Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–119

DOI: xxxxxx

now
the essence of knowledge

Mining Query Logs: Turning Search Usage Data into Knowledge

Fabrizio Silvestri[1]

[1] *ISTI - CNR, via G. Moruzzi, 1 – 56124 Pisa, Italy – fabrizio.silvestri@isti.cnr.it*

Abstract

Web search engines have stored in their logs information about users since they started to operate. This information often serves many purposes. The primary focus of this survey is on introducing to the discipline of query mining by showing its foundations and by analyzing the basic algorithms and techniques that are used to extract useful knowledge from this (potentially) infinite source of information. We show how search applications may benefit from this kind of analysis by analyzing popular applications of query log mining and their influence on user experience. We conclude the paper by, briefly, presenting some of the most challenging current open problems in this field.

Contents

1 Introduction **1**

1.1 Web Search Engines 1
1.2 Sketching the Architecture of a Web Search Engine 3
1.3 Fun Facts about Queries 7
1.4 Privacy Issues in Query Log Mining 9

2 The Nature of Queries **11**

2.1 Basic Statistics 12
2.2 Trends and Historical Changes in Queries 17
2.3 Summary 21

3 User Interactions **23**

3.1 Search Sessions 24
3.2 Social Networks from Query-Click relations 28
3.3 Summary 30

4 Enhancing Effectiveness of Search Systems **31**

4.1 Historical Notes and Preliminaries 33
4.2 Query Expansion 35
4.3 Query Suggestion 39
4.4 Personalized Query Results 44
4.5 Learning to Rank 55
4.6 Query Spelling Correction 66
4.7 Summary 70

5 Enhancing Efficiency of Search Systems **72**

5.1 Caching 72
5.2 Index Partitioning and Querying in Distributed Web search Systems 85
5.3 Summary 101

6 New Directions **102**

6.1 Eye Tracking 102
6.2 Web Search Advertisement 103
6.3 Time-series Analysis of Queries 104
6.4 Summary 106

Conclusions **107**

Acknowledgements **108**

References **109**

1

Introduction

"History teaches everything, even the future."
– Alphonse de Lamartine, speech at Macon 1847.

Think about it, for a moment: after checking e-mails, and checking your favorite on-line newspaper and comic strip, what is the first thing you do when connected to the web? You probably open a search engine and start looking for some information you might need either for work or for leisure: news about your favorite actor, news about presidential candidates, and so on.

Even though they are quite rooted in our lives, web search engines are quite new on the scene.

Query Log Mining is a branch of the more general *Web Analytics* [113] scientific discipline. Indeed, it can be considered a special type of web usage mining [214] 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.* [11]"

In particular, query log mining is concerned with all those techniques aimed at discovering interesting patterns from query logs of web search engine with the purpose of enhancing either effectiveness or efficiency of an online service provided through the web.

Keep into account that 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 [108].

1.1 Web Search Engines

Systems that can be considered similar to modern web search engines started to operate around 1994. The now-defunct *World Wide Web Worm* (*WWWW*) [148] created by Oliver McBryan at the University of Colorado, and the *AliWeb* search engine [127] 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 (just to name a few). Nowadays, searching is considered one of the most useful application on the web. As reported in 2005 by *Pew Research Center for The People & The Press* [163]:

> *"search engines have become an increasingly important part of the online experience of American internet users. The most recent findings from Pew Internet & American Life tracking surveys and consumer behavior trends from the comScore Media Metrix consumer panel show that about 60 million American adults are using search engines on a typical day"* [190].

Even if this quote dates back to 2005, it is very likely that those survey results are still valid (if not still more positives for search engines). On the other side of the coin, search engines' users are satisfied by their search experience [191].

In a paper overviewing the challenges in modern web search engines' design, Baeza-Yates *et al.* [14] 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 application, 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.

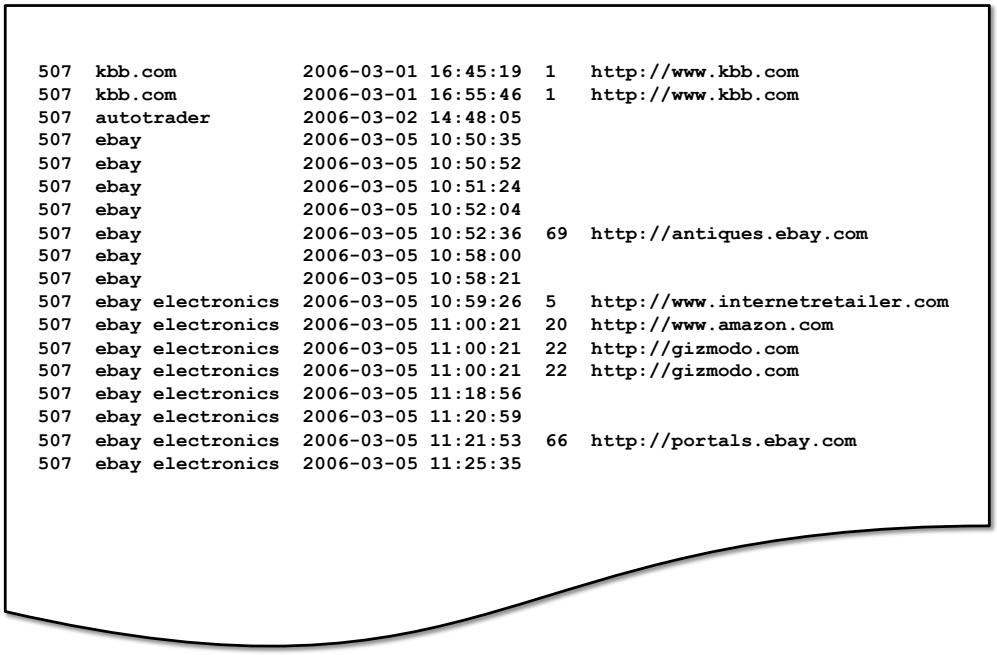
Web search engines are part of a broader class of software systems, namely Information Retrieval (IR) Systems. Basically, IR systems were born in the early 1960s due to two major application needs. Firstly, allowing searching through digital libraries. Secondly, the need for computer users to search through the data they were collecting in their own digital repositories.

Intuitively, an IR system is a piece of software whose main purpose is to return a list of documents in response to a user query. Thus far, this description makes IR systems similar to what a DB system is. 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. Just to make a very simple 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. Even worse, *polysemy* also happens. In Spanish the word *meta* means *goal*.

To this extent, a web search engine is in all respects an IR system [221] only on a *very large scale*. The uncertainty in users' intent is also present in web search engines. Differently from smaller scale IR systems, though, web IR systems can rely on the availability of a huge amount of usage information stored in *query logs*.

One of the most used ways of enhancing the users' search experience, in fact, is the exploitation of the knowledge contained within past queries. A query log, typically, contains information about users, issued queries, clicked results, etc. From this information knowledge can be extracted to improve the quality (both in terms of effectiveness and efficiency) of their system. Figure 1.1 shows a fragment of the AOL query log. The format of this query log represents a record using five features: user id, query, timestamp, rank of the clicked result, host string of the clicked URL.

