documents. Thus, the solution proposed has a cost $C_{\mathrm{I}}(n,k) = \sum_{i=1}^{k}[\ |S_q| \cdot (n-i)\ ] = k|S_q|\left(n - \frac{k+1}{2}\right) = O(nk)$.

xQuAD. It is a greedy algorithm that iteratively selects a new document, and pushes it into the current solution. The selection process consists in choosing each time the document $d^* \in R \setminus S$ that maximizes Equation (7.5). As specified in Section 7.3.1, such formula combines the probability for a document d of being relevant for a query q, i.e., $P(d|q)$ and the *diversity* of a candidate document d, respectively.

Similarly to the solution proposed in [5], at each step, the algorithm updates the coverage degree of solution S for each candidate document, then it scans $R \setminus S$ in order to choose the best document. The procedure is linear in the number of items in S_q multiplied by the number of documents in $R \setminus S$. Since the selection is performed k times, the final computational cost is given by $C_{\mathrm{X}}(n,k) = \sum_{i=1}^{k}[\ |S_q| \cdot (n-i)\ ]$. As for the Agrawal's solution, thus, $C_{\mathrm{X}}(n,k) = O(nk)$.

OptSelect. While QL_DIVERSIFY(k) aims to maximize the probability of covering useful categories, the MAXUTILITY_DIVERSIFY(k) aims to maximize directly the overall utility. This simple relaxation allows the problem to be simplified and

solved *optimally* in a very simple and efficient way. Furthermore, the constraints bounding the minimum number of results tied to a given specialization, boost the quality of the final diversified result list, ensuring that the covered specializations reflect the most popular preferences expressed by users in the past.

Another important difference between Equation (7.7) and Equation (7.4) is that the latter needs to select, in advance, the subset S of documents before computing the final score. In our case, instead, a simple arithmetic argument shows that:

$$\widetilde{U}(S|q) = \sum_{d \in S} \widetilde{U}(d|q) \tag{7.8}$$

where $\widetilde{U}(d|q)$ is the overall utility of document d for query q. This value is computed according to the following equation:

$$\widetilde{U}(d|q) = \sum_{q' \in S_q} (1-\lambda) P(d|q) + \lambda P(q'|q)\, \widetilde{U}(d|R_{q'}) \tag{7.9}$$

By combining (7.8), and (7.9) we obtain:

$$\begin{aligned} \widetilde{U}(S|q) = (1-\lambda)|S_q| \sum_{d \in S} P(d|q) + \\ + \lambda \sum_{q' \in S_q} P(q'|q) \sum_{d \in S} \widetilde{U}(d|R_{q'}) \end{aligned}$$

Therefore, to maximize $\widetilde{U}(S|q)$ we simply resort to compute for each $d \in R_q$: i) the relevance of d for the query q, ii) the utility of d for specializations $q' \in S_q$ and, then, to *select* the top-k highest ranked documents. Obviously, we have to carefully select results to be included in the final list in order to avoid choosing results that are relevant only for a single specialization. For this reason we use a collection of $|S_q|$ min-heaps each of those keeps the top $\lfloor k \cdot P(q'|q) \rfloor + 1$ most useful documents for that specialization. OptSelect (see Algorithm (4)) returns the set S maximizing the objective function in Equation (7.7). Moreover, the running time of the algorithm is linear in the size of the documents considered. Indeed, all the heap operations are carried out on data structures having a constant size bounded by k.

Similarly to the other two solutions discussed, the proposed solution is computed by using a greedy algorithm. OptSelect is however computationally less expensive than its competitors. The main reason is that for each inserted element, it does not recompute the marginal utility of the remaining documents w.r.t. all the specializations. The main computational cost is given by the procedure which tries to add elements to each heap related to a specialization in S_q. Since each heap is of at most k positions, each insertion has cost $\log_2 k$, and globally the algorithm costs $C_{\mathrm{O}}(n,k) = n|S_q| \log_2 k = O(n \log_2 k)$.

Algorithm 4 OptSelect(q, S_q, R_q, k)

1: $S \leftarrow \emptyset$;
2: $M \leftarrow$ *new* Heap(k); {*Heap(n) instantiates a new n-size min-heap.*}
3: **For Each** $q' \in S_q$ **Do**
4: $M_{q'} \leftarrow$ *new* Heap$(\lfloor k \cdot P(q'|q) \rfloor + 1)$;
5: **For Each** $d \in R_q$ **Do**
6: **If** $\widetilde{U}(d|R_{q'}) > 0$ **Then** $M_{q'}.push(d)$; **Else** $M.push(d)$;
7: **For Each** $q' \in S_q$ s.t. $M_{q'} \neq \emptyset$ **Do**
8: $x \leftarrow$ pop d with the max $\widetilde{U}(d|q)$ from $M_{q'}$;
9: $S \leftarrow S \cup \{x\}$;
10: **While** $|S| < k$ **Do**
11: $x \leftarrow$ pop d with the max $\widetilde{U}(d|q)$ from M;
12: $S \leftarrow S \cup \{x\}$;
13: **Return** (S);

Table 7.1 reports and compares the theoretical complexity of the three considered methods. Our newly proposed OptSelect algorithm is faster than the previously proposed ones.

Algorithm	Complexity
IASelect	$O(nk)$
xQuAD	$O(nk)$
OptSelect	$O(nlog_2k)$

Table 7.1. Time complexity of the three algorithms considered.

Empirical efficiency evaluation. In addition to the theoretical considerations above, we also conducted tests in the TREC 2009 Web track's Diversity Task framework to empirically compare the efficiency of the three solutions proposed. In particular, we measured the time required by OptSelect, xQuAD and IASelect to diversify the list of retrieved documents. All the tests were conducted on a Intel Core 2 Quad PC with 8Gb of RAM and Ubuntu Linux 9.10 (kernel 2.6.31-22).

Table 7.2 reports the average time required by the three algorithms to diversify the initial set of documents for the 50 queries of the TREC 2009 Web Track's Diversity Task. We study the performance by varying both the number of documents which the diversified result set is chosen from ($|R_q|$), and the size of the returned list S denoted by k (i.e. $k = |S|$). The results show that, for each value of k, the execution time of all the tested methods is linear by varying the size of R_q. The only difference among these trends is in favor of OptSelect which slope is lower than its competitors. By varying, instead, the value of k, the execution times follow the complexities resumed in Table 7.1. The most remarkable result is that, increasing the number of documents returned, OptSelect outperforms xQuAD and IASelect in all the conducted tests. In particular, OptSelect is two orders of magnitude faster than its competitors.

$\|R_q\|$	k				
	10	50	100	500	1000
OptSelect					
1,000	0.34	0.58	0.66	0.89	0.98
10,000	1.36	2.13	2.46	3.32	3.57
100,000	4.81	8.32	9.57	12.94	13.92
xQuAD					
1,000	0.43	1.64	3.31	14.82	30.18
10,000	3.27	16.69	32.22	148.41	298.63
100,000	36.27	143.67	285.69	1,425.82	2,849.83
IASelect					
1,000	0.57	1.68	3.92	20.81	39.82
10,000	4.23	23.03	40.82	203.11	409.43
100,000	48.04	205.46	408.61	2,039.22	4,071.81

Table 7.2. Execution time (in msec.) of OptSelect, xQuAD, and IASelect by varying both the size of the initial set of documents to diversify ($|R_q|$), and the size of the diversified result set ($k = |S|$).

7.5 Testing Effectiveness

We conducted our experiments to measure the effectiveness of the three methods in the context of the diversity task of the TREC 2009 Web track [58]. The goal of this task is to produce a ranking of documents for a given query that maximizes the coverage of the possible aspects underlying this query, while reducing its overall redundancy with respect to the covered aspects. In our experiments, we used ClueWeb-B, the subset of the TREC ClueWeb09 dataset[2] and two query logs (AOL and MSN).