How query logs interact with search engines has been studied in many papers. For a general overview, [12, 20] are good starting point references.



```
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. 1.1: A fragment of the AOL query log [162].

In this paper we review some of the most recent techniques dealing with query logs and how they can be used to enhance web search engine operations. We are going to summarize the basic results concerning query logs: analyses, techniques used to extract knowledge, most remarkable results, most useful applications, and open issues and possibilities that remain to be studied.

The purpose is, thus, to present ideas and results in the most comprehensive way. We review fundamental, and state-of-the-art techniques. In each section, even if not directly specified, we review and analyze the algorithms used, not only their results. This paper is intended for an audience of people with basic knowledge of computer science. We also expect readers to have a basic knowledge of Information Retrieval. Everything not at a basic level is analyzed and detailed.

Before going on, it is important to make clear that all the analyses and results reported were not reproduced by the author. We only report results as stated in the papers referenced. In some cases we slightly adapted them to make concepts clearer.

1.2 Sketching the Architecture of a Web Search Engine

A search engines is one of the most complicated pieces of software a company may develop. Consisting of tens of interdependent modules, it represents one of the toughest challenge in today's computer engineering world.

Many papers and books sketch the architecture of web search engines. For example Barroso *et al.* [33] present the architecture of Google as it was in 2003. Other search engines are believed to have similar architectures. When a user enters a query, the user's browser builds a URL (for example `http://www.google.com/search?q=foundations+trends+IR`). 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 a HTTP request to the selected data-center, and thereafter, the query processing is entirely local to that center. After the query is answered by the local data-center, the result is returned in the form of a HTML page, to the originating client.

Figure 1.2 shows they way the main modules of a web search engine are connected.

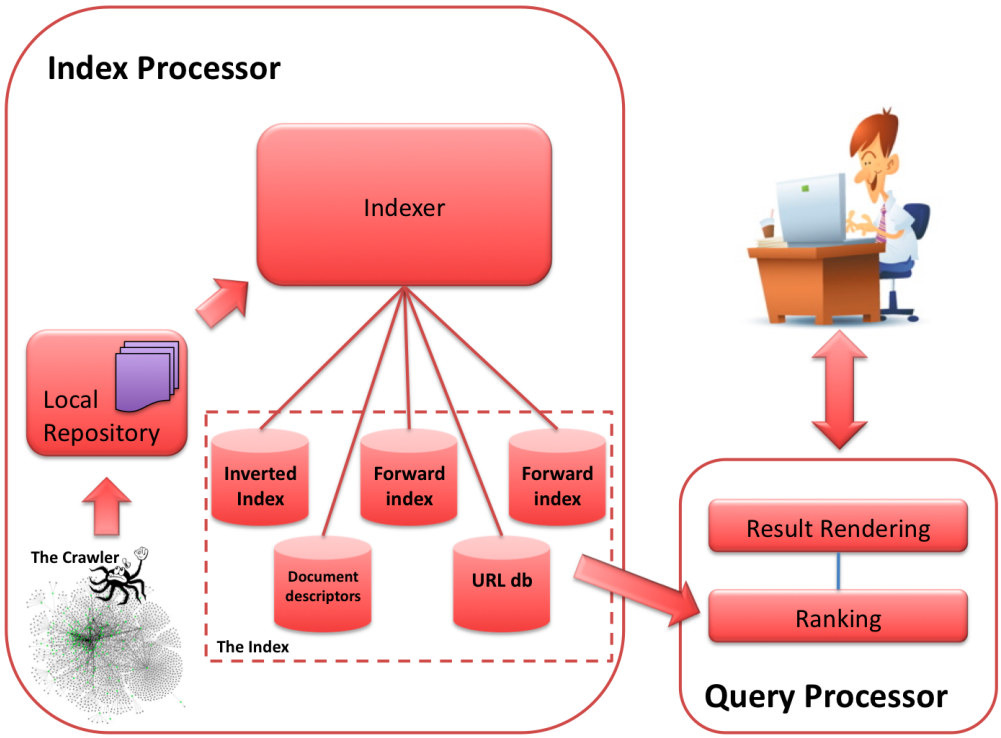


Fig. 1.2: The typical structure of a web search engine. Note that throughout the text IR core, and query server will be used interchangeably.

Web search engines get their data from different sources: the web (primarily), Image and video repositories (e.g. Flickr, or YouTube), etc. In particular, in the case of web content, 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 [159], or HITS [125]). In modern web retrieval systems, crawlers continuously run and download pages from the web updating incrementally the content of the document cache. For more information on crawling, interested readers can refer to Castillo's Ph.D. thesis on web Crawling [57].

The textual (i.e. hypertextual) 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. The format of the index is the subject of Chapter 5 where we review some of the most used techniques for optimizing index allocation policies.

Usually in real systems the design is tailored to favor aggregate request throughput not peak server response time [33].

In real-world search engines, the index is distributed among a set of *query servers* coordinated by a *broker*. The broker, accepts a query from the 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. Figure 1.3 shows the interactions taking place among query servers and the broker.

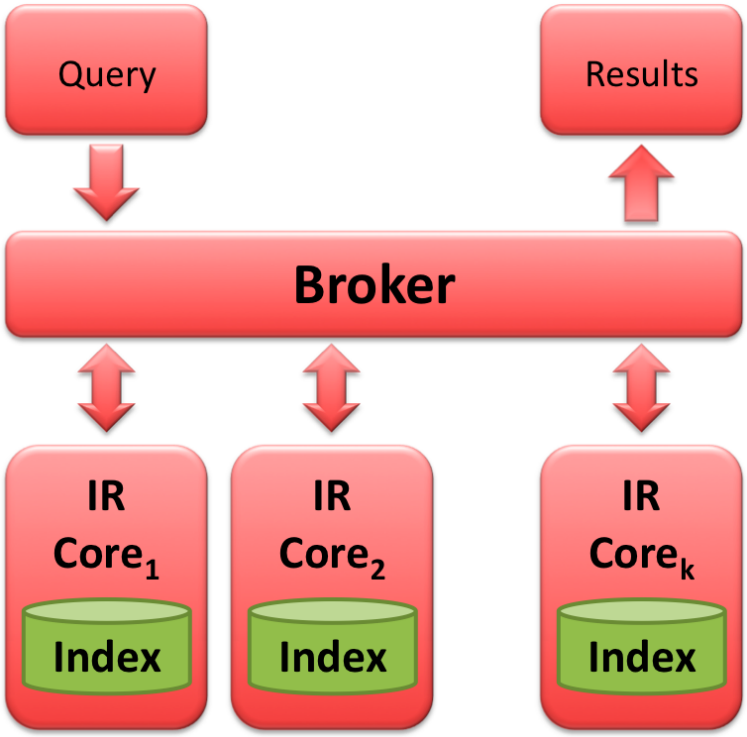


Fig. 1.3: The typical structure of a distributed web search engine.

The Broker is usually the place where queries are grabbed and stored in the query logs. A module dedicated to analyze past queries is also usually available within the architecture components.

1.2.1 The index

An Inverted File index on a collection of web pages consists of several interlinked components. The principal ones are: the *lexicon*, i.e. the list of all the *index terms* appearing in the collection, and the corresponding set of *inverted lists*, where each list is associated with a distinct term of the lexicon. Each inverted list contains, in turn, a set of *postings*. Each posting collects information about the *occurrences* of the corresponding term in the collection's documents. For the sake of simplicity, in the following discussion we consider that each posting only includes the identifier of

the document (*DocID*) where the term appears, even if postings actually store other information used for document ranking purposes (e.g. in the implementation [204] each posting also includes the positions and the frequency of the term within the document, and context information like the appearance of the term within specific html tags).

Several sequential algorithms have been proposed in the past, which try to balance the use of memory hierarchy in order to deal with the large amount of input/output data involved in query processing. The inverted file index [221] is the data structure typically adopted for indexing the web. This occurs for three reasons. First, an inverted file index allows the efficient resolution of queries on huge collections of web data [246]. In fact, it works very well for common web queries, where the conjunction of a few terms is to be searched for. Second, an inverted file index can be easily compressed to reduce the space occupancy in order to better exploit the memory hierarchy [204]. Third, an inverted file can be easily built using a sort-based algorithm in time complexity that is the same order of a sorting algorithm [246].

Query answering using inverted file is a very straightforward task. We illustrate the basic AND operation and refer to other papers for a thorough analysis of the remaining operations. Given a query as a conjunction of two terms ($t_1 \wedge t_2$), the query resolution proceeds by firstly looking up t_1 and t_2 in the lexicon to retrieve the corresponding inverted lists l_1, and l_2. The result set is then built by intersecting the two lists, thus, returning those documents having the two terms in common. During the intersection step a scoring function is also computed to evaluate the likeliness of a document to be relevant for the query. The top r results are then selected (in typical web search engines r is usually set to 10 results) and successively returned to the users who originated the query. Query processing can be done in two different ways: *Document-At-A-Time* (DAAT), when document lists for terms are scanned contemporary, as opposed to the *Term-At-A-Time* (TAAT) strategy, where each term is considered separately [219].

Another important feature of inverted file indexes is that they can be easily partitioned. Let us consider a typical distributed web search engine: the index can be distributed across the different nodes of the underlying architecture in order to enhance the overall system's throughput (i.e. the number of queries answered per each second). For this purpose, two different partitioning strategies can be devised.

The first approach requires to horizontally partition the whole inverted index with respect to the lexicon, so that each index server stores the inverted lists associated with only a subset of the index terms. This method is also known as *term partitioning* or *global inverted files*. The other approach, known as *document partitioning* or *local inverted files*, requires that each index server becomes responsible for a disjoint subset of the whole document collection (vertical partitioning of the inverted index). Figure 1.5 graphically depicts such partitioning schemes.

The construction of a document-partitioned inverted index is a two-staged process. In the first stage each index partition is built locally and independently from a partition of the whole collection. The second phase collects global statistics computed over the whole inverted index. One of the most valuable advantages of document partitioning is the possibility of easily performing updates. In fact, new documents may simply be inserted into a new partition to independently index separately from the others [171].

Since the advent of web search engines, a large number of papers have been published describing different architectures for search engines, and search engine components [47, 25, 10, 155, 33, 98, 99,

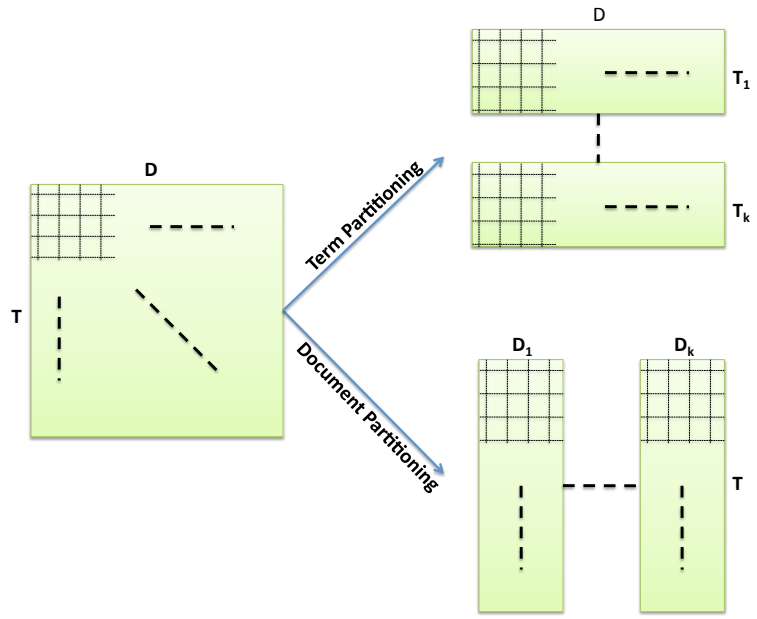


Fig. 1.4: The two different ways of partitioning an inverted index. Rows of the whole $T \times D$ matrix are the lexicon entries, Columns represent the posting lists.

149, 205, 152]. Many other papers [103, 102, 13, 14] enumerate the major challenges search engine developers must address in order to improve their ability to help users in finding information they need. Interested readers shall find in the above referenced papers many interesting insights. Needless to say, you shall not find any particular details, in this survey, about the *real* structure of a search engine. Usually, this kind of information is highly confidential and it is very unlikely that search companies will ever disclose them.

1.3 Fun Facts about Queries

Due to their "commercial importance", finding query logs has always been a difficult task. The very first publicly available query log dates back to 1997. Doug Cutting, representing Excite, a major search service to that date, made available for research a set of user queries as submitted to Excite. Since then, the other query logs made publicly available were the AltaVista log, the TodoBR query log, and the AOL log.

AOL eventually fired employees involved in the public release of their log. This confirms, even more strongly, the particular level of privacy characterizing such data. Obviously, this may sound worse than it is. Search Engine companies are still releasing their data, only that they adopt more conservative policies and release data under research licenses preventing broad distribution.

Figure **??** shows a cloud of the 250 most frequent queried terms in the AOL query log.

Queries posed by users are somewhat entertaining. To have an idea of what every day users search through search engines, consider these queries that were actually extracted from the (in)famous

Fig. 1.5: A cloud of the 250 most frequent queried terms in the AOL query log [162]. Picture has been generated using http://www.wordle.net.

AOL Query Log[1].

In today's hectic world, people often get very stressed. Stress produces distraction and user #427326 probably was a little more stressed than the average. At 2006-04-21 21:16:51, in fact, he was looking for the following sentence "where is my computer". Well, probably is closer than what you were suspecting. Actually, searching for this sentence on popular search engines result in around 200,000 results. Gosh! Many stressed people out there![2]

Again, people gets stressed easily today. I dare you to guess what was user #582088 looking for by entering the following keywords "can you hear me out there i can hear you i got you i can hear you over i really feel strange i wanna wish for something new this is the scariest thing ive ever done in my life who do we think we are angels and airwaves im gonna count down till 10 52 i can". Hint: try by yourself and enter the above sentence. What is the result? In your opinion, what was user doing while typing the query?

Search engines publish some of the most interesting submitted queries. *Interestingness*, here, is a relative concept. Depending on the search engine company, interesting may mean different things. At Google, for instance, Zeitgeist[3] is a

> "*cumulative snapshot of interesting queries people are asking – over time, within country domains, and some on Google.com – that perhaps reveal a bit of the human condition.*"

Zeitgeist does not reveal the most searched queries, but only those having had a "sudden", and "unexpected" raise in popularity. For instance, late in 2007 Italian Zeitgeist ranked "*federico calzolari*"[4] as the most "inflated" query was. Many (mainly Italian) newspapers, and blogs started

[1] `http://www.techcrunch.com/2006/08/06/aol-proudly-releases-massive-amounts-of-user-search-data/`
[2] Indeed, many results are on people asking where is "My Computer" icon on their desktop.
[3] `http://www.google.com/press/zeitgeist.html`
[4] `http://googleitalia.blogspot.com/2007/12/zeitgeist-di-novembre.html` (in Italian)

to ask who is the person referred to in the query. The name was that of a Ph.D. student in Pisa that periodically queried Google for his name. This resulted in an unexpected raise in popularity for the query term thus ending up in the Zeitgeist. Many people, mainly journalists, started to discuss whether or not Federico Calzolari has hacked the Google ranking algorithm.

It is important to point out that the discussion above seems to imply that one could guess the intent of the users by looking at query session. This is far from being true. As it is shown later on, the identification of users' tasks is a very challenging activity. The main goal of this paragraph is to make readers aware of: (i) the variety of information in query logs, and (ii) the detail that, in principle, can be obtained about a single user.

An interesting recent paper dealing in a scientific way with discovering information about search engine index content by carefully probing it using queries out of a query log is Bar-Yossef and Gurevich [28].

1.4 Privacy Issues in Query Log Mining

The most recent scandal concerning privacy and query logs happened in 2006 at AOL. AOL compiled a statistical sampling of more than twenty million queries entered by more than 650,000 of their customers, and then made this DB available to the public for research purposes. While user names were replaced by numbers, these numbers provide a "thread" by which queries by a given user could be identified so that if, for example, a user entered some piece of information which permits their identity to be discerned, all the other queries they made during the sampling period could be identified as theirs. AOL received so much criticism for releasing this data that it eventually fired two employees. The real problem was that they released ALL off the data to EVERYONE. A Non-Disclosure-Agreement form for researchers to sign, would have saved a lot of pain to AOL people that were fired after the mishap.

Many commercial search engines overcome to this problem by simply not publishing their logs. Is this approach good? Yes for some reasons, no for others. Roughly speaking, it is good that people (in general) cannot access query log data. As already said above they might be used to infer users' preferences, tastes, and other personal information that might be used against their will. On the other hand, as pointed also out by Judit Bar-Ilan in [27]

> *"[...] interesting results can be obtained from query logs without jeopardizing the privacy of the users."*

While Bar-Ilan showed that it is possible to sanitize a query log in order to prevent private information to be disclosed, Jones *et al.* in [120] showed that even heavily scrubbed query logs, still containing session information, have significant privacy risks.

This paper does not deal with this (extremely important) issue, but we would not have been comfortable without making the reader aware of this issues. More important, we think this would clarify why many studies reported here are made on (sometimes) old and outdated logs, or logs privately held by companies not sharing them.

The interested reader shall find an introduction and some thoughts about privacy and log publishing in recently published papers [166, 230, 1, 129]. Recently, Cooper published a very detailed survey on query log privacy-enhancing techniques [64], readers interested in this topic shall find a very thorough analysis of the most recent techniques dealing with privacy preserving analysis of

query logs.

Recently ASK[5] has given the possibility to users to explicitly deny the storing of their usage data. On the other hand, Google, Yahoo, and Microsoft, continuously ask users for the permission to store their preferences, behaviors, and data in general. What is the most correct behavior? It depends on search engines' policies, thus we do not enter into details on how these are managed.

The remainder of this work presents the most recent results and advances that have used query logs as (the main) source of information. It is worth mentioning here that not always the experiments presented might be reproduced. This is something that in science should be avoided [88]. Unfortunately, as already said above, the main source of knowledge (the query logs) are mainly kept by search engine companies that for many reasons (not last, privacy issues) are very reluctant of give them away, even to scientists. Therefore, many times in this article, the experimental evaluation is based on results obtained by others and presented in the literature. We apologize in advance to both authors of the mentioned papers, and to readers.

Before entering into the details of our survey, it is important to remark that query log mining is a very hot topic nowadays. The material covered by this survey is to be considered as a valid starting point for those interested in knowing something more on the topic. Proceedings of the major conference series (e.g. SIGIR, WWW, SIGMOD, VLDB, SIGKDD, CIKM, etc. Just to name a few) and top journals (e.g. ACM TOIS, ACM TWEB, ACM TKDD, ACM TOIT, Information Processing & Management, JASIST, IEEE TKDE, etc.) are the best source for the state-of-the-art works on this field. Furthermore, we use the same notation used by the authors of the surveyed papers. This, in our opinion, makes each (sub)section of the survey more independent and leave to the reader the possibility of selecting the techniques he is interested on.

That said, let the journey into the marvelous world of queries begin...

[5] `http://www.ask.com`

2

The Nature of Queries

This chapter presents the characterization of query logs by showing how are query distributed over time. In particular, we analyze topic-popularity, term-popularity, differences with respect to the past, variations of topics during day hours, etc.

In the past, there have been studies showing how people interacted with small scale IR systems[1] [105, 77, 199, 212]. Query logs of large scale web search engines and small scale IR systems are fundamentally different. For example, it is well known that queries to web search engines are unlikely to contain more than three terms whereas in IR systems' that are usually used in 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 user. Search operators (like quotes, '+', '-', etc.) are rarely used on the web.

This chapter presents a variety of statistics, along with an analysis of how data are computed. In fact, to deal with a huge number of queries it is also important to design efficient techniques for quickly analyzing such data.

A number of papers have been published describing characteristics of query logs coming from some of the most popular search engines. In these papers [111, 231, 209, 156, 158, 35, 211, 109, 147, 162, 126, 110, 112, 34], many interesting statistics are shown. Some of them are reviewed throughout the chapter. Indeed, results from the query logs are reviewed and presented throughout the whole survey, thus, to make the presentation more clear their main characteristics are summarized in Table 2.1. Analyses on other different logs have been performed by other authors. We present their characteristics when it is required for the sake of clarity in the description of the results or technique presented.

[1] In particular those studies were referring to IR systems not directly dealing with web users. Simply put, when referring to small scale IR systems we are not referring to web search engines like Google or Yahoo!

Query log name	Public	Period	# Queries	# Sessions	# Users
Excite '97	Y	Sep '97	1,025,908	211,063	$\sim 410,360$
Excite '97 (small)	Y	Sep '97	51,473	-	$\sim 18,113$
Altavista	N	Aug 2^{nd} - Sep 13^{th} '98	993,208,159	285,474,117	-
Excite '99	Y	Dec '99	1,025,910	325,711	$\sim 540,000$
Excite '01	Y	May '01	1,025,910	262,025	$\sim 446,000$
Altavista (public)	Y	Sep '01	7,175,648	-	-
Tiscali	N	Apr '02	3,278,211	-	-
TodoBR	Y	Jan - Oct '03	22,589,568	-	-
TodoCL	N	May – Nov '03	-	-	-
AOL (big)	N	Dec 26^{th} '03 – Jan 1^{st} '04	$\sim 100,000,000$	-	$\sim 50,000,000$
Yahoo!	N	Nov '05 – Nov '06	-	-	-
AOL (small)	Y	Mar 1^{st} - May 31^{st} '06	36,389,567	-	-

Table 2.1: The main query logs that have been used by the analyses reviewed in this survey. The dash sign (-) means that the feature in the relative column was non disclosed. Figures are obtained by the papers using them, no analyses has been done for the purpose of this survey.

2.1 Basic Statistics

Typical simple statistics that can be drawn from query logs are: query popularity, term popularity, average query length, and distance between repetitions of queries or terms.

Silverstein *et al.* [200, 201] are the first to analyze a large query log of the Altavista search engine containing about a billion queries submitted in a period of 42 days. The exhaustive analysis presented by the authors point out some interesting results. Tests conducted included the analysis of the query sessions for each user, and of the correlations among the terms of the queries. Similarly to other work, their results show that the majority of the users (in this case about 85%) visit the first page of results only. They also showed that 77% of the users' sessions end up just after the first query. The log contains a huge number of queries and account to 285 million users. As the authors state, a smaller log could be influenced by ephemeral trends in querying (such as searches related to news just released, or to a new record released by a popular singer). For this reason results are considered by Silverstein *et al.* precise and general.

Jansen *et al.* [111] analyzes a log made up of $51,473$ queries posed by $18,113$ users of Excite. In the log users are anonymous. Therefore, the information is completely decontextualized. That is, no user profiling is available. The log from which the experiments are carried out is publicly available to scholars.

Lempel and Moran [132], and Fagni *et al.* [76] surveyed the content of a publicly available Altavista log. They present a log made up of $7,175,648$ queries issued to AltaVista during Summer of 2001. No information is disclosed about the number of users they logged. This second AltaVista log covers a time period almost three years after the first studies presented by Jansen *et al.* and Silverstein *et al.* The log is not as large as the first Altavista log, yet it represent a very nice picture of search engine users.

On average web search queries are quite short. In the case of the 1998 Excite Log, on average a query contained 2.35 terms with less than 4% of the queries having more than 6 terms. In the case of the "private" AltaVista log, the average query length is, again, 2.35 with a standard deviation of 1.74. For the second AltaVista log, instead, the average query length is slightly above 2.55. One of

the possible reason is that, for instance, The web is a general medium, used by different people from different part of the world looking for disparate information. IR systems, in the past, were instead mainly used by professionals and librarian looking for a precise information, thus, they spent more effort in formulating a more elaborate query. In general that study highlights that users querying search engines, i.e. web IR systems, are fundamentally different from people that used vertical (and smaller scale) IR systems.

The distribution of query popularity, and of term popularity, have been shown to follow a power-law. That is 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. Putting it into formula, $y = Kx^{-\alpha}$, where K is a real positive constant corresponding to the query with the highest popularity. Since $\log(y) = -\alpha \log(x) + \log(K)$, by substituting $X = \log(x)$, and $Y = \log(y)$, power-law distributions have the form of a straight line when plotted on a log-log scale. Figure 2.1 shows a plot graphically representing power-laws for various values of the parameter α.

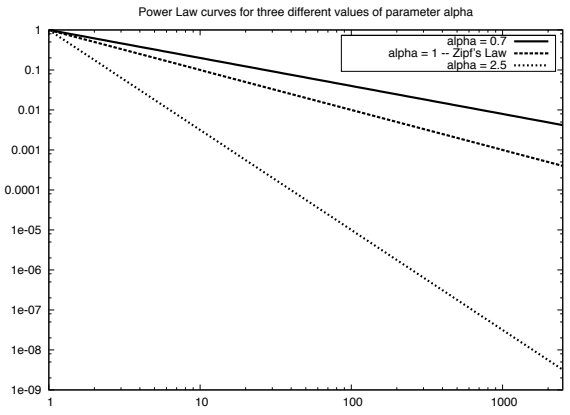
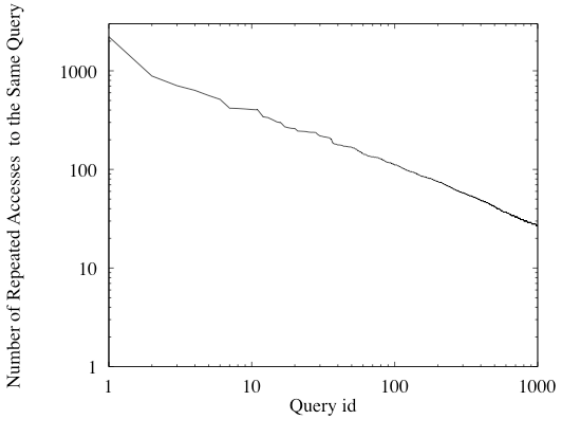


Fig. 2.1: Three examples of power-law curves for different values of the parameter α. The curve corresponding to $\alpha = 1$ is usually said to identify the Zip's law [245].

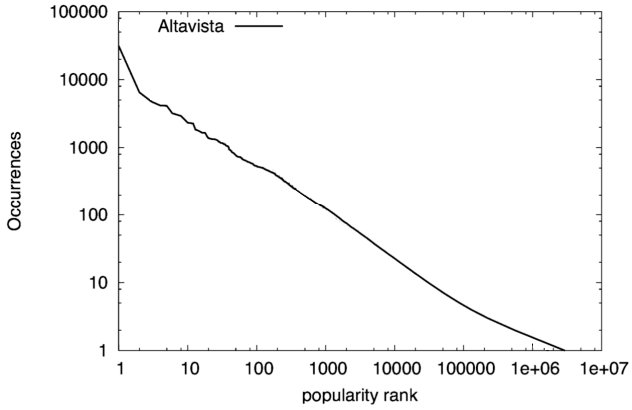
Markatos [75] was the first to show that query popularity follows a power-law with an exponent $\alpha \sim 2.4$. He analyzes the Excite log and plots the occurrence of the first $1,000$ most popular queries. Markatos reports in a graph like the one shown in Figure 2.2a that the popularity follows the usual linear trend in a log-log scale. We can see from the plot that the most popular query is submitted $2,219$ times, while the $1,000$-th most popular query is submitted 27 times [75]. A power-law trend is confirmed also in other studies and other query logs, such as the AltaVista [132] (Figure 2.2b), and Yahoo! [15] (Figure 2.2c) logs.

Tables 2.3a, and 2.3b detail the 20 top queries for the Excite and Altavista logs, respectively[2]. As one would probably have guessed, many queries in both logs refer to sex and sexually explicit topics ("XXX"). While, unsurprisingly, many others can be somewhat related to XXX as well. As often happens, there are some unexpected outcomes in query logs. For instance, rather surprisingly

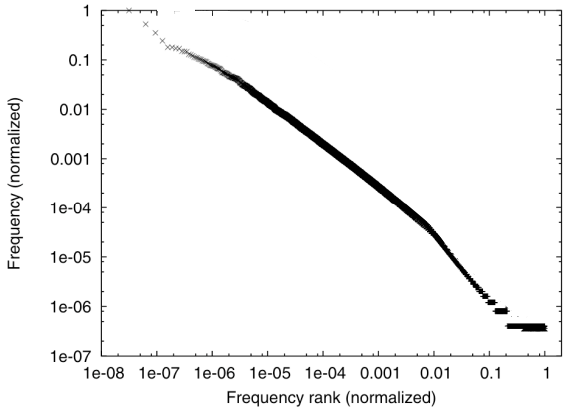
[2] Provisioning of the same information for the Yahoo! log is not possible due to privacy and restriction policies.



(a) The 1,000 most popular queries in the Excite Log [75].



(b) Query popularity of Altavista queries [132].



(c) Query popularity of Yahoo! queries [15].

Fig. 2.2: Plots displaying query popularity for various query logs.

the *most* frequent query in the case of the Excite log is the empty query! They account for more than 5% of the queries. Authors of [111] try to explain this strange fact. Probably, the most obvious reason is that users often wrongly type queries in the search box. This phenomenon could also be due to how search engines react to user actions. For instance, Excite had a link pointing to a "*More Like This*" function that, if clicked, returned pages related to the one selected. Excited counted that as an empty query thus raising the empty query count. Therefore, the frequency of empty query in this logs, could, more likely, identify the usage of the "*More Like This*" feature of Excite.

As it can be seen from Tables in 2.3 many different topics are represented in query logs. Table 2.4a, from [209], shows the percentage of queries submitted for each topic to the Excite search engine in 1997. Categorizing queries into topics is not a simple task. There are papers showing techniques for assigning labels to each query. Recent papers on the topic [94, 193, 224, 36, 37, 49] adopts a set of multiple classifiers subsequently refining the classification phase. Due to space limitations we cannot provide here a complete and detailed analysis of query classification literature. Interested readers can refer to the literature for a thorough analysis of this subject.

Classification of the Excite queries made by Spink *et al.* in [209] shows that in no way is pornography a major topic of web queries, even though the top ranked query terms may indicate

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) Excite.

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) Altavista.

Fig. 2.3: The most popular queries out of the Excite and publicly available Altavista Logs. Potentially offending terms have been replaced by similar terms containing asterisks ('*'). Query have not previously filtered to remove stop-words and terms in queries have not been reordered.

this. Only one in about six web queries have been classified as about sex. Web users look interested on a wide range of different topics. Commerce, including travel, employment, and a number of economic matters are also high on the list. Close to 10% of queries are about health and the sciences.

Authors of [36], and [34] show similar results on a different query log. The log is made up of billions of web queries constituting the total query traffic for a 6-month period of AOL, a general-purpose commercial web search service. Categories are different, and results (in terms of category percentages breakdown) are slightly different. This difference is very likely due to the different period of time in which the analysis was conducted. While the Excite log is of 1997, the AOL log is of 2003, which is more than 6 years after. In particular, as noticed in other works, porn queries fell considerably (unless queries pertaining to "Other" category can be associated with XXX interests).

Terms (as atomic constituents of queries) are distributed according to a power-law as well (in particular a double-pareto log-normal distribution). In fact, the curve of term distribution is steeper, denoting that the most frequent terms are much more frequent than the rest of terms. Just as an example, Figure 2.5 shows log-log plots of the term popularity distribution in the case of the same three query logs, namely: Excite, Altavista, and Yahoo!.

An interesting statistic to draw from query logs is how terms co-occur. In [213], a follow-up of the work presented in [111], Spink *et al.* present a table of the fifty most frequently co-occurrent terms. We report here, for the sake of completeness, their results in Table 2.2.

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) Excite [209].

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) AOL [34].

Fig. 2.4: Distribution of query samples across general topic categories for two different query logs. Excite 2.4a, and AOL 2.4b.

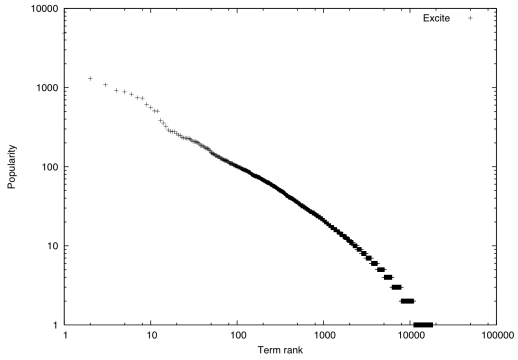
and-and	6,116	of-and	690	or-or	501	women-nude	382	sex-pics	295
of-the	1,901	pictures-of	637	sex-pictures	496	pics-nude	380	north-carolina	295
pics-free	1,098	how-to	627	nude-pictures	486	of-department	365	free-teen	293
university-of	1.018	and-the	614	for-sale	467	united-states	361	free-porn	290
new-york	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 2.2: List of the fifty most co-occurring terms in the Excite log ($term_1$ - $term_2$ frequency) [213].

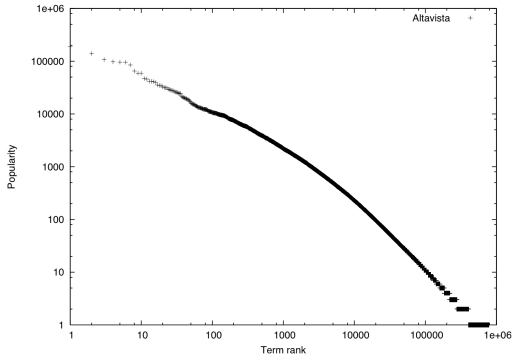
The above mentioned table only shows how terms co-occur in queries without reflecting topic popularity. In fact, the majority of term pairs are about non-XXX topics while in the same analysis they found that XXX queries were highly represented. This could, for instance, indicate that for some topics people use more terms to search for precise information, for other topics the same user need can be satisfied by short queries.

As it has been seen, queries repeat themselves. Since many queries are seen only a few times, one would expect that in the majority of the cases the distance between subsequent submissions of the same query would be very large. The distance, in terms of queries, with which queries are submitted again is shown in Figure 2.6.

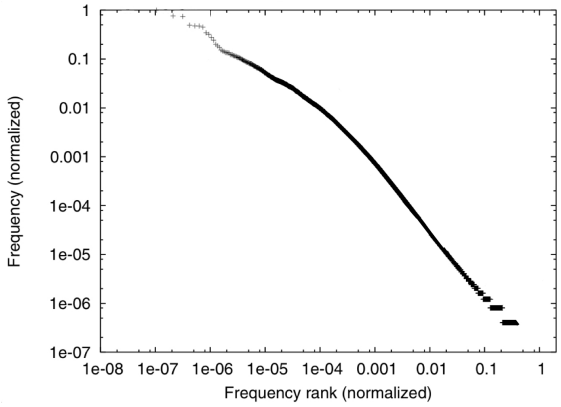
Differently from what is expected, the majority of queries have distances that are less than $1,000$ queries. A possible reason is the inherent bursty [124] nature of query logs: a large number of people start looking for a topic almost at the same time. This observation is very important, as we show in the rest of the survey that 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.



(a) Excite.



(b) Altavista.



(c) Yahoo! (from [15]).

Fig. 2.5: Plots displaying the number of requests for terms in various query logs.

2.2 Trends and Historical Changes in Queries

Queries are issued on several different topics [162] depending also on the historical period [209]. Going at a daily granularity level of analysis, some of the topics are more popular in an hour than in another [35, 34].

During the daytime frequency of querying varies considerably. Ozmutly *et al.* [158] analyze query frequency against arrival time for the Excite query log in a time period ranging from 9AM to 5PM. Table 2.3 shows how frequencies are distributed within hours of the day.

Querying activity is higher during the first hours of the day than the afternoon. There is a sharp decrease in the number of queries submitted going down from 679 at 9AM to 224 at 4PM. That is 30% of the queries that are usually submitted at 9AM. These numbers are small if compared to the number of queries submitted to today search engines. When compared to the same statistics in 2006 [162], results are completely turned upside-down. At 9AM queries submitted are almost half of those submitted at 5PM (Figure 2.7).

Spink *et al.* [209] showed how time periods affects querying behavior of users. In Table 2.4,

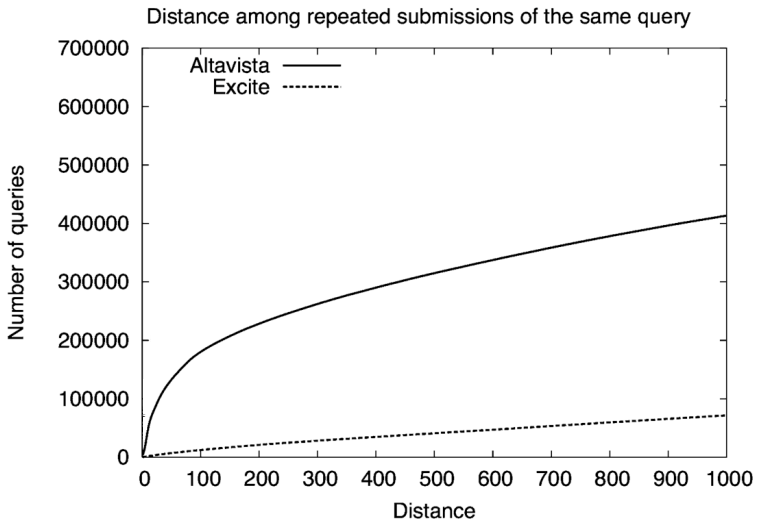


Fig. 2.6: Distances (in number of queries) between subsequent submissions of the same query for the Altavista and Excite log.

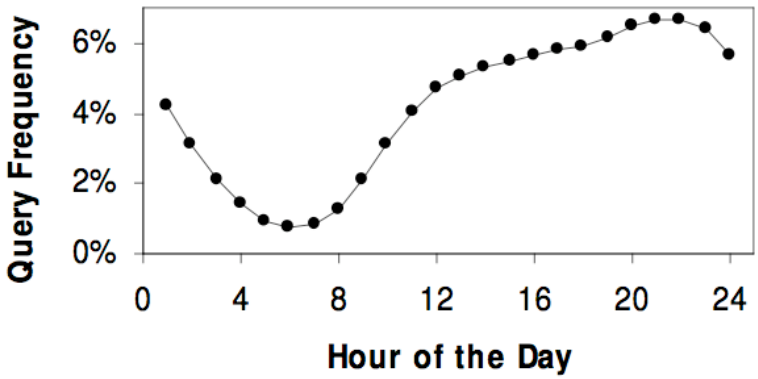


Fig. 2.7: Frequency of query submitted to the AOL search engine during the day [162].

extracted from the above mentioned paper, it is possible to observe that querying behavior is not changed from a statistical point of view, in a period of 4 years. The mean number of terms per query is only slightly raised in 2001, while the number of terms per query, the main queries per user, are basically, unchanged in four years. Even if this study dates back to 2001, it is very likely that the results are still valid today. Users mostly tend to look for places to buy goods, or to look for particular sites they *already* know. For this reason, the number of keyword is usually low.