To assess effectiveness we have followed the guidelines of the Diversity Task of TREC. We have used the ClueWeb-B dataset, i.e. the subset of the TREC ClueWeb09 dataset[3] collection used in the TREC 2009 Web track's Diversity Task, comprising a total of 50 million English Web documents. A total of 50 topics were available for this task. Each topic includes from 3 to 8 sub-topics manually identified by TREC assessors, with relevance judgements provided at subtopic level. As an example the first TREC topic is identified by the query *obama family tree*, and three subtopics are provided: i) *Find the TIME magazine photo essay "Barack Obama's Family Tree"*, ii) *Where did Barack Obama's parents and grandparents come from?*, and iii) *Find biographical information on Barack Obama's mother.* The query associated with each topic of the TREC 2009 Web track was used as initial ambiguous/faceted query.

The two query logs used are AOL and MSN. The AOL data-set contains about 20 millions of queries issued by about 650,000 different users. The queries were

[2] `http://boston.lti.cs.cmu.edu/Data/clueweb09/`
[3] `http://boston.lti.cs.cmu.edu/Data/clueweb09/`

submitted to the AOL search portal over a period of three months from 1st March, 2006 to 31st May, 2006. The MSN Search query log contains 15 millions of queries submitted to the MSN US search portal over a period of one month in 2006. Queries are mostly in English. Both query logs come with all the information needed to address the diversification problem according to our approach.

The two query logs were first preprocessed in order to devise the logical user sessions as described in Section 7.3. Moreover the sessions obtained were used to build the model for the recommendation algorithm described in [42]. Given a query q, such algorithm was used to compute efficiently the set and the associated probabilities of its popular specializations S_q (see Algorithm 3).

The results obtained for the diversity task of the TREC 2009 Web track are evaluated according to the two official metrics: α-NDCG and IA-P. The α-normalized discounted cumulative gain (α-NDCG [59]) metric balances relevance and diversity through the tuning parameter α. The larger the value of α, the more diversity is rewarded. In contrast, when $\alpha = 0$, only relevance is rewarded, and this metric is equivalent to the traditional NDCG [90]. Moreover, we used the intent-aware precision (IA-P [5]) metric, which extends the traditional notion of precision in order to account for the possible aspects underlying a query and their relative importance. In our evaluation, both α-NDCG and IA-P are reported at five different rank cutoffs: 5, 10, 20, 100, and 1000. While the first four cutoffs focus on the evaluation at early ranks which are very important in a web context, the last cutoff gives the value of the two metrics for all the set of results. Both α-NDCG and IA-P are computed following the standard practice in the TREC 2009 Web-Track's Diversity Task [58]. In particular, α-NDCG is computed with $\alpha = 0.5$, in order to give an equal weight to relevance and diversity.

An ad-hoc modified version of the Terrier[4] IR platform was used for both indexing and retrieval. We extended Terrier in order to obtain short summaries of retrieved documents, which are used as document surrogates in our diversification algorithm. We used Porter's stemmer and standard English stopword removal for producing the ClueWeb-B index. We evaluate the effectiveness of our method in diversifying the results retrieved using a probabilistic document weighting model: DPH Divergence From Randomness (DFR) model [6].

Table 7.3 shows the results of the tests conducted with the DPH baseline (no diversification), i) our OptSelect, ii) Agrawal's IASelect, and iii) the xQuAD framework. We set $|R_{q'}| = 20$, $k = 1000$, and $|R_q| = 25,000$. Furthermore, xQuAD and OptSelect use a value for parameter λ equal to 0.15 (the value maximizing α-NDCG@20 in [131]). We applied the utility function in (7.1) to the snippets returned by the Terrier search engine instead of applying it to the whole documents, and we forced its returning value to be 0 when it is below a given threshold c. Nine different values of the utility threshold c were tested. The specializations and

[4] `http://www.terrier.org`

the associated probabilities were obtained in all the cases by using the previously described approach [42].

The results reported in the Table show that OptSelect and xQuAD behave similarly, while IASelect performs always worse. OptSelect shows good performance for small values of c, in particular for $c \in \{0, 0.05\}$. For both the two values of the threshold, OptSelect obtains very good α-NDCG performance and the best IA-P values. A deeper analysis of Table 7.3 shows that OptSelect noticeably outperforms the other two methods in terms of IA-P@5 for $c = 0.05$. The best α-NDCG performances for OptSelect are instead obtained for $c = 0.20$. For this value of the threshold, OptSelect shows a good trade-off between α-NDCG and IA-P, in particular for short results' lists (@5, @10, @20). However, none of these differences can be classified as statistically significant according to the Wilcoxon signed-rank test at 0.05 level of significance. By increasing the value of the threshold c, effectiveness starts to degrade. For $c \geq 0.75$ all the algorithms perform basically as the DPH baseline.

The xQuAD framework obtains good α-NDCG and IA-P performance for $c = 0.05$. xQuAD performs well also for $c = 0.20$. Note that our formulation of the xQuAD framework performs better than reported in the original paper by Santos *et al.* [131]. Essentially, this behavior could be explained by the following two reasons: i) our method for measuring the "diversity" of a document based on Equation (7.1) is superior to the one used in [131], ii) our method for deriving specializations, and their associated probabilities is able to carry out more accurate results. We leave this analysis to a future work. By comparing OptSelect ($c = 0.20$) and xQuAD ($c = 0.05$), we highlight better performances for OptSelect in terms of IA-P@5, and IA-P@20, while the xQuAD framework slightly outperforms OptSelect for α-NDCG, with an exception for α-NDCG@5 where the two methods behave similarly.

Agrawal's IASelect shows its best performances when the threshold c is not used. However, it never outperforms OptSelect and xQuAD. Both α-NDCG and IA-P values measured improve over the DPH baseline, but are always remarkably lower than the best values obtained using OptSelect and xQuAD.

7.5.1 Evaluation based on Query Log Data

A second evaluation we propose exploits the user sessions and the query specializations coming from the query logs of two commercial search engines. The aim of this evaluation is to show the importance of having a good diversification method based on real users' interests.

The two query logs were split into two different subsets. The first one (containing approximatively the 70% of the queries) was used for training (i.e., to build the data structures described in the previous section), and the second one for testing. For any ambiguous query q obtained by applying the Algorithm 3 to the test set of each query log, we first submitted the query to the Yahoo! BOSS search engine,

	c	α-NDCG					IA-P				
		@5	@10	@20	@100	@1000	@5	@10	@20	@100	@1000
DPH Baseline	-	0.190	0.212	0.240	0.303	0.303	0.092	0.093	0.088	0.058	0.006
OptSelect	0	**0.213**	0.227	0.255	0.318	0.352	0.111	0.100	**0.092**	0.061	0.012
	0.05	**0.213**	0.228	0.256	0.319	0.352	**0.112**	**0.101**	0.091	0.061	0.012
	0.10	0.195	0.220	0.246	0.312	0.343	0.102	0.097	0.090	**0.062**	0.012
	0.15	0.190	0.216	0.246	0.305	0.341	0.101	0.098	0.090	0.061	0.012
	0.20	**0.214**	**0.241**	**0.262**	**0.324**	**0.359**	0.110	0.101	0.090	0.060	0.012
	0.25	0.190	0.213	0.238	0.305	0.339	0.095	0.098	0.087	0.058	0.012
	0.35	0.186	0.206	0.235	0.302	0.335	0.089	0.090	0.086	0.058	0.012
	0.50	0.186	0.208	0.236	0.300	0.334	0.091	0.091	0.087	0.058	0.012
	0.75	0.190	0.212	0.240	0.303	0.337	0.092	0.093	0.088	0.058	0.012
xQuAD	0	0.211	0.241	0.260	0.320	0.354	0.103	0.102	0.090	0.058	0.012
	0.05	**0.214**	**0.242**	**0.260**	**0.323**	**0.355**	**0.108**	**0.103**	**0.089**	**0.058**	0.012
	0.10	0.193	0.226	0.249	0.308	0.341	0.101	0.101	0.090	0.058	0.012
	0.15	0.200	0.227	0.253	0.315	0.348	0.099	0.095	0.087	0.058	0.012
	0.20	0.204	0.234	0.262	0.321	0.354	0.096	0.099	0.087	0.058	0.012
	0.25	0.181	0.211	0.236	0.303	0.336	0.090	0.095	0.085	0.058	0.012
	0.35	0.185	0.209	0.239	0.302	0.335	0.091	0.092	0.088	0.058	0.012
	0.50	0.190	0.212	0.240	0.303	0.336	0.092	0.093	0.087	0.058	0.012
	0.75	0.190	0.212	0.240	0.303	0.337	0.092	0.093	0.088	0.058	0.012
IASelect	0	**0.206**	**0.215**	**0.245**	**0.302**	0.334	0.097	0.089	0.079	0.044	0.009
	0.05	0.205	0.214	0.243	0.299	0.330	**0.098**	0.090	0.078	0.044	0.009
	0.10	0.193	0.200	0.227	0.279	0.309	**0.098**	0.088	0.075	0.039	0.008
	0.15	0.169	0.185	0.207	0.259	0.288	0.089	0.078	0.064	0.039	0.008
	0.20	0.182	0.197	0.229	0.284	0.314	0.085	0.074	0.067	0.046	0.009
	0.25	0.198	0.214	0.243	0.301	0.332	0.092	0.083	0.076	0.052	0.011
	0.35	0.192	0.208	0.241	0.299	0.332	0.095	**0.093**	0.087	0.057	0.012
	0.50	0.192	0.214	0.243	0.306	**0.338**	0.093	0.091	0.087	**0.058**	0.012
	0.75	0.190	0.212	0.240	0.303	0.337	0.092	**0.093**	**0.088**	**0.058**	0.012

Table 7.3. Values of α-NDCG, and IA-P for OptSelect, xQuAD, and IASelect by varying the threshold c.

then we re-ranked the results list by means of the Algorithm 4 to obtain the corresponding diversified list of results. Finally, we compared the two lists obtained by means of the utility function as in Definition 7. The goal is to show that our diversification technique can provide users with a list of k documents having a utility greater than the top-k results returned by the Yahoo! BOSS Search Engine.

To assess the impact of our diversification strategy on the utility of the diversified results list, we computed the ratio between the normalized utilities of the results in S and the top-k results in R_q, i.e., the diversified and the original one. More formally, we computed

$$\frac{\sum_{i=1}^{k} \widetilde{U}(d_i \in S)}{\sum_{i=1}^{k} \widetilde{U}(d_i \in R_q)}$$

where S is the diversified list produced by OptSelect, whereas R_q is the original list of results obtained from Yahoo! BOSS. It is clear that if the two lists share all the results, the ratio is equal to 1.

In our tests, we set the number of results retrieved from Yahoo! BOSS ($|R_q|$) equal to 200, while both $|R_{q'}|$ and k equal to 20.

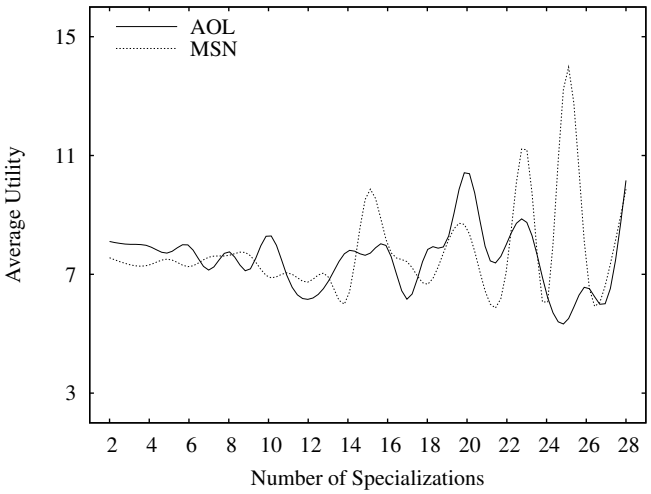


Fig. 7.1. Average utility per number of specializations referring to the AOL and MSN query logs.

Figure 7.1 shows the average utility per number of specializations for the two query logs considered in our experiments. In all cases taken into account, our method diversifies the final list by improving the usefulness measure for a factor ranging from 5 to 10 with respect to the usefulness of the original result set.

Furthermore, we measured the number of times our method is able to provide diversified results when they are actually needed, i.e., a sort of *recall* measure. This was done by considering the number of times a user, after submitting an ambiguous/faceted query, issued a new query that is a specialization of the previous one. In both cases we are able to provide diversified results for a large fraction of the queries. Concerning AOL, we are able to diversify results for the 61% of the cases, whereas for MSN this *recall* measure raises up to 65%.

7.6 A Search Architecture Enabling Efficient Diversification of Search Results

In the previous sections we have sketched the OptSelect algorithm as an efficient and effective solution for the diversification task. Here, we show how such a solution needs to be adapted in order to be plugged in a modern Machine Learned

Ranking (MLR) system having a pipelined architecture. In modern WSE query response time constraints are satisfied employing a two-phase scoring. The first phase inaccurately selects a small subset of potentially relevant documents from the entire collection (e.g. a BM25 variant). In the second phase, resulting candidate documents are scored again by a complex and accurate MLR architecture. The final rank is usually determined by additive ensembles (e.g. boosted decision trees [179]), where many scorers are executed sequentially in a chain and the results of the scorers are added to compute the final document score.

Let us assume that, given a query q, MLR algorithms are used to rank a set $D = \{d_1, \ldots, d_m\}$ of documents according to their relevance to q. Then the k documents with the highest score are returned. To this end, additive ensembles are used to compute the final score $s(d_i)$ of a document d_i as a sum over many, simple scorers, i.e. $s(d_i) = \sum_{j=1}^{n} f_j(d_i)$, where f_j is a scorer that belongs to a set of n scorers executed in a sequence. Moreover, the set of scorers is expected to be sorted by decreasing order of importance. This because, as argued in [47], if we can estimate the likelihood that d_i will end up within the top-k documents, we can early exit the $s(d_i)$ computation at any position $t < n$, computing a partial final score using only the first t scorers. For these reasons, it is important to define a solution that is fully integrable with the existing systems. Another important aspect to consider is the cost of each f_j that must be sustainable w.r.t. the others scorers. In particular, we assume that the cost c of computing $f_j(d_i)$ is constant and the total cost of scoring all documents in D is, thus $\mathcal{C}(D) = c \cdot m \cdot n$. For tasks with tight constraints on execution time, this cost is not sustainable if both m and n are high (e.g. $m > 10^5$ and $n > 10^3$ as shown in [47]).