As it has been shown above, users have changed their preferences and inclinations during time. Obviously, the more penetrated a new technology is the more users become skilled and acquainted with using it. Probably users' understanding of the potentiality of this new medium, the web, has

Hour of the Day	frequency
9:00 – 10:00	679
10:00 – 11:00	486
11:00 – 12:00	437
12:00 – 13:00	367
13:00 – 14:00	358
14:00 – 15:00	333
15:00 – 16:00	304
16:00 – 17:00	224

Table 2.3: Number of query arrivals with respect to hours of the day — Excite query set [158].

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 2.4: Comparative statistics for Excite web queries [209].

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 2.5: Comparison of categories breakdown (in %) for Excite web queries (from 1997 to 2001), and Altavista (2002) [110].

made them prone to use it as a way of conducting business.

From the data in Table 2.5 it is evident that users (at least those of US-based search engines) querying for People, Place or Things was accounting for nearly 50% in 2002. Moreover, there is a clear rise in interest from users for this category: back in 1997 queries referring to People, Place or Things accounted for less than 7%. The 25% of users in 2002 queries for Commerce, Travel, Employment or Economy and Computers, Internet or Technology. This percentage has seen an "up-and-down" trend[3], varying from a minimum of about 25% and to a maximum of 35%. A steady falling trend, instead, is that of sex and pornography: it was accounting for almost the 17% of queries back in 1997, whereas in 2002 the percentage of users looking for Sex related information

[3] Unless due to the use of different classification algorithms for the different query logs.

felt down to 3.3%.

Going to a finer detail level, in [34] Beitzel *et al.* measure the relative popularity of different categories over the hours in a day. The percentage of total query volume broken-down to a selected group of category can be seen in Figure 2.8. Clearly different topical categories are more and less popular at different times of the day. For instance, Personal and Finance popularity raises during the first hours of the morning, between 7 and 10 a.m.; whereas Porn is a category whose popularity raises during late-night until 6 a.m.

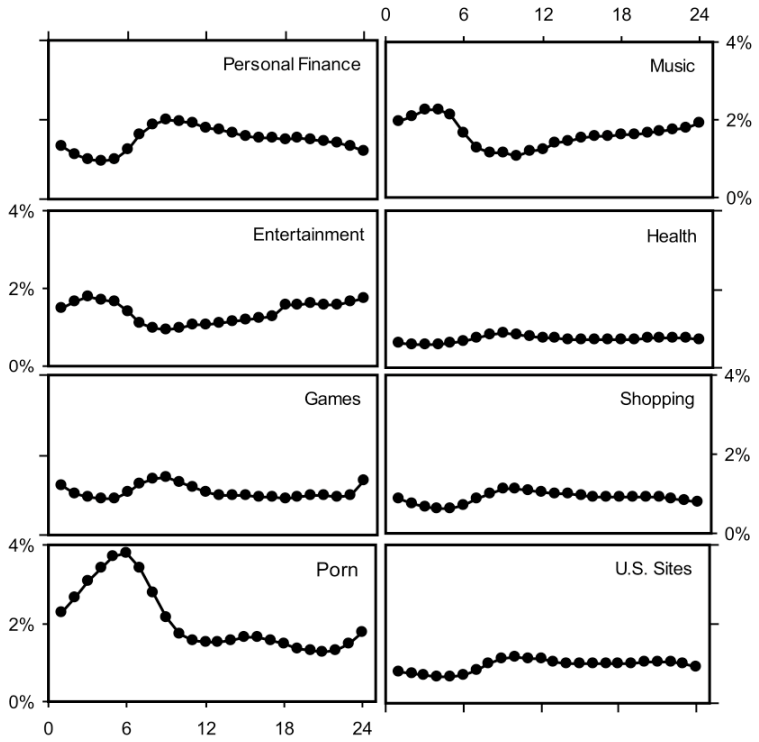


Fig. 2.8: Percentage of the total query stream covered by selected categories over hours in a day [34].

Although evidently some categories change more than others during the day, the comparison of the relative level of popularity shift is difficult due to the differences in scale of each of their percentages of the query stream. To overcome this issue, Beitzel *et al.* [34] compute the Kullback-Leibler (KL) divergence [128] (Equation 2.1) 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.

$$D\left(p\left(q|t\right)\|p\left(q|c,t\right)\right)=\sum_{q}p\left(q|t\right)\log\frac{p\left(q|t\right)}{p\left(q|c,t\right)} \tag{2.1}$$

Using the KL-divergence, it is possible to measure a sort of "*most surprising*" category for a particular time of 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. The histogram in Figure 2.9 shows the result of the KL-divergence computation over the same categories as Figure 2.8.

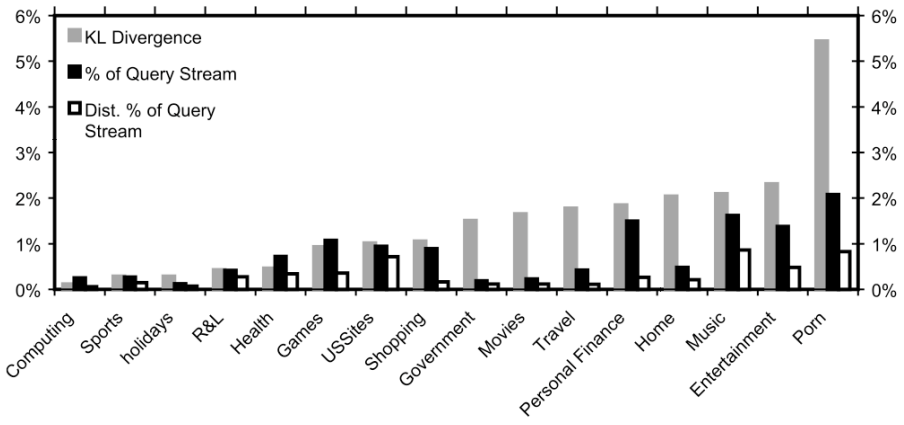


Fig. 2.9: Average percentage of query stream coverage & KL-divergence for each category over hours in a day. [34].

A more recent paper shows similar results on a MSN web search engine query log [241]. Results are not detailed with respect to topics, as in the case of the previous paper, yet they do not disagree with the overall results shown in [34].

Surprisingly, this analysis revealed that the top three categories in terms of popularity are pornography, entertainment, and music. Furthermore, it is worth being noticed that the KL-divergence is not directly correlated with the number of queries placed in each category. Also shown in Figure 2.9 is the average percentage of the entire query volume and distinct queries that match each category. Although the categories that cover the largest portions of the query stream also have the most relative popularity fluctuation, this correlation does not continue throughout all categories. Beitzel *et al.* [34] reach the same conclusion by thoroughly discussing a more through analysis on weekly and monthly basis.

2.3 Summary

In this chapter we presented an overview of the papers presenting statistics computed over different search engine query logs sampled over different periods of time. Some of the conclusions that can be drawn are common to all of the logs considered:

- Queries contains very few terms, on average ranging between 2 and 3 terms. This means that devising good results for a query is a very difficult task given that this very low number of terms often contains also ambiguous terms.
- The distribution of query popularity follows a power law. The most popular queries account for a very small fraction of the total number of unique queries. This phenomenon, also knows as *the Long Tail* [9], is pretty well known today and seems to arise whenever we deal with social and economical aspects of the new (internet) economy.
- Two conflicting claims have been presented. Following Spink *et al.* [209] it seems that X-rated query popularity is declining. Beitzel *et al.* [35], instead, claim that XXX queries are more *surprising* than others on certain time periods. On the other hand, non-XXX

queries do not show any particular peaks in submission frequency. This is the reason why they define XXX queries more frequent than others.

3

User Interactions

The previous chapter has shown how users query a search engine. It has been shown how users build queries and how queries repeat during time. This chapter is devoted to the study of how users interact with search engine Systems. What happen when a user has submitted a query and results are shown? For instance, how can be decided if a query has been correctly answered, if a user is satisfied by the search results? Another interesting aspect to investigate is how people change queries if those have not produced satisfying results. In fact, in search engines it is important to support such an activity also in addition to every efforts that could be made to enhance the search engine precision.

Back in 1994 when Yahoo! was first founded the two Stanford's grad students Jerry Yang, and David Filo were looking for a way to organize their bookmarks. They wanted to make people's life easier when they looked for some URLs they knew about. Actually, due to the overwhelming quantity of URLs present in the web nowadays, maintaining such a list manually (as it was done by Yahoo!'s founders back then) is unacceptable. Still, people want to find the a site's URL as fast as they can. For instance, it has been shown in the previous section that one of the most frequent queries in the public AltaVista log was "Yahoo". That means the people was looking for the Yahoo's URL.

Studies investigate the goals users have when using a web search engine. As it has been shown in the previous section, *web* IR and "*traditional*" IR users are very different. Usually they tend to type less, but still they want highly precise results.

In one of the first paper devoted to discovery user intent behind queries Andrei Broder [48] studies the goal a user wants to reach when submitting a query to a search engine. Following Broder's formulation a query can be either a *Navigational query* – where the immediate intent is to reach a particular site (e.g. *Greyhound Bus*, *american airlines home*, or *Don Knuth*); 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 queries* – where the intent is to perform some web-mediated activity (e.g. *online music*, or *online flower delivery service*).

Table 3.1 shows the result of a survey presented to Altavista users to try to determine their intent.

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

Table 3.1: Query classification on the basis of user survey. Adapted from [48].

The results shown in the table have been obtained by means of a series of questions presented to users through a "pop-up" windows 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 3.1 corresponds to a *manual* analysis of query entries. They firstly selected at random a set of $1,000$ queries and removed both non-English queries, and sexually oriented queries. From the remaining set the first 400 queries were inspected. Queries that were neither transactional, nor navigational, were assumed to be informational in intent.

Results from both the survey, and manual inspection confirmed what we were arguing in the previous chapter: 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 web through a 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.

Queries themselves are not always enough to determine users intent. Furthermore, one of the key objectives of a search engine is to evaluate the quality of their results. Implicit measures that ares 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.

3.1 Search Sessions

A series of queries can be part of a single, information seeking activity. There are some studies on the effect of request chains on the search engine side. The main goal of this kind of analysis is showing how users interact with the search engine and how they modify queries depending on results obtained by the search system and also how users use multitasking and task switching on search sessions[1] [210, 23].

The first interesting thing to observe is how users interact with the search engine from a page request point of view. As shown in many studies users rarely visit result pages beyond the first one.

[1] Roughly, a search session is a succession of queries submitted by the same user looking for an information.

In [213], the Excite query log is analyzed and it has been shown that about the 78% of users do not go beyond the first page of results. Different query logs are analyzed yielding to similar results by Lempel and Moran [132], and Fagni *et al.* [76]. Table 3.2 shows the results as reported by Fagni *et al.* [76] for three different query logs. In all the three cases, the probability that a user will go after the fifth page of results is under 0.02.

Query Log	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Excite	77.59	8.92	3.98	2.37	1.54	1.09	0.78	0.60	0.45	0.37
Tiscali	83.20	5.50	2.86	1.74	1.23	0.74	0.54	0.41	0.32	0.26
Altavista	64.76	10.25	5.68	3.41	2.54	1.83	1.42	1.16	0.94	0.88

Table 3.2: Percentage of Queries in the Logs as a Function of the Index of the Page Requested [76].

The figures shown above seem to persist throughout all the studies presented so far.

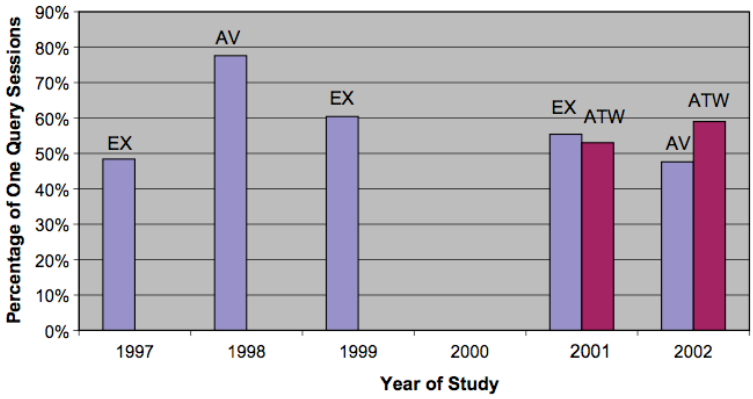


Fig. 3.1: Percentage of single query sessions. From [110].

Jansen and Spink [110] show, Figure 3.1, the percentage of single-query sessions in different query logs. In US web search engines[2], it does not appear that the complexity of interactions is increasing as indicated by longer sessions (i.e., users submitting more web queries). In 2002, approximately 47% of searchers on AltaVista submitted only one query, down from 77% in 1998.

A deeper analysis is conducted by Fagni *et al.* [76] where the probability of clicking the "*Next*" button of the search engine result page is estimated

Figure 3.2 shows that the probability of clicking the "*Next*" button increases as the number of result page increases. This may suggest that users issuing informational queries usually go through a higher number of pages. In particular, half the users on page two go to page three, and around 60-70% of users on page three go to page four.

During a search session a user often try to refine (or slightly modify) queries in order to get to the result he wants. This behavior is studied by Lau and Horvitz [130] by categorizing queries

[2] ATW – AlltheWeb.com, AV – AltaVista, EX – Excite

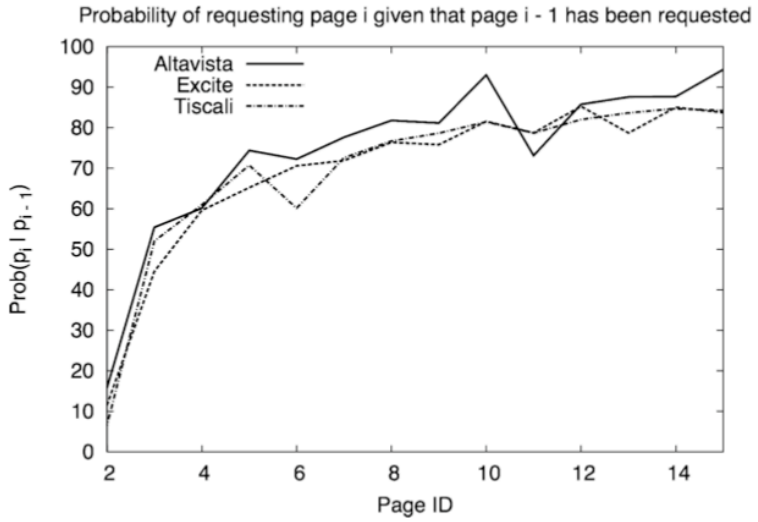


Fig. 3.2: Probability of pushing the next button for three different query logs. From [76].

according to seven categories. *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.

Figure 3.3, shows how queries are categorized within the Excite query log. As it is evident, in the majority of the cases most actions were either new queries or requests for additional information. Even though, a large percentage of users (around 30%) were issuing a modification (either a refinement, or a specification, or a reformulation) of a previously submitted query.

Previous results can be seen as a quantitative analysis of how users interact with the search system. A different point of view is represented by the analysis of *Multitasking* and *Task Switching* in query sessions. Multitasking sessions are those of users seeking information on multiple topics at the same time. Studies recently presented, show that users have an inclination to carry on multitasking queries. For instance, Ozmutlu *et al.* [157] show that in the 11.4% of the cases users are pursuing multitasking sessions. This percentage raises up 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 that raises to 3.2 when considering only multi-topic sessions.

An interesting event to detect is the query re-submission, or information *re-retrieval* [216]. The behavior analyzed with this kind of technique is how often users search for the same information he searched before. It is a quite common trend, nowadays, that users search instead of bookmarking. A

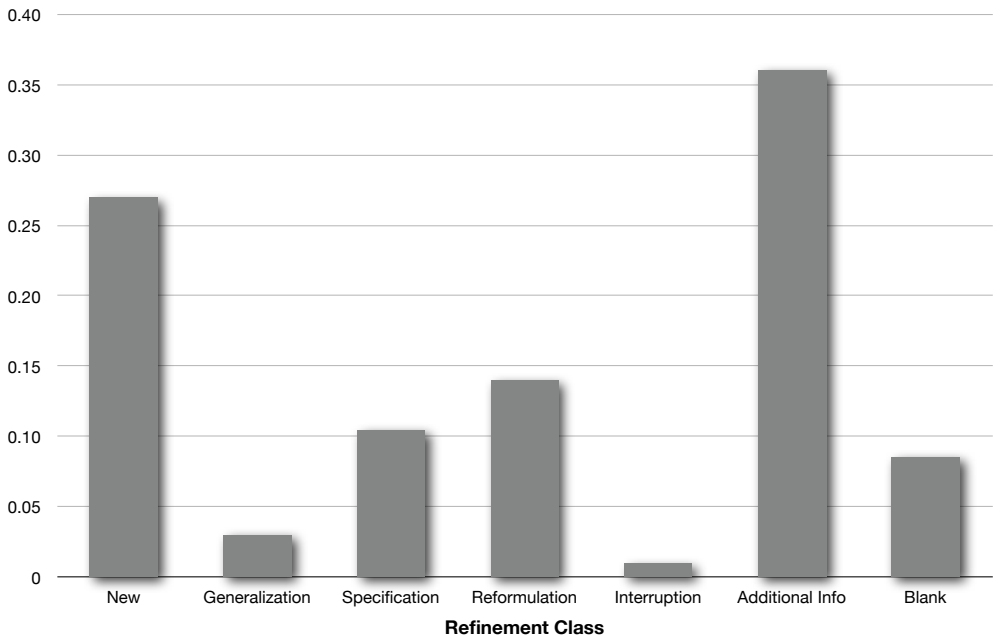


Fig. 3.3: Breakdown of the 4,960 queries analyzed in [130] into the different query modification categories defined.

possible example is a user that reaches the home page of a conference by issuing a query to a search engine. The user may be interested in looking for news about that conference again in two months (for instance to read the call for paper, to check submission deadline, to check paper format, to check the list of accepted papers, to register and book hotels).

All queries: 13,060 queries (100%)	**Overlapping Click Queries** – 5072 queries (39%)			No Common Clicks 7988 (61%)
	Equal Click Queries – 3777 (29%)		Some Common Clicks 1295 (10%)	
	Single Identical Click 3737 (29%)	Multiple Identical Clicks 40 (< 1%)		
Equal Query Queries 4256 (33%)	**Navigational Queries** 3100 (24%)	36 (< 1%)	635 (5%)	485 (4%)
Different Query 8804 (67%)	637 (5%)	4 (< 1%)	660 (5%)	7503 (57%)

Fig. 3.4: Percentage of the different query types. From [216].

Figure 3.4 shows the results of an analysis conducted on a small sample taken from search traces of queries issued to the Yahoo! search engine over a period of an entire year (August 1, 2004 to July 31, 2005) by 114 users. The average trace was 97 days long and search traces were considered for inclusion only if they included queries issued during at least four of the last ten days of the sample period.

The study was thorough and complete, queries for re-retrieval intentions were considered not only for their syntactic form (equal query strings) but also for the set of clicked result URLs. From the total 5,072 queries, 39% of the queries had some overlapping in the clicked set of URLs, of them 24% had equal query string, and equal clicked URL. Therefore, repeat searches and repeat clicks are very common. Almost contemporary to Teevan *et al.* [216] Sanderson and Dumais [187] evaluate the re-finding behavior in web search engines. The dataset they used 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. The first noticeable thing they discovered, 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 greater than the one presented by the previous study by Teevan *et al.* [216] (and also different from another previous result, again, of Teevan *et al.* [215]). This could be done by the particular search engine, and the period of time covered by queries in the log. The conclusion, anyway, 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 be think of 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.

3.2 Social Networks from Query-Click relations

Queries recorded in query logs can be used to build social networks. Some recent studies, have shown some interesting insights and views on such data. Differently from what have been done in the analysis of multitasking in search sessions, Baeza-Yates and Tiberi [19] and Baeza-Yates [21] consider queries as a whole, and study how queries and clicks combined can help in determining relations between queries. They base their analyses on a query log consisting of a not very large number of queries, around fifty million queries, having in mind the objective of capturing some semantic meaning behind user actions (i.e. clicks).

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 very high-dimensional space where each dimension corresponds to a unique URL u that have been 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 – i.e. $u_1, u_2, \ldots, u_5$, 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 3.1. In the formula D is the number of dimensions, i.e. the number of distinct clicked URLs, of the space.