To achieve the previously specified goal, WSE needs some additional modules in order to enable the diversification stage, see Figure 7.2. Briefly, our idea is the following. Given a query q, perform simultaneously both the selection of the documents potentially relevant for q from the entire collection (module `BM25`) and the retrieve of the specializations for q (module `SS`). Assuming that `SS` performs faster than both `DR` and `BM25`, the module $f_{\texttt{DVR}}$ can be placed in any position of the MLR pipeline, i.e. $f_1 \rightarrow_{\ldots} f_n$. The target of $f_{\texttt{DVR}}$ is, then, to exploit Equation (7.1) for properly increasing the rank of the incoming documents as the other pipelined scorers do. Note that in this case, that is different from OptSelect running context, the final extraction of top-k documents is left to the MLR pipeline that already performs this operation automatically. In the following, we give more detail on our approach.

For any given query q submitted to the engine, we dispatch q to the document retriever `DR` that processes the query on the inverted index, and to the module `SS` that generates the specializations S_q for q. `SS` processes q on a specific inverted index structure derived from query logs: the same proposed in [42]. `SS` returns a set of specializations S_q, a distribution of probability $P(q'|q)\, \forall q' \in S_q$, and a set $R_{q'} \forall q' \in S_q$ of sketches representing the most relevant documents for each special-

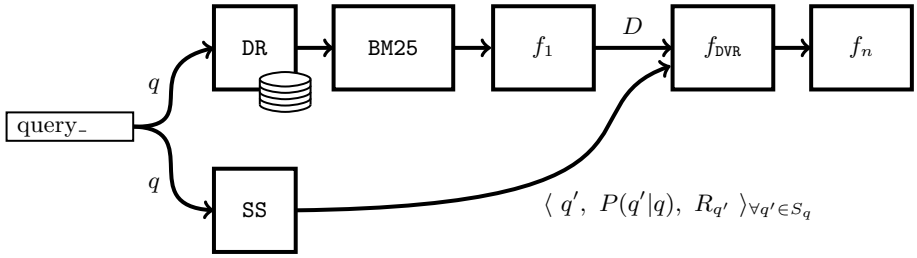


Fig. 7.2. A sketch of the WSE architecture enabling diversification.

ization. Concerning the feasibility in space of the inverted index in SS, note that each set $R_{q'}$ related to a specialization $q' \in S_q$ is very small compared to the set of whole documents R_q to re-rank, i.e. $|R_{q'}| \ll |R_q|$. Furthermore, using *shingles* [46], only a sketch of a few hundred bytes, and not the whole documents, can be used to represent a document without significant loss in the precision of our method[5]. Resuming, let ℓ be the average size in bytes of a shingle representing a document and let h be the average space needed to store the set S_q of specializations for a query q by using the related inverted index, we need at most $(N \cdot |S_{\widehat{q}}| \cdot |R_{\widehat{q}'}| \cdot \ell + N \cdot h)$ bytes for storing N ambiguous query along with the data needed to assess the similarity among results lists. For example, considering a number of ambiguous queries of order of hundreds of thousands, tens of specializations per query, and hundreds of documents per specialization, we need an inverted index for SS of about 10 GB.

Now, let us focus on $f_{\texttt{DVR}}$. As the other modules belonging to the MLR pipeline, also $f_{\texttt{DVR}}$ receives a set of documents D as a stream from its preceding module, scores the elements, then release the updated set. However, contrarily to other diversifying methods analyzed in [50], $f_{\texttt{DVR}}$ is able to compute on the fly the diversity-score for each document d. In fact, exploiting the knowledge retrieved from the query log, our approach does not require to know in advance the composition of D to diversify the query result because SS provides the proper mix of different means related to q. In particular, we firstly compute for each $d \in D$ the related shingle. As stated in [46], the related sketch can be efficiently computed (in time linear in the size of the document d) and, given two sketches, the similarity $1 - \delta(d, d')$ of the corresponding documents (i.e. $d \in D$ and each document d' returned by SS, i.e. $d' \in R_{q'}\ \forall q' \in S_q$) can be computed in time linear in the size of the sketches. The resulting similarity thus concurs to compute $U(d|R_{q'})$, i.e. the variation of final score of the document d.

[5] note that shingles are already maintained by the WSE for near duplicate document detection.

7.7 Summary

We studied the problem of diversifying search results by exploiting the knowledge derived from query logs. We presented a general framework for query result diversification comprising: i) an efficient and effective methodology, based on state-of-the-art query recommendation algorithms, to detect ambiguous queries that would benefit from diversification, and to devise all the possible common specializations to be included in the diversified list of results along with their probability distribution, ii) OptSelect: a new diversification algorithm which re-ranks the original results list on the basis of the mined specializations, iii) a Web search architecture based on additive Machine Learned Ranking (MLR) systems extended with a new module computing the diversity score of each retrieved document.

A novel formulation of the problem has been proposed and motivated. It allows the diversification problem to be modeled as a maximization problem. The approach is evaluated by using the metrics and the datasets provided for the TREC 2009 Web Track's Diversity Task. Our experimental results show that our approach is both efficient and effective. In terms of efficiency, our approach performs two orders of magnitude faster than its competitors and it remarkably outperforms its competitors in all the tests.

In terms of effectiveness, our approach outperforms the Agrawal's IASelect, and it shows the best results in terms of IA-P [5]. It produces also results that are comparable with the xQuAD framework in terms of α-NDCG [59].

Finally, we sketched a Web search architecture based on additive Machine Learned Ranking (MLR) systems enabling query result diversification, and we outline a first preliminary study on the feasibility of the technique.

8

Conclusions and Future Work

Queries and their clicked results implicitly reveal the opinion of users about their searches. This information is provided implicitly by users and recorded in search engine *query logs*. By mining query logs it is possible to derive a knowledge representing one of the most used ways of enhancing the users' search experience.

In this thesis we introduced four new contributions in two important fields: Web information retrieval and query log mining.

We studied the effect of time on recommendations generated using *Query Flow Graphs* [35] (QFGs) (see Chapter 4). We showed how to extend QFG-based recommendation models to evolving data. We showed that the interests of search engine users change over time and new topics may become popular, while other interests that focused for some time the attention of the crowds can suddenly loose importance. The knowledge extracted from query logs can thus suffer from an aging effect, and the models used for recommendations rapidly become unable to generate useful and interesting suggestions. We showed that the building of a new fresh QFG from scratch is expensive. To overcome this problem we introduced an *incremental* algorithm for updating an existing QFG. The solution proposed allows the recommendation model to be kept always updated by incrementally adding fresh knowledge and deleting the aged one.

In order to validate our claims and assess our methodology, we built different QFGs from the query log of a real-world search engine, and we analyzed the quality of the recommendation models obtained from these graphs to show that they inexorably age. Then, we proposed a general methodology for dealing with aging QFG models that allows the recommendation model to be kept up-to-dated in a remarkably lower time than that required for building a new model from scratch. As a side result we proposed a parallel/distributed solution allowing to make QFG creation/update operations scalable.

In Chapter 4 we proved that the knowledge extracted from historical usage data can suffer an aging effect. In Chapter 5 we thus studied the effects of incremental model updates on the effectiveness of two query suggestion algorithms. We intro-

duced a new class of query recommender algorithms that update *incrementally* the model on which recommendations are drawn. Starting from two state-of-the-art algorithms, we designed two new query recommender systems that continuously update their models as queries are issued. The two incremental algorithms differ from their static counterparts by the way in which they manage and use data to build the model.

In addition, we proposed an automatic evaluation mechanism based on four new metrics to assess the effectiveness of query recommendation algorithms. The experimental evaluation conducted by using a large real-world query log shows that the incremental update strategy for the recommendation model yields better results for both coverage (more than 20% queries covered by both IAssociationRules, and ICoverGraph) and effectiveness due to the "fresh" data that are added to the recommendation models. Furthermore, this improved effectiveness is accomplished without compromising the efficiency of the query suggestion process.

In Chapter 6 we proposed a very efficient solution for generating effective suggestions to Web search engine users based on the model of *Search Shortcut* [22]. Our original formulation of the problem allows the query suggestion generation phase to be re-conducted to the processing of a full-text query over an inverted index. The current query issued by the user is matched over the inverted index, and final queries of the most similar satisfactory sessions are efficiently selected to be proposed to the user as query shortcuts. The way a satisfactory session is represented as a virtual document, and the IR-based technique exploited, allows our technique to generate in many cases effective suggestions even to rare or not previously seen queries.

By using the automatic evaluation approach based on the metric defined in Equation (6.1), SS outperforms QFG in quality of a 0.17, while the improvement over CG was even greater (0.22). In 36 evaluated sessions out of 100, SS generates useful suggestions when its competitors CG and QFG fails to produce even a single useful recommendation.

An additional contribution of the chapter regards the evaluation methodology used, based on a publicly-available test collection provided by a highly reputed organization such as the NIST. On the basis of the evaluation conducted by means of the user-study, SS remarkably outperforms both QFG and CG in almost all the tests conducted. In particular, suggestions generated by SS covers the same number or more TREC subtopics than its two counterparts in 44 cases out of 50. In 36 cases the number of subtopics covered by SS suggestions was strictly greater. Only in 5 cases QFG outperforms SS, while this never happens with CG. Also when considering effectiveness, i.e. the number of relevant suggestions among the top-10 returned, SS results the clear winner with an average number of relevant suggestions equal to 8.3, versus 4.3, and 4.2 for QFG, and CG, respectively. Moreover, differently from competitors SS results to be very robust w.r.t. data

sparsity, and can produce relevant suggestions also to queries which are rare or not present in the query log used for training.

As future work we intend to evaluate the use the whole head of the user session for producing query recommendations. Furthermore, we want to study if the sharing of the same final queries induces a sort of "clustering" of the queries composing the satisfactory user sessions. By studying such relation which is at the basis of our query shortcut implementation, we could probably find ways to improve our methodology. Finally, it would be interesting to investigate how IR-like diversification algorithms (e.g., [5]) could be integrated in our query suggestion technique in order to obtain diversified query suggestions [102, 38].

Finally, in Chapter 7 we studied the problem of diversifying search results by exploiting the knowledge derived from query logs. We presented a general framework for query result diversification comprising: i) an efficient and effective methodology, based on state-of-the-art query recommendation algorithms, to detect ambiguous queries that would benefit from diversification, and to devise all the possible common specializations to be included in the diversified list of results along with their probability distribution, ii) OptSelect: a new diversification algorithm which reranks the original results list on the basis of the mined specializations.

A novel formulation of the problem has been proposed and motivated. It allows the diversification problem to be modeled as a maximization problem. The approach is evaluated by using the metrics and the datasets provided for the TREC 2009 Web Track's Diversity Task. Our experimental results show that our approach is both efficient and effective. In terms of efficiency, our approach performs two orders of magnitude faster than its competitors and it remarkably outperforms its competitors in all the tests.

In terms of effectiveness, our approach outperforms the Agrawal's IASelect, and it shows the best results in terms of IA-P [5]. It produces also results that are comparable with the xQuAD framework in terms of α-NDCG [59].

Finally, we sketched a Web search architecture based on additive Machine Learned Ranking (MLR) systems enabling query result diversification, and we outline a first preliminary study on the feasibility of the technique.

Some possible research directions can be drawn on this topic. Firstly, we will conduct a deeper analysis of the feasibility of our proposed Web search architecture based on additive Machine Learned Ranking (MLR) systems. Secondly, we will study how to exploit users' search history for personalizing their results diversification. Finally, we are planning to use click-through data to improve the effectiveness of our method.

References

1. E. Adar. User 4xxxxx9: Anonymizing query logs. In *Query Logs Workshop, WWW*, volume 7. Citeseer, 2007.
2. Eytan Adar, Daniel Weld, Brian Bershad, and Steven Gribble. Why we search: Visualizing and predicting user behavior. In *In Proc. WWW'07*, pages 161–170, Banff, Canada, 2007.
3. Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6):734–749, 2005.
4. R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, 1993.
5. Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *Proc. WSDM'09*. ACM, 2009.
6. Gianni Amati, Edgardo Ambrosi, Marco Bianchi, Carlo Gaibisso, and Giorgio Gambosi. FUB, IASI-CNR and university of Tor Vergata at trec 2007 blog track. In *Proc. TREC*, 2007.
7. Aris Anagnostopoulos, Andrei Z. Broder, and David Carmel. Sampling search-engine results. In *Proc. WWW'05*. ACM, 2005.
8. V. Authors. About web analytics association. `http://www.webanalyticsassociation.org/?page=aboutus`, retrieved on December 2010.
9. R. Baeza-Yates. Applications of web query mining. *Advances in Information Retrieval*, pages 7–22, 2005.
10. R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges in distributed information retrieval. In *International Conference on Data Engineering (ICDE)*, 2007.
11. R. Baeza-Yates, C. Hurtado, and M. Mendoza. Ranking boosting based in query clustering. In *Atlantic Web Intelligence Conference, Cancun, Mexico*, 2004.
12. R. Baeza-Yates, C. Hurtado, and M. Mendoza. Improving search engines by query clustering. *Journal of the American Society for Information Science and Technology*, 58(12):1793–1804, 2007.
13. Ricardo Baeza-Yates. Graphs from search engine queries. In *Proc. SOFSEM'07*, pages 1–8, Harrachov, CZ, 2007.
14. Ricardo Baeza-Yates. Web mining or the wisdom of the crowds. In *Proceeding of the 2008 conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 3–3, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

15. Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. The impact of caching on search engines. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 183–190, New York, NY, USA, 2007. ACM.
16. Ricardo Baeza-Yates and Alessandro Tiberi. Extracting semantic relations from query logs. In *Proc. KDD'07*. ACM, 2007.
17. Ricardo Baeza-Yates and Alessandro Tiberi. Extracting semantic relations from query logs. In *KDD '07*, pages 76–85, New York, NY, USA, 2007. ACM.
18. Ricardo A. Baeza-yates, Carlos A. Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Proc. EDBT'04*, pages 588–596, 2004.
19. Evelyn Balfe and Barry Smyth. Improving web search through collaborative query recommendation. In *Proc. ECAI'04*. IOS Press, 2004.
20. Evelyn Balfe and Barry Smyth. A comparative analysis of query similarity metrics for community-based web search. In Héctor Muñoz-Avila and Francesco Ricci, editors, *ICCBR*, volume 3620 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2005.
21. J. Bar-Ilan. Access to query logs-an academic researchers point of view. In *Query Log Analysis: Social And Technological Challenges Workshop at WWW*, 2007.
22. Ranieri Baraglia, Fidel Cacheda, Victor Carneiro, Diego Fernandez, Vreixo Formoso, Raffaele Perego, and Fabrizio Silvestri. Search shortcuts: a new approach to the recommendation of queries. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, pages 77–84, New York, NY, USA, 2009. ACM.
23. Ranieri Baraglia, Carlos Castillo, Debora Donato, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Aging effects on query flow graphs for query suggestion. In *Proc. CIKM'09*. ACM, 2009.
24. Ranieri Baraglia, Carlos Castillo, Debora Donato, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. The effects of time on query flow grraph-based models for query suggestion. In *Proc. RIAO'10.*, 2010.
25. L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The Google cluster architecture. *IEEE micro*, 23(2):22–28, 2003.
26. Doug Beeferman and Adam Berger. Agglomerative clustering of a search engine query log. In *Proc. KDD'00*. ACM, 2000.
27. Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, Ophir Frieder, and David Grossman. Temporal analysis of a very large topically categorized web query log. *J. Am. Soc. Inf. Sci. Technol.*, 58:166–178, January 2007.
28. Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David Grossman, and Ophir Frieder. Hourly analysis of a very large topically categorized web query log. In *In Proc. SIGIR'04*, pages 321–328. ACM Press, 2004.
29. Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David Grossman, Ophir Frieder, and Nazli Goharian. Fusion of effective retrieval strategies in the same information retrieval system. *J. Am. Soc. Inf. Sci. Technol.*, 55:859–868, August 2004.
30. Steven M. Beitzel, Eric C. Jensen, Ophir Frieder, David D. Lewis, Abdur Chowdhury, and Aleksander Kolcz. Improving automatic query classification via semi-supervised learning. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 42–49, Washington, DC, USA, 2005. IEEE Computer Society.
31. Steven M. Beitzel, Eric C. Jensen, David D. Lewis, Abdur Chowdhury, and Ophir Frieder. Automatic classification of web queries using very large unlabeled query logs. *ACM Trans. Inf. Syst.*, 25, April 2007.