$$W(e) = \frac{q \cdot q'}{|q| \cdot |q'|} = \frac{\sum\limits_{i \leq D} q_i \cdot q'_i}{\sqrt{\sum\limits_{i \leq D} q_i^2} \sqrt{\sum\limits_{i \leq D} {q'_i}^2}} \tag{3.1}$$

Graph edges can be computed/discovered in a fast way by using a *sort-based* algorithm on the list

of URLs clicked by each query. Weights, instead, are simply computable by a linear (in the worst-case) algorithm. Furthermore, for the sake of the analysis conducted, the log have been pruned by removing queries not resulting in any click. After the graph have been built, to speed-up the analysis phase, nodes corresponding to queries with very few clicks and edges with small weights are filtered out. Edges are then classified according to three different type of relations:

- *Identical cover* (a.k.a. *red edges* or *type I*). The set of URLs clicked by the two queries at both ends are identical. This is an indirect edge inferring that the two queries are semantically equivalent.
- *Strict complete cover* (a.k.a. *green edges* or *type II*). The set of URLs clicked for the query at one end is strictly included in the set of URLs of the query at the other end. The edge is direct and semantically speaking the first query is more specific than the second one.
- *Partial cover* (*black edges* or *type III*). None of the previous two conditions hold. No semantical information can be extracted, one can argue, for instance, that the query is multi-topical.

Query	**Sim**	**Type of Equivalence**
tcfu ↔ teachers federal credit union	1.0	acronym
fhb ↔ first hawaiian bank	1.0	acronym
wtvf ↔ new channel 5	1.0	synonyms (Nashville TV channel)
ccap ↔ wcca	1.0	synonyms (Wisconsin court Internet access)
free hispanic chat↔ latinopeoplemeet	1.0	synonym for domain name
lj ↔ www.livejournal.com	1.0	acronym for URL
babel fish ↔ altavista babel fish	1.0	synonyms
aj ↔ askgeeves	1.0	synonyms with misspell
yahoo for kids ↔ yahooligains	0.9	synonym for misspelled domain name
unit conversion ↔ online conversion	0.85	synonym
merriam↔m-w.com	0.84	name for domain name
yahoo directions↔maps.yahoo.com	0.48	synonym for URL

Table 3.3: Equivalent queries

how to learn guitar → online guitar lessons → berklee college of music
latest nokia mobiles → mobile phones
toyota auto parts → wholesale car parts → used auto parts → auto parts
wire window boxes → window box → decorative iron → wrought iron fence
www.mysiemens.com → siemens phone

Table 3.4: Example of query refinement (→ means from more to less specific)

In the above two Tables (Table 3.3 and Table 3.4), an example of such a classification from [19] is shown. Looking at Table 3.3 it is possible to notice how the technique is effective in discovering relations not easily found in other ways. In particular the aj synonym for the askjeeves search engine, or the synonym for the free hispanic chat and the latinopeoplemeet website. Also results shown in Table 3.4 are worth of being looked at. The first row of the Table, in particular, shows

a possible path to help people seeking for guitar lessons to find a college where guitar is taught to play (how to learn guitar → online guitar lessons → berklee college of music).

Another interesting results of Baeza-Yates and Tiberi [19] is a graph mining methodology able to automatically classify query relations using a novel metric exploiting the Open Directory Project for computing similarity between queries. Due to lack of space, we do not enter into details of how the similarity was computed. Authors conclude the paper by observing that the query log was actually small, if compared to real, multi-day, query logs, yet results were really nice and encouraging. In fact, even considering that the method is not 100% precise, the number of queries submitted every day is so high that it would allow the discovery of more than 300 million potentially interesting relations per day!

3.3 Summary

This chapter showed how query log data is used to detect search engine user behavior. The most noticeable results are:

- Users, in the vast majority of the cases ($\sim 78\%$) look at the first page of results only. In case they go through the second result page, the likelihood that they move on the third, the fourth, and so on, is very high.
- Queries can be classified in *Informational*, *Navigational*, and *Transactional*. Informational queries are those submitted by users looking for information on a particular topic (e.g. *San Francisco* or *normocytic anemia*). Navigational queries are those submitted by users looking, mostly, for the URL of a particular page they are looking for (e.g. *Greyhound Bus*, *american airlines home*, or *Don Knuth*). Transactional queries are those submitted by users looking for websites enable the buying of goods on the Internet (e.g. *online music*, or *online flower delivery service*). Queries are almost evenly distributed on the three categories.
- Users re-submit the same queries over and over, for this reason search engine designers should be aware of this and think of solutions to take advantage of this phenomenon
- Query sessions can be used to devise refinement (i.e. *generalization* or *specialization*) of queries. An analysis of a social network formed over queries out of a search engine query log, for instance, can help in devising surprising relations. A remarkable example is the query generalization path '*how to learn guitar → online guitar lessons → berklee college of music*' that relates the query "*how to learn guitar*" with a popular music college.

4

Enhancing Effectiveness of Search Systems

Previously submitted queries represent a very important mean for enhancing effectiveness of search systems. As already said, query logs keep track of information regarding interaction between users and the search engine. Sessions, i.e. the 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. All in all, 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.

Differently from what is presented in the chapter about enhancing efficiency, all the techniques presented in this part impact on the user experience starting from the interface presented by the search engine. Just to set the ground, and let our discussion be not only abstract, consider, the screenshot shown in Figure 4.1. For the query "Britney Spears" the search engine has prepared a

Fig. 4.1: A view on Yahoo! query suggestion system in action. In this screenshot it is shown how queries related to "Britney Spears" are proposed to users by the search engine. It is very likely that suggestions are obtained by mining query logs in order to infer query relations.

list of suggestions that most likely derives from an analysis of previously and frequently submitted queries (for example, "britney spears blackout" is a typical example of "time-dependent" query suggestion.)

Next in this part, we shall show that the analysis of past queries presents many pitfalls. In particular, it has been shown that users clicks are biased towards results that are ranked high in the result list [118]. This observation is at the basis of a number of studies aimed at "un-biasing" clicks in order to better understand users' goals.

Furthermore, remember also from the introduction we have stated that data collection can be done at different levels. One is usually more familiar with the server side logging. There exist, anyway, techniques to collect data client side. Toolbars, for instance, are an example of such a technique.

In the rest of this chapter, we analyze the possibilities offered by web mining techniques on query logs for the enhancement of search effectiveness by reviewing the current state of the art in this sector and by presenting the most relevant and useful results.

We start by providing some historical notes that helps readers in understanding how users' feedback has been used before the advent of search engines. In particular we will show some preliminary studies motivating the use of query logs for enhancing effectiveness of search.

The first technique we present is "*Query Expansion*". Expanding a query consists of adding terms to the original queries to increase the precision of the engine on the top-k results presented. It is a technique that has been studied, originally, out of the context of query log analysis. The inclusion of query log information has boosted considerably the effectiveness of the technique.

Relative to query expansion, "*Query Suggestion*" is a technique that is shown in this chapter. Suggesting queries is mainly a way to provide not experienced users with a list of queries that have been proven to be effective for expert users.

Instead of suggesting queries to users, search engine may act differently depending on the user itself. For instance a mathematician issuing the query "game theory" is more interested in knowing the mathematical foundations of the field. An economist issuing the same query, "game theory", is likely to be more interested in knowing about applications to economics of game theory. "*Personalization*" is the technique that adapt results to a given user. Using query logs to personalize results

has been proven to be effective and the chapter shows some interesting results. Personalization is concerned with adapting query results to a particular user. Similarly, one can use information about past user interactions to statically weight web pages. As PageRank [47] computes a static importance score using only structural information, techniques presented in this part "*Learn to Rank*" a page depending on the clicks it receives when returned as a result.

Finally, we present techniques to "*Spell-check and Correct*" queries users input.

4.1 Historical Notes and Preliminaries

Worth to point out is that the idea of analyzing user behavior is not new. This is in fact the rationale behind the well-known concept of *relevance feedback* is that a user might not know in advance what he was looking for, but for sure he knows as soon as he sees it. In fact, exploiting user's feedback has been proven to be en effective technique for enhancing IR systems' precision [185]. Probably the most important of this kind of algorithm is Rocchio's formula [183], which was developed as part of the the SMART project. Basically, it is a three staged technique. The first stage consists of finding a given number of documents considered relevant for a given query. The second stage consists of letting users interactively select what they retain to be the most relevant documents, and then it goes on with the third stage consisting in re-ranking all the documents according to a sort of Nearest Neighbor criterion. This last stage is performed by modifying the query according to Rocchio's formula given in Equation4.1

$$q_m = \alpha q_0 + \beta \frac{1}{|D_r|} \sum_{d_j \in D_r} d_j - \gamma \frac{1}{|D_{nr}|} \sum_{d_j \in D_{nr}} d_j \tag{4.1}$$

In the formula: q_m is the modified query; q_o is the original query; D_r, D_{nr} are the set of relevant and non-relevant documents respectively; α, β, γ are real-valued parameters used to weight the contribution of each element in the formula. Basically, the new query is obtained by the old one by reducing the weights of terms leading to the retrieval of irrelevant documents, and by increasing the weights of terms leading to the retrieval of relevant documents. As it is evident from the formula, there is the need for identify the relevant documents. In small scale IR systems an iterative process was employed where users were asked to mark relevant documents. In the case of web search engines, the relevance feedback information is obtained by the analysis of past queries and their relative clicked results. Getting back to web search engines. Can a Rocchio-like approach be used?

In a nice study performed by Joachims and Radlinski [118] it appears evident that "*the top position lends credibility to a result, strongly influencing user behavior beyond the information contained in the abstract*" (i.e. the snippet). They registered the number of clicks a given position obtained in two different conditions: *normal* and *swapped* rank. In the swapped rank setting, the position of the top ranked result was swapped with the result occupying originally the second position. Results are shown in Figure 4.2 and shows that there is only a slight dropping in the percentage of clicks the first result obtains, whereas the number of clicks the second result is, more or less, stable.

More recently, Craswell *et al.* [66] have presented a more thorough elaboration bias due to presentation order. They present a model of click probability trying to capture the dependency on position. As Joachims and Radlinski [118] they present results from a user study. Differently from them, though, the scale of the experiment is larger. They gather their data, in fact, from results

clicked on answers returned by a major search engine.

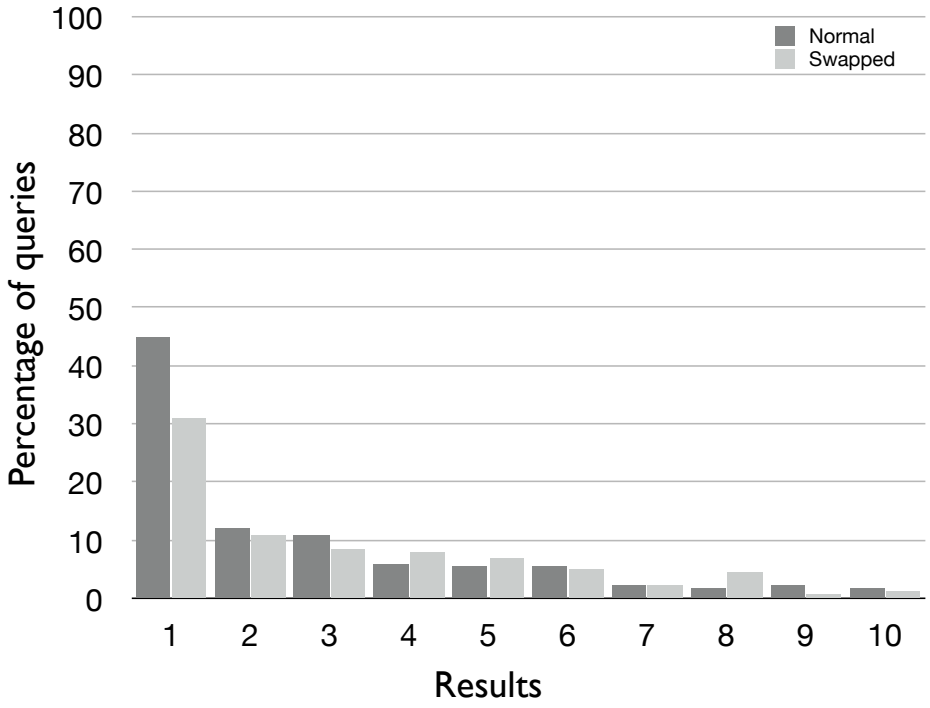


Fig. 4.2: Percentage of queries where a user clicked the result presented at a given rank, both in the normal and swapped conditions [118].

As said in the introduction, one of the main challenges in doing research with query logs is that query logs, themselves, are very difficult to obtain. In the case of effectiveness enhancing methods another big issue is also the quest for proving that a techniques is, actually, effective. The lack of datasets and well-defined metrics makes the discussion more faith-oriented than scientific-oriented. Furthermore, most of the techniques we review are either tested on a small group of (in most cases) homogeneous people, or metrics are tested on some sort of human-annotated testbeds. For this reason, we put more focus on the description of the techniques more than on the showing of their effectiveness. Indeed, comparing techniques is also, in most cases, impossible due to the very diversity of the datasets used.

Furthermore, query mining for effectiveness enhancement does not come for free. It brings with it some important issues that need to be considered. One of the most important and urgent is *privacy*, as already said in the introduction, is a very big concern for search engines. They, in fact may risk to raise a lot of arguments by people whose searches have been mined to build a profile. Therefore, unless handled in the strictest confidence possible, many people could get nervous for having the feeling of being *spied*.

Furthermore, profile-based (i.e. context-based) search is computationally expensive. If we compute results depending on who is submitting a query, many efficiency enhancement techniques (like caching, collection selection, and so on, see the next chapter for an analysis of these techniques)

soon become not exploitable anymore.

As a final remarkable point, all the techniques that are presented in the following can be used at different levels of the chain user-"search engine". Queries, in fact, can be stored server-side (the search engine, in this case, is the server) to build knowledge over global statistics. Historical information can be stored to client-side (user's browser, in this case, is the client) to build knowledge *only* on the basis of the local information available on the user's desktop. An intermediate solution is represented by the use of proxies storing information about a group of (possible) homogenous users. In all the cases, the same techniques can be applied to enhance the search experience of users.

4.2 Query Expansion

The statistics presented in "The Nature of Queries" chapter showed that web queries are very likely to be short, poorly built, and sometimes mistyped. For these reasons, the recalling power of queries is too much strong resulting in overwhelming the user with a lot of (sometimes) useless results. One of the main means with which a search engine precision might be enhanced are *query expansion* and *query suggestion*.

To confirm that these two mechanisms are rather important also in the real-world, it is worth noticing that quite recently web search engines have started to insert within their results both suggestions, and refinements to queries submitted by users. For instance, ASK within its result page inserts two boxes: "Narrow your search" (query expansion), and "Expand your search" (query suggestion), both containing related, and potentially useful, queries.

The first question we want to answer is: can query expansion be of help? The main advantage of query expansion is that users tune interactively the expansions and that when based on query log analysis, expansion is more effective due to the fact that queries are expanded with terms that previous users have specified to improve the query. The main drawback is that the search (and also the interactive search process) can take slightly longer.