32. Bodo Billerbeck, Falk Scholer, Hugh E. Williams, and Justin Zobel. Query expansion using associated queries. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 2–9, New York, NY, USA, 2003. ACM.
33. P. Boldi, F. Bonchi, C. Castillo, and S. Vigna. Query reformulation mining: models, patterns, and applications. *Information Retrieval*, pages 1–33, 2010.
34. P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *Proc. WWW'04*. ACM Press, 2004.
35. Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. The query-flow graph: model and applications. In *Proc. CIKM'08*. ACM, 2008.
36. Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, and Sebastiano Vigna. Query suggestions using query-flow graphs. In *Proc. WSCD'09*. ACM, 2009.
37. Paolo Boldi, Francesco Bonchi, Carlos Castillo, and Sebastiano Vigna. From 'dango' to 'japanese cakes': Query reformulation models and patterns. In *Proc. WI'09*. IEEE, September 2009.
38. Ilaria Bordino, Carlos Castillo, Debora Donato, and Aristides Gionis. Query similarity by projecting the query-flow graph. In *Proc. SIGIR'10*. ACM, 2010.
39. Ilaria Bordino and Daniele Laguardia. Algebra for the joint mining of query log graphs. http://www.dis.uniroma1.it/∼bordino/qlogs_algebra/, 2008.
40. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine* 1. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.
41. Daniele Broccolo, Ophir Frieder, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Incremental algorithms for effective and efficient query recommendation. In *Proc. SPIRE'10*, volume 6393 of *LNCS*, pages 13–24. Springer, 2010.
42. Daniele Broccolo, Lorenzo Marcon, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. An efficient algorithm to generate search shortcuts. Technical Report N. /cnr.isti/2010-TR-017, CNR ISTI Pisa Italy, 2010.
43. Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36:3–10, September 2002.
44. Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich, Vanja Josifovski, Donald Metzler, Lance Riedel, and Jeffrey Yuan. Online expansion of rare queries for sponsored search. In *Proc. WWW'09*. ACM, 2009.
45. Andrei Z. Broder, Marcus Fontoura, Evgeniy Gabrilovich, Amruta Joshi, Vanja Josifovski, and Tong Zhang. Robust classification of rare queries using web knowledge. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 231–238, New York, NY, USA, 2007. ACM.
46. Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29:1157–1166, September 1997.
47. B.B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 411–420. ACM, 2010.
48. John Canny. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th annual international ACM SIGIR 2002 Conference*, pages 238–245, New York, NY, USA, 2002. ACM.
49. Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. Context-aware query suggestion by mining click-through and session data. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 875–883, New York, NY, USA, 2008. ACM.

50. Gabriele Capannini, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Efficient diversification of search results using query logs. In *Proc. WWW'11*, New York, NY, USA, 2011. ACM.
51. Gabriele Capannini, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Efficient diversification of web search results. *Proceedings of the VLDB, Volume 4, Issue 7*, April 2011.
52. Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. SIGIR'98*. ACM, 1998.
53. Claudio Carpineto, Stanislaw Osiński, Giovanni Romano, and Dawid Weiss. A survey of web clustering engines. *ACM Comput. Surv.*, 41(3):1–38, 2009.
54. Aysegul Cayci, Selcuk Sumengen, Cagatay Turkay, Selim Balcisoy, and Yucel Saygin. Temporal dynamics of user interests in web search queries. *ICAINAW*, 0:762–767, 2009.
55. D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 554–560. ACM, 2006.
56. Steve Chien and Nicole Immorlica. Semantic similarity between search engine queries using temporal correlation. In *In Proc. WWW'05*, pages 2–11, New York, NY, USA, 2005. ACM Press.
57. Paul Alexandru Chirita, Claudiu S. Firan, and Wolfgang Nejdl. Personalized query expansion for the web. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 7–14, New York, NY, USA, 2007. ACM.
58. C. Clarke, N. Craswell, and I. Soboroff. Preliminary report on the TREC 2009 Web track. In *Proc. TREC'09*. ACM, 2009.
59. Charles L.A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proc. SIGIR'08*. ACM, 2008.
60. Paul Clough, Mark Sanderson, Murad Abouammoh, Sergio Navarro, and Monica Paramita. Multiple approaches to analysing query diversity. In *Proc. SIGIR'09*. ACM, 2009.
61. Kevyn Collins-Thompson and Jamie Callan. Query expansion using random walk models. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 704–711, New York, NY, USA, 2005. ACM.
62. A. Cooper. A survey of query log privacy-enhancing techniques from a policy perspective. *ACM Transactions on the Web (TWEB)*, 2(4):1–27, 2008.
63. S. Cucerzan and R.W. White. Query suggestion based on user landing pages. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 875–876. ACM, 2007.
64. H. Cui, J.R. Wen, J.Y. Nie, and W.Y. Ma. Probabilistic query expansion using query logs. In *Proceedings of the 11th international conference on World Wide Web*, pages 325–332. ACM, 2002.
65. Doug Downey, Susan Dumais, and Eric Horvitz. Heads and tails: studies of web search with common and rare queries. In *Proc. SIGIR'07*. ACM, 2007.
66. Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24:51–78, January 2006.
67. C.H. Fenichel. Online searching: Measures that discriminate among users with different types of experiences. *Journal of the American Society for Information Science*, 32(1):23–32, 1981.

68. L. Fitzpatrick and M. Dent. Automatic feedback using past queries: social searching? In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 306–313. ACM, 1997.
69. Bruno M. Fonseca, Paulo Golgher, Bruno Pôssas, Berthier Ribeiro-Neto, and Nivio Ziviani. Concept-based interactive query expansion. In *Proc. CIKM'05*. ACM, 2005.
70. Bruno M. Fonseca, Paulo B. Golgher, Edleno S. de Moura, and Nivio Ziviani. Using association rules to discover search engines related queries. In *LA-WEB'03*. IEEE CS, 2003.
71. Ophir Frieder, David A. Grossman, Abdur Chowdhury, and Gideon Frieder. Efficiency considerations for scalable information retrieval servers. *J. Digit. Inf.*, 1(5), 2000.
72. G.P.C. Fung, J.X. Yu, P.S. Yu, and H. Lu. Parameter free bursty events detection in text streams. In *Proceedings of the 31st international conference on Very large data bases*, pages 181–192. VLDB Endowment, 2005.
73. Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *Proc. WWW'09*. ACM, 2009.
74. Luis Gravano, Vasileios Hatzivassiloglou, and Richard Lichtenstein. Categorizing web queries according to geographical locality. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 325–333, New York, NY, USA, 2003. ACM.
75. Taher H. Haveliwala. Topic-sensitive pagerank. In *Proc. WWW'02*. ACM, 2002.
76. Daqing He and Ayse Göker. Detecting session boundaries from web user logs. In *BCS-IRSG*, pages 57–66, 2000.
77. Daqing He, Ayse Göker, and David J. Harper. Combining evidence for automatic web session identification. *Inf. Process. Manage.*, 38:727–742, September 2002.
78. Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22:89–115, January 2004.
79. Lngrid Hsieh-yee. Effects of search experience and subject knowledge on the search tactics of novice and experienced searchers. *Journal of the American Society for Information Science*, 44:161–174, 1993.
80. Bernard J. Jansen. *Understanding User-Web Interactions via Web Analytics*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009.
81. Bernard J. Jansen and Marc Resnick. An examination of searcher's perceptions of nonsponsored and sponsored links during ecommerce web searching. *J. Am. Soc. Inf. Sci. Technol.*, 57:1949–1961, December 2006.
82. Bernard J. Jansen and Amanda Spink. An analysis of web searching by european alltheweb.com users. *Inf. Process. Manage.*, 41:361–381, March 2005.
83. Bernard J. Jansen and Amanda Spink. How are we searching the world wide web?: a comparison of nine search engine transaction logs. *Inf. Process. Manage.*, 42:248–263, January 2006.
84. Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32:5–17, April 1998.
85. Bernard J. Jansen, Amanda Spink, Chris Blakely, and Sherry Koshman. Defining a session on web search engines: Research articles. *J. Am. Soc. Inf. Sci. Technol.*, 58(6):862–871, 2007.
86. Bernard J. Jansen, Amanda Spink, and Sherry Koshman. Web searcher interaction with the dogpile.com metasearch engine. *J. Am. Soc. Inf. Sci. Technol.*, 58:744–755, March 2007.

87. Bernard J. Jansen, Amanda Spink, and Bhuva Narayan. Query modifications patterns during web searching. In *Proceedings of the International Conference on Information Technology*, ITNG '07, pages 439–444, Washington, DC, USA, 2007. IEEE Computer Society.
88. Bernard J. Jansen, Mimi Zhang, and Amanda Spink. Patterns and transitions of query reformulation during web searching. *IJWIS*, 3(4):328–340, 2007.
89. Anni Järvelin, Antti Järvelin, and Kalervo Järvelin. s-grams: Defining generalized n-grams for information retrieval. *IPM*, 43(4):1005 – 1019, 2007.
90. Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
91. R. Jones, R. Kumar, B. Pang, and A. Tomkins. I know what you did last summer: query logs and user privacy. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 909–914. ACM, 2007.
92. Rosie Jones and Kristina L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *CIKM '08*, pages 699–708. ACM, 2008.
93. Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proc. WWW'06*. ACM, 2006.
94. J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.
95. Jon Kleinberg. Bursty and hierarchical structure in streams. In *Proc. KDD'02*. ACM, 2002.
96. Sherry Koshman, Amanda Spink, and Bernard J. Jansen. Web searching on the vivisimo search engine. *J. Am. Soc. Inf. Sci. Technol.*, 57:1875–1887, December 2006.
97. M. Koster. ALIWEB-Archie-like indexing in the Web. *Computer Networks and ISDN Systems*, 27(2):175–182, 1994.
98. R. Kumar, J. Novak, B. Pang, and A. Tomkins. On anonymizing query logs via token-based hashing. In *Proceedings of the 16th international conference on World Wide Web*, pages 629–638. ACM, 2007.
99. Tessa Lau and Eric Horvitz. Patterns of search: analyzing and modeling web query refinement. In *Proceedings of the seventh international conference on User modeling*, pages 119–128, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.
100. Ronny Lempel and Shlomo Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 19–28, New York, NY, USA, 2003. ACM.
101. Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Gabriele Tolomei. Identifying task-based sessions in search engine query logs. In *Proc. WSDM'11*, pages 277–286, New York, NY, USA, 2011. ACM.
102. Hao Ma, Michael R. Lyu, and Irwin King. Diversifying query suggestion results. In *Proc. AAAI'10*. AAAI, 2010.
103. C.D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press New York, NY, USA, 2008.
104. C.D. Manning, H. Sch
"utze, and MITCogNet. *Foundations of statistical natural language processing*, volume 59. MIT Press, 1999.
105. Evangelos P. Markatos. On caching search engine query results. In *Computer Communications*, page 2001, 2000.
106. Mazlita Mat-Hassan and Mark Levene. Associating search and navigation behavior through log analysis: Research articles. *J. Am. Soc. Inf. Sci. Technol.*, 56:913–934, July 2005.

107. O.A. McBryan. GENVL and WWWW: Tools for Taming the Web. In *Proceedings of the First International World Wide Web Conference*, volume 341. Citeseer, 1994.
108. Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *Proc. CIKM'08*. ACM, 2008.
109. S. Muthukrishnan. Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, 2005.
110. G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.
111. H. Cenk Ozmutlu and Fatih Çavdur. Application of automatic topic identification on excite web search engine data logs. *IPM*, 41(5):1243–1262, 2005.
112. H. Cenk Ozmutlu, Amanda Spink, and Seda Ozmutlu. Analysis of large data logs: an application of poisson sampling on excite web queries. *Inf. Process. Manage.*, 38:473–490, July 2002.
113. S. Ozmutlu, H.C. Ozmutlu, and A. Spink. Multitasking Web searching and implications for design. *Proceedings of the American Society for Information Science and Technology*, 40(1):416–421, 2003.
114. Seda Ozmutlu, Amanda Spink, and Huseyin C. Ozmutlu. A day in the life of web searching: an exploratory study. *Inf. Process. Manage.*, 40:319–345, March 2004.
115. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
116. Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems*, InfoScale '06, New York, NY, USA, 2006. ACM.
117. A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, volume 2007. Citeseer, 2007.
118. Benjamin Piwowarski and Hugo Zaragoza. Predictive user click models based on click-through history. In *Proc. CIKM'07*. ACM, 2007.
119. Barbara Poblete, Myra Spiliopoulou, and Ricardo A. Baeza-Yates. Privacy-preserving query log mining for business confidentiality protection. *TWEB*, 4(3), 2010.
120. F. Radlinski, M. Szummer, and N. Craswell. Inferring query intent from reformulations and clicks. In *Proceedings of the 19th international conference on World wide web*, pages 1171–1172. ACM, 2010.
121. Filip Radlinski and Susan Dumais. Improving personalized web search using result diversification. In *Proc. SIGIR'06*. ACM, 2006.
122. Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Proc. KDD'05*. ACM Press, 2005.
123. Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proc. ICML'08*. ACM, 2008.
124. Davood Rafiei, Krishna Bharat, and Anand Shukla. Diversifying web search results. In *Proc. WWW'10*. ACM Press, 2010.
125. Jasson D. M. Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd ICML 2005 Conference*, pages 713–719, New York, NY, USA, 2005. ACM.
126. Matthew Richardson. Learning about the world through long-term query logs. *ACM TWEB*, 2(4):1–27, 2008.
127. S.E. Robertson and S. Walker. Okapi/keenbow at trec-8. *NIST SPECIAL PUBLICATION SP*, pages 151–162, 2000.
128. Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.