It has been highlighted by Cui *et al.* [69] that queries and documents are rather poorly correlated. Using a two-month query logs (about 22 GB) from the Encarta search engine (http://encarta.msn.com), and 41,942 documents from the Encarta website, they measure the gap between the *document space* (all the terms contained in documents) and the *query space* (all the terms contained in queries). For each document in the document space they construct a corresponding *virtual document* in the query space by collecting all queries for which the document has been *clicked* on. A virtual document is 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 in turn measured by computing the cosine similarity between the two corresponding vectors. Actually, since many terms in document vectors appeared with zero weights in its corresponding query vector, to obtain a fairer measure, they only used the most important words in the document vectors as resulting from a simple $tf \times idf$ term ranking schema. Results confirm what expected: in most cases, the similarity values of term usages between user queries and documents are between 0.1 and 0.4 with only a small percentage of documents having similarity above 0.8. The average similarity value across the whole document collection is 0.28, which means the average internal angle between the query vector and the document vector is 73.68 degrees confirming that there is a quite large gap between the query space and the document space. Expanding a query might be of great help to bridge this gap as much as possible.

Query expansion is a technique dating back to seventies. Actually, one of the first works making explicit use of past queries to improve the effectiveness of query expansion techniques in traditional IR systems is Fitzpatrick and Dent [79]. It was proposed just a few years before the actual boom of search engines. The method, called *past-query feedback*, tests its effectiveness against the TREC-5 dataset. It uses a query DB made upon a combination of 50 ad hoc TREC-5 queries, 50 ad hoc TREC-3 queries, and 10 ad hoc TREC-4 queries. Basically, they build off-line an *affinity pool* made up of documents retrieved by similar past queries. When a query is submitted it is firstly checked against the affinity pool (which represent a sort of "high-quality" document repository for the purpose of expansion) and from the resulting top scoring documents, a set of "important" terms is automatically extracted to enrich the past query. Term relevance (i.e. importance) is computed using a straightforward $tf \times idf$ scoring function. Past-queries feedback showed an improvement of 38.3% in mean average precision if compared to the non query expansion case. We do not enter into the details of this method since it does not refer only to the case of web search engines. Besides, the method shows the *in-embryo* idea underlying most of the methods proposed thus far in the literature.

One of the first works exploiting past usage information to expand web queries is the one presented by Cui *et al.* [69]. Their method works by exploiting correlations among terms in *clicked* documents and user queries. Indeed, the exploitation of click-through data is due to the general assumption that usually is made in these kind of studies: *clicked documents are relevant to a query.* The method starts by extracting a set of *query sessions* from the query log. A query session, for a given user, consists of a query and a list of clicked results. For example the query session "Britney Spears"—4,982—2,212—8,412 represents a user submitting the query "Britney Spears" and successively clicking on documents 4,982, 2,212, and 8,412 respectively. For each one of the clicked documents, in each query session, a list of terms is extracted. The set of all terms contained within each clicked documents makes up the *Document Terms* set. The set of all the terms contained within all the queries, instead, forms 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 is created between t_d, and t_q 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 then weighted by 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, i.e. $P(t_d|t_q)$. By an algebra argument[1], this probability can be computed as:

$$P(t_d|t_q) = \sum_{D_i \in S_q} P(t_d|D_i) \frac{\text{freq}(t_q, D_i)}{\text{freq}(t_q)}$$

where:

- S_q is the set of clicked documents for queries containing the term t_q.
- $P(t_d|D_i)$ is the normalized weight of the term t_d in the document D_i divided by the maximum value of term weights in the same document D_i.
- $\text{freq}(t_q, D_i)$ is the number of the query sessions in which the query word t_q and the document D_i appear together.
- $\text{freq}(t_q)$ is the number of queries containing term t_q.

[1] For more information on the details of this formula derivation, see Cui *et al.* [69]

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 formula of the cohesion weight (CoWeight) is given by:

$$\text{CoWeight}\,(Q, t_d) = \log\left(\prod_{t_q \in Q} P\,(t_d|t_q) + 1\right) \tag{4.2}$$

It is worth remarking that we are assuming that terms within a query are independent (this justifies the productory in the formula). The cohesion weight is used to build a list of weighted candidate expansion terms. Finally, the top-k ranked terms (those with the highest weights) are actually selected as expansion terms for Q.

Pseudo-Relevance Feedback [234] is one of the most famous techniques used to expand queries. Some techniques build upon Pseudo-Relevance Feedback to provide better expansions.

In the previously presented paper, the query log used is from the Encarta search engine (http://encarta.msn.com), and the relative 41,942 documents are from the Encarta website. To assess the improvement in effectiveness, a total of 30 queries are extracted randomly from a combination of the Encarta query logs, from the TREC query set, and from a small subset of hand-crafted queries. The query-log-based method was compared against the baseline of not using query expansion, and the *local context analysis* method (that does not make use of query log information) proposed by Xu and Croft [234]. For the local context method the number of expanded terms for each query was set to 30, whereas for the log-based it was set to 40. The mean average precision for the baseline on the test collection was 17.46%, whereas the local context method scored a mean average precision of 22.04%, an improvement of around the 26.24% on the baseline. The log-based method scored a mean average precision of 30.63% corresponding to an improvement of around 75.42% on the baseline. The number of terms used in the expansion has an impact on the results, actually another experiments showed that when more than 60 terms are considered in the expansion the mean average precision of the log-based method starts to decrease.

The technique of Xu and Croft [234] is quite obsolete if compared to today's search engine technology. In particular, expanding a query to be longer than 30 terms is definitely not viable due to the overhead it causes in the corresponding query processing phase.

Indeed, Query/Documents terms co-occurrence is just a possible way of associating document terms to query terms.

Billerbeck *et al.* [42] use the concept of *Query Association*. User queries become *associated* with a document if they are "similar" that document. Queries, thus, enrich documents with an associated *Surrogate Document*. Surrogate documents are used as a source of terms for query expansion. Obviously not all of the queries concur to form query associations. First of all, whenever a query is submitted it is associated with the top K documents returned. 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, then, is computed using a standard scoring function. In the experiments performed by the authors they used the Okapi BM25 [119] scoring function. In case a document reached its full capacity of associated queries, the least similar query is replaced by a newly submitted one only in case it has a similarity higher than the least similar associated query. Depending on two parameters, the method has to be fine tuned with respect to the different values of K, and M. In a recent paper, Scholer *et al.* [189] empirically set $K = 5$, and $M = 3$.

Summaries, thus, tend to be small but composed of high quality surrogate documents.

Billerbeck *et al.* [42] use the Robertson and Walker's term selection value formula [181] (Equation 4.3) to select the e expanding terms.

$$\text{weight (t)} = \frac{1}{3} \log \frac{(\text{RetDocs}\,(t) + 0.5) \,/\, (f_t - \text{RetDocs}\,(t) + 0.5)}{(|T| - \text{RetDocs}\,(t) + 0.5) \,/\, (\text{NDocuments} - f_t - |T| + \text{RetDocs}\,(t) + 0.5)} \quad (4.3)$$

where:

- $\text{RetDocs}\,(t)$ is the number of returned documents containing term t.
- f_t is the number of documents in the collection containing t.
- NDocuments is the total number of documents in the collection.

How the above equations are used? First of all, we shall present a generalization of a query expansion algorithm:

(1) For a newly submitted query q, an initial set T of top ranked documents is built.
(2) From T, a list of e expanding terms is selected using a term selection value formula.
(3) A new query made up of the previous one with appended the top most scoring terms from the collection is submitted again to the search engine to obtain a new list of results.

In standard query expansion, steps (1), and (2) of the above generalized expansion algorithm are performed within the full-text of the entire collection. Billerbeck *et al.* [42] refer to this as the FULL-FULL method. Considering a surrogate documents in either step (1), or (2), or both we may have the following three combinations:

- FULL-ASSOC, where the original query ranks documents of the full text collection, and expansion terms are selected from the surrogate document. Therefore, step (1) of expansion is on the full text of documents in the collection, while step (2) is based on query associations.
- ASSOC-FULL, where surrogates have been built and retrieved instead of the full-text, but the query expansion is computed selecting highly scoring terms from the full-text.
- ASSOC-ASSOC, where both retrieval, and expansion are computed on surrogate documents.

In addition they also experiment with an expansion schema called QUERY-QUERY where steps (1), and (2) are performed directly on previously submitted queries, instead of considering query associations. The QUERY-QUERY schema captures the idea of expanding queries with terms used by past users in queries considered better specified (e.g. expanding the query *Ratatouille Movie* into *Ratatouille Movie Pixar*).

Experiments conducted on a TREC collection showed the superiority of the ASSOC-ASSOC schema that outperformed the classical FULL-FULL by 18-20%. In this setting, the value for K, and M were set to those empirically found by Scholer *et al.* [189] to perform better. It is worth noticing how the combination of both past queries and full-text is superior to using either one of the two independently.

Just to make the discussion more concrete, following an example of choice of terms for expansion. For the query *earthquake*, the FULL-FULL method expanded the query with the terms: "*earthquakes*

tectonics earthquake geology geological", whereas the ASSOC-ASSOC expanded earthquake with a more descriptive "*earthquakes earthquake recent nevada seismograph tectonic faults perpetual 1812 kobe magnitude california volcanic activity plates past motion seismological*".

The query expansion techniques shown so far, have been rarely applied by search engines. As it is evident from the discussion so far, they suffer, mainly, of scalability issues. In many cases the analysis done is very heavy and cannot scale very well. Another possibility is represented by query suggestion. In the next part we present these techniques that have been proven to be effective and scalable in search engines.

4.3 Query Suggestion

As it has been shown so far, in query expansion only a general view of queries/documents/snippets associations are taken into account. That is, if two users having different tastes submit the same query, this is expanded with the same terms. Therefore, it is very likely that one of the two results unsatisfied.

As opposed to query expansion, query suggestion is the technique consisting of exploiting information on past users' queries to propose a particular user with a list of queries related to the one (or the ones, considering past queries in the same session) submitted. Furthermore, the number of suggestions can be kept as short as we want, usually it ranges from two to five queries suggested. With query expansion, in fact, users can select the best similar query to refine their search, instead of having the query automatically, and uncontrollably, stuffed with a lot of different terms. Obviously, this only partially overcomes the issue above, because if a topic is still underrepresented it is very unlikely that it is suggested within a query suggestion list.

Roughly speaking, drawing suggestions from queries submitted in the past can be interpreted as a way of "*exploiting queries submitted by experts to help non-expert users* [17]." 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, as stated in [239], to find queries similar to another one consists of simply looking for those queries sharing terms regardless of every other possible feature. So, for instance, the query "*George Bush*" would be considered, to some extent, similar to the query "*George Michaels*" given that, both share the term *George*. Obviously, this simple example shows that the naïve approach might result misleading for users.

In literature there have been presented quite a lot of works on query recommendation. Ranging from selecting queries to be suggested from those appearing frequently in query sessions [80], to use clustering to devise similar queries on the basis of cluster membership [17, 18, 22], to use click-through data information to devise query similarity [242, 68].

Query sessions (see the User Action chapter for more information) can be a precious source of information for devising potentially related queries to be suggested to a user. The idea is that if a lot of previous users when issuing the query q_1 also issue query q_2 afterwards, query q_2 is suggested for query q_1. One choice for capturing this idea is association rule mining [7] and it is actually used to generate query suggestions by Fonseca *et al.* [80].

Basically, the setting for a typical associations-mining algorithm consists of a set D of itemsets (i.e. sets of items) A, each of which is drawn from a set I of all possible items. Each A is a member of the power set 2^I. The database considered is made up of *transactions*, each transaction is a set

of items. Given two non-overlapping itemsets A, and B, an association rule is a formula $A \Rightarrow B$ stating that the presence of A in a transaction implies the presence of B. The algorithm has two parameters: the minimum support value σ – i.e. the minimum number of transactions containing $A \cup B$, and the confidence γ – i.e. the minimum accepted probability of having B in a transaction given that A is contained within the same transaction.

Computing association rules in very large DBs can be computationally expensive. In fact the approach by Fonseca *et al.* [80] allows only rules of the form $q_i \Rightarrow q_j$. Therefore, the effort needed to compute the frequent itemset mining phase[2] is considerably reduced.

For each query q_i, all the rules $q_i \Rightarrow q_1, q_i \Rightarrow q_2, \ldots, q_i \Rightarrow q_m$ are extracted and sorted by confidence level. To experiment the technique it has been used a query log coming from a real Brazilian search engine and consisting of 2.312.586 queries. The mining process is carried out by setting the support $\sigma = 3$, and by extracting all the possible association rules. To assess the validity of their approach they conducted a survey among a group of five members of their laboratory by asking for the top 5 frequent queries whether the proposed suggestions were relevant or not. Results were encouraging, even if the assessment phase is not convincing enough. For example, for the top 95 frequently submitted queries the system is able to achieve a 90.5% precision measured as the number of suggestions retained relevant for the five laboratory members surveyed. Clearly, the population of assessors is biased (they are all members of the same lab), thus potentially another group of people might have found those results less meaningful.

The number of queries suggested is also important. Obviously, in this case it is not true that the more the better. Actually, increasing the number of queries causes a drop in the suggestion quality. The 90.5% figure in precision was measured for the case of five query suggested. Precision drops to 89.5% when ten queries are suggested, down to 81.4% when 20 queries are suggested.

Actually, there is a trade off not highlighted by authors. Although 90.5% precision for five queries corresponds to more than four queries relevant, 89.5% precision for 10 queries, instead, consists of around nine queries out of 10. The list of potentially interesting queries is thus richer in the case of more suggestion shown. On the other hand, users presented with a long list of queries might experience a "*swamping effect*" resulting in users simply ignoring suggestions. Therefore, fine-tuning the number of suggestions is of utmost importance for the success of the method. Indeed, the need for fine tuning the number of suggestions is on a per-query basis: for frequently submitted queries a long number of suggestion would be better, for rarely submitted ones the number of suggestion should be, very likely, kept inherently small.

As opposed to association rule mining, Zaïane and Strilets [239] use a *Query Memory* to store past queries and retrieve them according to the one submitted. In a sense, this method computes associations on-the-fly at query resolution time.

A Query Memory is a set of queries represented by six different features: (i) *BagTerms*: the bag-of-words representation of the query, i.e. the unordered set of terms contained within the query string; (ii) *Count*: the frequency with which the query has been submitted; (iii) *Ldate*: the timestamp recording the last time the query was submitted; (iv) *Fdate*: the timestamp recording the first time the query was submitted; (v) *QResults*: the query result list stored as records made up of *Rurl* – the URL, *Rtitle* – the title, and *Rsnippet* – bag-of-words representation of the snippet,

[2] In association rule mining the computation is made in two steps. The first step is the frequent itemset mining and consists of retrieving all the subsets of the DB's transactions exceeding the minimum support σ threshold.

of each result page; (vi) *Rdate*: the timestamp recording the date at which results were obtained (note that this timestamp not necessarily relates to Ldate and Fdate).

Let Q be the submitted query, Δ be the Query Memory, Q.terms be the bag-of-words representing the query terms, and Q.results be the array storing the results. In particular, Q.results $[i]$ is the i-th entry of the result set for Q.

By using this representation we have seven different methods for returning a set of queries similar to the submitted one.

(1) *Naïve query-based method*: returning queries having in common at least one term.
$\{q \in \Delta \text{ s.t. } q.\text{terms} \cap Q.\text{terms} \neq \emptyset\}$

(2) *Naïve simplified URL-based method*: returning queries having in common at least one URL in the result lists.
$\{q \in \Delta \text{ s.t. } q.\text{QResults.Rurl} \cap Q.\text{QResults.Rurl} \neq \emptyset\}$

(3) *Naïve URL-based method*: returning queries having in common a large fraction of URLs in the result list. In the formula below, θ_m, and θ_M are a minimum and maximum threshold (both real numbers ranging from 0 to 1) used to tune the intersection cardinality.
$\{q \in \Delta \text{ s.t. } \theta_m \leq \frac{|q.\text{QResults.Rurl} \cap Q.\text{QResults.Rurl}|}{|Q.\text{QResults.Rurl}|} \leq \theta_M\}$

(4) *Query-Title-based method*: returning queries where terms in their result titles are contained in the submitted query.
$\{q \in \Delta \text{ s.t. } \exists i \; q.\text{QResults}\,[\text{i}]\,.\text{RTitle} \cap Q.\text{terms} \neq \emptyset \text{ and } q.\text{QResults}\,[i] \notin Q.\text{QResults}\}$

(5) *Query-Content-based method*: it is the same as 4 only considering snippet terms instead of title terms.
$\{q \in \Delta \text{ s.t. } \exists i \; q.\text{QResults}\,[\text{i}]\,.\text{RSnippet} \cap Q.\text{terms} \neq \emptyset \text{ and } q.\text{QResults}\,[i] \notin Q.\text{QResults}\}$

(6) *Common query title method*: returning queries whose results share title terms.
$\{q \in \Delta \text{ s.t. } \exists i, j \; q.\text{QResults}\,[\text{i}]\,.\text{RTitle} \cap Q.\text{QResults}\,[\text{j}]\,.\text{RTitle} \neq \emptyset\}$

(7) *Common query text method*: it is the sam as 6 only considering snippet terms instead of the title terms.
$\{q \in \Delta \text{ s.t. } \exists i, j \; q.\text{QResults}\,[\text{i}]\,.\text{RTitle} \cap Q.\text{QResults}\,[\text{j}]\,.\text{RTitle} \neq \emptyset\}$

Experiments in the paper were, as in the previous case, conducted on a user study. They collected more than half a million different queries from the Metacrawler search engine. From this set of queries they extracted and harvested the results of a subset of 70,000 queries (which constitutes the Query Memory). Submitting queries to the search engine and harvesting result is, indeed, very computationally and network intensive. More realistically, since the query recommender is usually on the search engine side, not only the query trace would be immediately available, but also the inverted indexes of the search engine would also be available avoiding the submission of queries for results harvesting.

The most interesting result claimed is that "*it was difficult to find a winner (i.e. the similarity measure with the best recommendation)*" [239]. Indeed, from a general perspective, the main reason for this to hold, could be the fact that the validation process is a very subjective one.

One of the most interesting observation made by Zaïane and Strilets [239] is on the scalability of their approach to a real-world setting. Actually the query memory is a DB for queries from which depends the effectiveness of a recommendation method. The richer the query memory, the better the suggestions computed. Yet, the way query records are stored is a crucial point and the paper does not discuss it as deep as it should. Searching for queries containing a given keyword,

or keywords, would require in real world systems an index *per-se*, making the methods paying a double index access for each submitted query. Obviously, these two accesses can be done in parallel using a distributed architecture.

Baeza-Yates *et al.* [17] use a clustering approach to query recommendation. The query recommender algorithm operates using a two tiered approach. An offline process clusters past queries using query text along with the text of clicked URLs. The online process follows a two-stage approach: (i) given an input query the most representative (i.e. similar) 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 (this is called *support* in their paper and should not be misinterpreted as support of association rules [7]). Interestingly this work as a lot of points in common to the one of Puppin and Silvestri [172] that has another (quite different) purpose. Enhancing the effectiveness of collection representation in collection selection functions. We shall talk about this topic in the Enhancing Efficiency of Search Systems chapter.

The offline query clustering algorithm operates over queries enriched by a selection of terms extracted from the documents pointed by the user clicked URLs. Clusters are, thus, computed by using an implementation of the k-means algorithm [107] contained in the CLUTO software package[3]. The similarity between queries is computed according to a vector-space approach. That is, each query q is represented as a vector $\vec{q}$ whose i-th component q_i is computed as

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

where:

- $\text{Clicks}\,(q,u)$ is the percentage of clicks URL u receives when answered in response to the query q.
- $\text{Tf}\,(t_i,u)$ is the number of occurrences of the term t in the document pointed to URL u.

The sum is computed by considering all the clicked URLs for the query q. The distance between two queries is computed by the cosine similarity of their relative vectors.

The k-means algorithms is sensitive to the value of parameter k, i.e. the number of computed clusters. To measure the optimal number of cluster an empirical evaluation measuring the *compactness* of clusters have been performed. A run of a k-means algorithm is executed with different values of k. Each clustering result is assigned to a function computing the total sum of the similarities between the vectors and the centroids of the clusters that are assigned to. For the considered dataset the number of query cluster has been set to $k = 600$. The query log (and the relative collection) they use, comes from the TodoCL search engine and contains 6,042 unique queries along with associated click-through information. 22,190 clicks are registered in the log spread over 18,527 different URLs.

The experimental evaluation of the algorithm is performed on ten different queries. Evaluation is done again by a user study. The evaluation assesses that presenting query suggestions ranked by support (that is the frequency with which the query occur in the log) yields to more precise and high quality suggestions.

[3] `http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview`

Recently, Jones *et al.* [121] have proposed a model for generating queries to be suggested based the concept of query rewriting. A query is rewritten into a new one by means of query or phrase substitutions (e.g. the query "*leopard installation*" can be rewritten into "*mac os x 10.5 installation*"). The rewriting process is based on a Log-Likelihood Ratio (LLR) measure to evaluate interdependencies between terms of queries. The metric tests the hypothesis that the probability of seeing a term q_2 in a query is the same wether or not term q_1 has been seen. Basically, the hypothesis $H_1 : P(q_2|q_1) = p = P(q_2|\neg q_1)$ is tested against the hypothesis $H_2 : P(q_2|q_1) = p_1 \neq p_2 = P(q_2|\neg q_1)$. The log likelihood-ratio is a statistical test for making a decision between two hypotheses based on the value of this ratio [145]. The log likelihood ratio is computed as

$$\text{LLR} = -2 \log \frac{L(H_1)}{L(H_2)}$$

and large values of the ratio means that the likelihood of the two words of being dependent is higher. From observations, query pairs with high log likelihood ratio are identified as *substitutables*. Furthermore given that the likelihood ratio, computed as $\lambda = \frac{L(H_1)}{L(H_2)}$, is distributed as a χ^2, a score of 3.84 for log likelihood ratio gives a 95% confidence in rejecting the null hypothesis (therefore, the relation between the two terms (or phrases) is statistically significant). Indeed a 95% confidence means 1 in 20 not correctly detected substitutable. For this reason in [121] they set the log likelihood ratio threshold to a high level of 100 thus making the method highly selective and precise. Indeed, since they are dealing with logs of millions of queries even with such a high threshold level the number of candidates found is acceptable.

A noteworthy point of the paper is the systemization of possible suggestion into four classes going from the most precise, down to the less precise class: *precise rewriting*, *approximate rewriting*, *possible rewriting*, and *clear mismatch*. *Precise rewriting* (class **1**) means that the query suggested has exactly the same semantics of the one to be replaced (e.g. *automotive insurance* is a precise rewriting of *automobile insurance*). In *approximate rewriting* (class **2**) the scope of the initial queries is narrowed or broadened (e.g. *ipod shuffle* is a more narrow of, but still related to *apple music player*). *Possible rewriting* (class **3**) is a still less precise query suggestion methodology: queries are either in some categorical relationship (e.g. *eye-glasses* and *contact lenses*), or describes a complementary object (e.g. *orlando bloom* vs. *johnny depp*). The last class (**4**), *clear mismatch*, is the less precise and contains query pairs where no relationships can be found among them.

Using these four classes we can identify the two general tasks of query suggestions, namely *Specific Rewriting* (or 1+2), and *Broad Rewriting* (or 1+2+3). The former consists in generating suggestions of the first and second class, the latter consists in generating suggestions of the first, second, and third class. Given these tasks, the problem of generating query suggestion can be reformulated into a classification problem.

In the original paper many classifier have been tested. Among the others, a linear classifier is used. Training is performed on a manually annotated set of substitutions extracted from queries of a query log. The extraction is made on the basis of the log likelihood ratio defined above, and the features considered are of three different types: *characteristics of original and substituted query in isolation* (e.g. length, character encoding, etc.); *syntactic substitution characteristics* (edit distance, number of tokens in common, etc.); *substitution statistics* (e.g. log likelihood ratio, $P(q_2|q_1)$, etc.)

We omit the details on how the classifier has been trained and tested and on how data has been prepared. We report, here, only the main results. Interested readers are encouraged to read the

whole paper by Jones *et al.* [121] to get the full picture of their technique.

The learned ranking function is given by

$$\begin{aligned} f(q_1, q_2) = 0.74 \quad &+ \quad 1.88 \, \text{editDist}(q_1, q_2) \\ &+ \quad 0.71 \, \text{wordDist}(q_1, q_2) \\ &+ \quad 0.36 \, \text{numSubst}(q_1, q_2) \end{aligned} \tag{4.4}$$

and it is computed for query pairs whose log likelihood ratio is above an empirical threshold set to 100 (i.e. LLR > 100).

One of the main advantage of the rewriting approach is that it is possible to experimentally asses the quality of suggestions without recurring to user surveys. For the log used in [121], the precision of the method has been measured to be 76% with a recall figure of 92%.

As in the case of query expansion, query suggestion might suffer of the same problem due to different tastes of users. Next paragraph addresses exactly this aspect and shows how information on past queries and more specifically on *who* submitted them can be used to create results tailored on a particular user category.

As a final note, it is worth mentioning a body of literature that recently has started to consider link recommendations instead of query suggestions to enhance the effectiveness of a web search engine. Basically, the idea is as follows: use click-through information to infer what users are looking for and instead of suggesting a possible query suitable for finding it, suggest immediately the site they were potentially looking for. Typical mechanisms for achieving this are quite similar to those shown above for query suggestion. The interested readers are encouraged to read [228, 227, 229, 41] as a starting point.

A slightly different problem, yet related with query recommendation is the problem of finding "Query Shortcuts" [31, 30, 29]. Authors define formally the *Search Shortcut Problem* (SSP) as a problem related with the recommendation of queries in search engines and the potential reductions obtained in the users session length. This new problem formulation allows a precise goal for query suggestion to be devised: recommend queries that allowed in the past users that used a similar search process, to successfully find the information they were looking for. The approach take so far for the approximation of the solution of the SSP is based on the adoption of collaborative filtering techniques. Authors point out to some limitations of traditional algorithms, mainly related with the sparsity of query log data, so future work should develop new algorithms specially designed for this new problem.

4.4 Personalized Query Results

When issuing a query from different places or in different moments, users may receives different results. Why? This is likely due to personalization of search results. Personalization consists of presenting different ranking depending on searcher tastes. For instance, if a mathematician issues the query "game theory" it is very likely that he will be returned with many results on theory of games and theoretical stuff. On the other hand, an economist would be rather more interested in how game theory has been (or can be) applied to real-world economy problems. How the search engine can be aware that the user is a mathematician rather than an economist? One possible answer is, again: mining query logs.

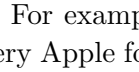
As stated by Pitkow *et al.* [165]: "*What's needed is a way to take into account that different*

people find different things relevant and that people's interests and knowledge change over time."

Personalization, consists of delivering query results ranked according to the particular tastes of a precise user (or class of users). Personalization, usually, is enabled by means of "*re-ranking*" search results according to a specific user's profile built automatically[4]. Obviously, personalization is not the "*panacea*" for search effectiveness: Teevan *et al.* [217] demonstrate that for queries which showed less variations among individuals re-ranking results according to a personalization function may be insufficient (or even dangerous).