129. E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *Proceedings of the 19th international conference on World wide web*, pages 841–850. ACM, 2010.
130. M. Sanderson and S. Dumais. Examining repetition in user search behavior. *Advances in Information Retrieval*, pages 597–604, 2007.
131. Rodrygo Santos, Craig Macdonald, and Iadh Ounis. Exploiting query reformulations for web search result diversification. In *Proc. WWW'10*. ACM Press, 2010.
132. Rodrygo Santos, Jie Peng, Craig Macdonald, and Iadh Ounis. Explicit search result diversification through sub-queries. In *Proc. ECIR'10*. ACM Press, 2010.
133. Rodrygo L.T. Santos, Craig Macdonald, and Iadh Ounis. Selectively diversifying web search results. In *Proc. CIKM'10*, New York, USA, 2010. ACM.
134. Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. WWW'01*. ACM, 2001.
135. A. Scime. *Web Mining: Applications and Techniques*. IGI Publishing Hershey, PA, USA, 2004.
136. Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating "word of mouth". In *Proc. SIGCHI'95*. ACM, 1995.
137. Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Q2C@UST: our winning solution to query classification in kddcup 2005. *SIGKDD Explor. Newsl.*, 7:100–110, December 2005.
138. Xuehua Shen, Bin Tan, and Chengxiang Zhai. Implicit user modeling for personalized search. In *CIKM '05*, pages 824–831, New York, NY, USA, 2005. ACM Press.
139. S. Siegfried, M.J. Bates, and D.N. Wilde. A profile of end-user searching behavior by humanities scholars: The Getty Online Searching Project Report No. 2. *Journal of the American Society for Information Science*, 44(5):273–291, 1993.
140. Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33:6–12, September 1999.
141. Fabrizio Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 1(1-2):1–174, 2010.
142. Fabrizio Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 1(1-2):1–174, 2010.
143. Barry Smyth. A community-based approach to personalizing web search. *Computer*, 40(8):42–50, 2007.
144. Barry Smyth, Evelyn Balfe, Oisin Boydell, Keith Bradley, Peter Briggs, Maurice Coyle, and Jill Freyne. A live-user evaluation of collaborative web search. In *IJCAI*, 2005.
145. Yang Song and Li-wei He. Optimal rare query suggestion with implicit user feedback. In *Proc. WWW'10*. ACM, 2010.
146. A. Spink and T. Saracevic. Interaction in information retrieval: selection and effectiveness of search terms. *Journal of the American Society for Information Science*, 48(8):741–761, 1997.
147. Amanda Spink, Bernard J. Jansen, Dietmar Wolfram, and Tefko Saracevic. From e-sex to e-commerce: Web search changes. *Computer*, 35:107–109, March 2002.
148. Amanda Spink, H. Cenk Ozmutlu, and Daniel P. Lorence. Web searching for sexual information: an exploratory study. *Inf. Process. Manage.*, 40:113–123, January 2004.
149. Amanda Spink, Minsoo Park, Bernard J. Jansen, and Jan Pedersen. Multitasking during web search sessions. *IPM*, 42(1):264–275, 2006.
150. Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. Searching the web: the public and their queries. *J. Am. Soc. Inf. Sci. Technol.*, 52:226–234, February 2001.

151. Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: discovery and applications of usage patterns from web data. *SIGKDD Explor. Newsl.*, 1:12–23, January 2000.
152. Yizhou Sun, Kunqing Xie, Ning Liu, Shuicheng Yan, Benyu Zhang, and Zheng Chen. Causal relation of queries from temporal logs. In *In Proc. WWW'07*, pages 1141–1142, New York, NY, USA, 2007. ACM Press.
153. L. Sweeney. k-ANONYMITY: a model for protecting privacy. *World*, 10(5):557–570, 2002.
154. Jaime Teevan, Eytan Adar, Rosie Jones, and Michael Potts. History repeats itself: repeat queries in yahoo's logs. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 703–704, New York, NY, USA, 2006. ACM.
155. Jaime Teevan, Eytan Adar, Rosie Jones, and Michael A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 151–158, New York, NY, USA, 2007. ACM.
156. Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *Proc. ICDM'06*. IEEE CS, 2006.
157. L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*. AAAI Press, Menlo Park California, 1998.
158. Erik Vee, Utkarsh Srivastava, Jayavel Shanmugasundaram, Prashant Bhat, and Sihem Amer Yahia. Efficient computation of diverse query results. In *Proc. ICDE'08*. IEEE CS, 2008.
159. V.S. Verykios, E. Bertino, I.N. Fovino, L.P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *ACM Sigmod Record*, 33(1):50–57, 2004.
160. M. Vlachos, P.S. Yu, V. Castelli, and C. Meek. Structural periodic measures for time-series data. *Data Mining and Knowledge Discovery*, 12(1):1–28, 2006.
161. Michail Vlachos, Suleyman S. Kozat, and Philip S. Yu. Optimal distance bounds for fast search on compressed time-series query logs. *ACM Trans. Web*, 4:6:1–6:28, April 2010.
162. Michail Vlachos, Christopher Meek, Zografoula Vagena, and Dimitrios Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *In Proc. SIGMOD'04*, pages 131–142, New York, NY, USA, 2004. ACM.
163. David Vogel, Steffen Bickel, Peter Haider, Rolf Schimpfky, Peter Siemen, Steve Bridges, and Tobias Scheffer. Classifying search engine queries using the web as background knowledge. *SIGKDD Explor. Newsl.*, 7:117–122, December 2005.
164. Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proc. SIGIR'06*. ACM, 2006.
165. Ji-Rong Wen, Jian-Yun Nie, and Hong-Jiang Zhang. Clustering user queries of a search engine. In *Proc. WWW'01*. ACM, 2001.
166. Ryen W. White, Mikhail Bilenko, and Silviu Cucerzan. Studying the use of popular destinations to enhance web search interaction. In *In Proc. SIGIR'07*, pages 159–166, New York, NY, USA, 2007. ACM.
167. L. Xiong and E. Agichtein. Towards privacy-preserving query log publishing. In *Query Log Analysis: Social And Technological Challenges Workshop in WWW*, 2007.
168. J. Xu and W.B. Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Transactions on Information Systems (TOIS)*, 18(1):79–112, 2000.

169. Jack L. Xu and Amanda Spink. Web research: The excite study. In Gordon Davies and Charles B. Owen, editors, *WebNet*, pages 581–585. AACE, 2000.
170. O. Za
"ıane and A. Strilets. Finding similar queries to satisfy searches based on query traces. *Advances in Object-Oriented Information Systems*, pages 349–359, 2002.
171. Oren Zamir and Oren Etzioni. Grouper: a dynamic clustering interface to web search results. In *Proc. WWW'99*. Elsevier North-Holland, Inc., 1999.
172. Cheng Xiang Zhai, William W. Cohen, and John Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *Proc. SIGIR'03*. ACM, 2003.
173. ChengXiang Zhai. *Risk minimization and language modeling in Information Retrieval*. PhD thesis, CMU, 2002.
174. ChengXiang Zhai and John Lafferty. A risk minimization framework for information retrieval. *IP&M*, 42(1):31–55, 2006.
175. Ying Zhang, Bernard J. Jansen, and Amanda Spink. Time series analysis of a web search engine transaction log. *Inf. Process. Manage.*, 45:230–245, March 2009.
176. Yuye Zhang and Alistair Moffat. Some observations on user search behavior. In *Proceedings of the 11th Australasian Document Computing Symposium*. Brisbane, Australia, 2006.
177. Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *Proceedings of the 15th international conference on World Wide Web*, pages 1039–1040. ACM, 2006.
178. Q. Zhao, S.C.H. Hoi, T.Y. Liu, S.S. Bhowmick, M.R. Lyu, and W.Y. Ma. Time-dependent semantic similarity measure of queries using historical click-through data. In *Proceedings of the 15th international conference on World Wide Web*, pages 543–552. ACM, 2006.
179. Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. *Advances in Neural Information Processing Systems*, 19, 2007.
180. Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proc. WWW'05*. ACM, 2005.