One of the first work discussing personalization of search results is presented by Haveliwala [96] that shows how to modify the PageRank algorithm [47] to bias score calculation toward pages related to a given topical category. The work presented in the paper does not make use of query logs to compute personalization functions, yet it shows an interesting method to statically re-rank pages according to user's preferences. Basically, the personalization vector of the PageRank algorithm is set to weight more those pages belonging to the topical categories chosen. In light of the random surfer point of view, the personalized PageRank models a topically-biased random surfer that instead of jumping to page chosen uniformly at random jumps *only* to pages of the category he belongs to.

Liu *et al.* [138] categorize users with a set of relevant categories. The categorization function is automatically computed on the basis of the retrieval history of each user. The category set is fixed and is considered to be the one also used by the search engine to categorize web pages[5].

The two main concepts used are *Search Histories* for users, and *Users Profile*. *Search History* for users is kept by means of a tree structure. For example, Apple $\rightarrow$ Food&Cooking $\rightarrow$ 

represents a search history storing that the query Apple for this user belongs to the Food&Cooking category, in answer to this search, user clicked on results page1.html and page2.html. For each user, and for each query, more than one category may be associated. In practice, though, more than two categories have rarely been associated with the same query by a single user. *Users Profile* stores the set of categories hit by the corresponding user. Each category is associated with a set of weighted keywords that are considered important for the description of that category. For each user, Search History and User Profile are stored internally as a set of three matrices DT, DC, and M.

The m-by-n matrix DT stores the associations between the m *Clicked* documents or issued queries, and the n distinct terms appearing in those documents or queries. $DT[i, j]$ is greater than zero if term j appears in document/query i. The entry is filled-in by computing the normalized TF-IDF score.

The m-by-p matrix DC stores the associations between documents/queries, and the p possible categories. Each entry $DC[i, j]$ is either 1 or 0 wether document/query i is related to topic j or not.

The p-by-n matrix M is the user profile and is *learnt* by the previous two matrices DT, and DC by means of a machine learning algorithm. Each row is a vector representing a category in the term-space.

To give an example consider a user submitting two queries *frog*, and *screen*. Suppose for the query *leopard* the user is interested in a particular species of frog; suppose for the query *screen* the user is interested in TV screens. The first and fourth rows of the DT matrix (Table 4.3a) store the

[4] It is possible for search engines to collect explicitly this data, yet we are interested in presenting non-intrusive methods that automatically collect user's information and devise user's profiles.

[5] For example http://www.google.com/dirhp, or http://dir.yahoo.com/

Doc/Term	leopard	medow	grass	screen	tv
D1	1	0	0	0	0
D2	0.58	0.58	0	0	0
D3	1	0.7	0.5	0	0
D4	0	0	0	1	0
D5	1	0	0	0.6	0.4

(a) The matrix *DT* storing issued queries and clicked documents on rows and terms in columns.

Doc/Categ	NATURE	HI-TECH
D1	1	0
D2	1	0
D3	1	0
D4	0	1
D5	0	1

(b) The matrix *DC* storing issued queries and clicked documents on rows and categories in columns.

Categ/Term	leopard	medow	grass	screen	tv
NATURE	1	0.4	0.4	0	0
HI-TECH	0	0	0	1	0.4

(c) The matrix M with the learnt user profile: on the rows the Category on the columns the category keywords.

Fig. 4.3: The matrix representation of user search history and profile.

queries, in fact all the entries are set to 0 except for the terms contained within the queries that are, instead, set to 1. The remaining rows store representative terms of the clicked documents weighed by their TF-IDF scores. The DC matrix (Table 4.3b) stores the associations between documents and categories. In the example, the first three rows are for the Nature category, the remaining two for the Hi-Tech category. The matrix M (Figure 4.3c) stores on each row the relative weight of the terms for the given category.

In addition to the three matrices defined per each user, three additional *general* matrices are generated independently from a particular user by the *Open Directory Project*[6] (ODP) category hierarchy. The three matrices are DTg, DCg, and Mg (where g stands for general). The label associated with the first two levels categories are used as documents, while the third level labels are used to extract labels.

The matrix M can be "deduced" using either user profiles only (DT, and DC), or using the general profile only (DTm, and DCm), or using both profiles. The process of generating the matrix M can be viewed as a *multi-class text categorization* task, therefore one can adopt different techniques to learn M. In [138] authors use three classes of learners. *Linear Least Squares Fit* (LLSF)-based methods [236], *Rocchio*-based methods[183], and a *k Nearest Neighbor* (kNN)-based method.

LLSF-based methods computes a p-by-n category matrix M such that $DT \times M^T$ approximates DC with the least sum of square errors. To compute such M we first compute the Singular Value Decomposition (SVD) of DT obtaining three matrices U, Σ, V such that U, and V are orthogonal matrices, and Σ is a diagonal matrix. SVD is computed in order to be able to have $DT = U \times \Sigma \times V^T$. After the SVD $M = DC^T \times U \times \Sigma^{-1} \times V^T$. A variant of LLSF is *pseudo*-LLSFT. It consists of considering only the first k columns of U, and V of the SVD to reduce the effect of noisy entries. Therefore, $M = DC^T \times U_k \times \Sigma_k^{-1} \times V_d^T$. This method stems from another, and more popular, Latent Semantic Indexing (LSI) [87] method used with big success in many different IR applications.

Rocchio-based methods assign each cell $M[i, j]$ using the following equation

$$M[i, j] = \frac{1}{N_i} \sum_{k=1}^{m} DT[k, j] \cdot DC[k, i]$$

[6] `http://www.dmoz.org/`

where m is the number of documents in DT, N_i is the number of documents that are related to the i-th category. A very nice variant of this method is the *adaptive* Rocchio. In adaptive Rocchio, entries of the matrix M are updated as the data comes in. The formula of the basic Rocchio algorithm, thus, is computed according to

$$M^t[i,j] = \frac{N_i^{t-1}}{N_i^t} M^{t-1}[i,j] + \frac{1}{N_i^t} \sum_{k=1}^{m} DT[k,j] \cdot DC[k,i]$$

where M^t is the user profile at time t, N_i^t is the number of documents related to the i-th category that have been accumulated from time zero to time t.

The last method, kNN, does not compute the matrix M. Instead, it first finds the k most similar documents among all document vectors in DT using the Cosine metric. Secondly, among the k neighbors a set of documents S related to category c is extracted using DC, and the final similarity between a user query q and c is computed as the sum of the similarities between q and the documents in S. Therefore, the following formula is used to compute the similarity between a query q and a category c_j:

$$sim(q, c_j) = \sum_{d_i \in \text{kNN}} cos(q, d_i) \cdot DC[i,j]$$

where $cos(q, d_i)$ is the cosine similarity between q and the descriptor of document j.

Apart from the case of the kNN-based method shown above, similarities between a query vector q and a category vector c_j, rows of M, is computed by the Cosine metric [186]. Therefore $sim(q, c_j) = \frac{(q,c_j)}{|q| \cdot |c_j|}$, where (q, c_j) is the scalar product between q, and c_j.

By denoting with c^u the user profile generated category, and with c^g the general user profile there are five possible ways of computing the similarity between a query vector q and a category c:

- Using only the user profile: $sim(q, c) = sim(q, c^u)$
- Using only the general profile: $sim(q, c) = sim(q, c^g)$
- Combo1: $sim(q, c) = \frac{1}{2} \cdot (sim(q, c^u) + sim(q, c^g))$
- Combo2: $sim(q, c) = 1 - (1 - sim(q, c^u)) \cdot (1 - sim(q, c^g))$
- Combo3: $sim(q, c) = \max(sim(q, c^u), sim(q, c^g))$

The accuracy is defined as:

$$\text{Accuracy} = \frac{1}{n} \sum_{c_j \in \text{topK}} \frac{1}{1 + \text{rank}_{c_i} - \text{ideal\_rank}_{c_i}}$$

and it is computed in a user study to evaluate the effectiveness of the method.

In the formula above, *topK* are the K category vectors having the highest cosine similarity measure with the query, rank_{c_i}, is an integer ranging from 1 to K and is the rank of category c_i as computed by $sim(q, c_j)$, ideal_rank_{c_i} is the rank assigned by users in the human-generated rank. Liu *et al.* [138] tested the methods by setting $K = 3$ (i.e. the top 3 scoring categories).

For instance, if categories c_1, and c_2 are ranked as first and second by the system, and first and third by humans the Accuracy is given by

$$\text{Accuracy} = \frac{1}{2} \left(\frac{1}{1+1-1} + \frac{1}{1+2-3} \right) = 0.5$$

The number of users surveyed in the original paper is seven, each one of them evaluating an amount of queries ranging from 26 to 61.

As a first result in Table 4.4a it is shown the comparison of the average accuracy of the different learning methods computed over the seven users when using only the user profile.

Method	pLLSF	LLSF	bRocchio	kNN
Avg Accuracy	0.8236	0.7843	0.8224	0.8207

(a) Average accuracy of the different learning methods using only the user profile over the seven users surveyed.

Method	User	General	Combo1	Combo2	Combo3
Avg Accuracy	0.8224	0.7048	0.8936	0.8917	0.8846

(b) Average accuracy of the different profile combination methods using non-adaptive Rocchio learning algorithm.

Fig. 4.4: Average accuracy of the different profile combination methods [138].

In Table 4.4b the accuracy of the ranking computed is measured by considering all the possible combinations and the Rocchio's algorithm as the learning method.

A nice result is obtained from the analysis of the adaptive learning algorithm. The main observations reported by Liu *et al.* [138] are:

- When the dataset on which the model is computed is small, accuracy of using the user search history derived profile is worse than the one learnt from the general profile. Therefore, the accuracy of combining both profiles is better than those using a single profile and the accuracy of using the user profile only is better than that using the general profile only, provided that there is sufficient historical data.
- For all the datasets the accuracy of combining user and general profiles is better than that using only one of them.
- As more and more data comes into the adaptive model, the user profile based model gets better and better.
- The accuracy approaches to 1 as the dataset increases.

Actually, considering the extremely high variability of queries in search engines, the adaptive learning method should be the one of choice because it is able to adapt promptly to this variations without requiring any efforts from search engine maintainers[7].

Another different approach followed by Boydell and Smith [45] is based on the use of snippets of clicked results to build an index for personalization. First of all, personalization is done, through re-ranking of the search results, at the proxy-side. Therefore, the technique described in the paper does not require the storing of usage information at the server-side. Furthermore, not requiring the storing of usage information at the server side makes this approach harmless with respect to the problems of users' privacy mentioned in Introduction. This approach, indeed, falls into the

[7] This is a similar observation made by Baraglia and Silvestri [32] in the context of online web recommender systems.

category of those collecting data proxy-side. Documents and queries collected for a subset of users (a community) are used.

The method is quite straightforward, we use the notation adopted by the authors to keep the discussion as similar as possible to that of the original paper. Let (C, u, q_T) be a search for query q_T by user u in the community C. Let selected$(C, q_T, r) = true$ if a result r has been selected when returned in response to a query q_T. The snippet for r is $s(r, q_T) = t_1, \ldots, t_n$, and can be considered as a surrogate for the *real* document r within the context of (C, u, q_T). The document r is thus indexed by the system only by considering $s(r, q_T)$. Since the same document r can be selected for different queries submitted by users of community C, the surrogate is actually represented as the union of all the $s(r, q_i)$ vectors.

$$S^C(r) = \bigcup_{i \text{ s.t. selected}(C,q_i,r)} s(r, q_i)$$

The rationale behind this representation is that it actually stores the relevant *parts* of a document r for the community C. The snippets are indexed by the Lucene[8] IR system. In addition to this local index, a *hit-matrix* H is used. Each entry of the hit-matrix $H_{i,j}$ stores the number of times result r_j has been selected in addition to query q_i.

Given a query q_T submitted within the community C, and a result r_j for which selected$(C, q_T, r_j) = true$, we firstly retrieve all the queries $q_1, \ldots, q_n$ that has been previously submitted and that has caused the selection of document r_j. The relevance of r_j for query q_T is computed using the following equation

$$\text{Relevance}(r_j, q_T, q, \ldots, q_n) = \text{TF} - \text{IDF}(r_j, q_T) \cdot (1 + \sum_{i=1}^{n} (\text{Rel}\,(r_j, q_j) \cdot \text{QuerySim}\,(q_T, q_i)))$$

where $\text{Rel}\,(r_j, q_j) = \frac{H_{i,j}}{\sum_{\forall j} H_{i,j}}$; and $\text{QuerySim}\,(q_T, q_i) = \frac{|q_T \cap q_i|}{|q_T \cup q_i|}$ (i.e. the Jaccard's Distance [25] between q_T, and q_i).

Also this method has been assessed through a user study. The study monitored a group of users for a period of 2 weeks recording a total of 430 search sessions. Results are recorded both at the end of the first week, and at the end of the entire period.

Metric	**Week 1**	**Week 2**
Total sessions	246	184
Overall Success Rate	41%	60%

Table 4.1: The success rate of the re-ranking algorithm as computed from the user study by Boydell and Smith [45].

As it is shown in Table 4.1 many users found the re-ranking of search result useful, in particular it is evident a raise in the second week denoting (in agreement with some of the conclusions drawn by Liu *et al.* [138]) that a longer training period produces a beneficial effect in the quality of personalization.

[8] `http://lucene.apache.org/`

In a recently presented work by Dou *et al.* [73], a *large-scale* evaluation of personalization strategies is shown. In particular, the study differs deeply from the previous ones since *it does not present any user studies*, but instead exploits an evaluation function based on sessions extracted from query logs. The MSN search engine, along with a query log coming from the same engine is used for the testing framework.

Some conclusions in the paper by Dou *et al.* [73] are the following.

- Personalization may lack of effectiveness on some queries and there is no need for personalization on those queries. Typically navigational queries do not have many advantages in using or not personalization. For example for the query "*Digg*", users tend to not look too much into the search results. The first result, usually showing the target's home page, is selected.
- Different strategies may have variant effects, and therefore variant effectiveness, on different queries. For instance, for the query "*free mp3 download*" all the previous methods will result ineffective, because those techniques are well-suited to queries on multiple topics (e.g. "*mouse*").
- Users that have submitted few queries in the past do not benefit too much from personalization, on the other hand users with a *short-term* user need are penalized by results that are personalized on the basis of the stratified knowledge. Again, as an example, the query "*mouse*" will likely produce a lot of results from the computer category for a user having submitted the majority of his queries in that category. If, for some reasons, the query was submitted to intentionally look for information on mice, the user experience will not be improved.

Therefore,

> *the effectiveness of a specific personalized search strategy may show great improvement over that of non personalized search on some queries for some users, and under some search contexts, but it can also be unnecessary and even harmful to search under some situations.* [73]

The evaluation framework for re-ranking that has been considered by Dou *et al.* [73], is made up of four parts:

(1) *Query results retrieval.*
(2) *Personalization.*
(3) *Ranked lists combination.*
(4) *Evaluation of personalization effectiveness.*

In *Query results retrieval* the top 50 search results are obtained from the MSN search engine for the query being tested. Furthermore, let U be the set of downloaded web pages, and let τ_1 be the ranking of pages in U as returned by the search engine. In *Personalization* phase, a personalization algorithm (see below) is used to generate a new list τ_2 from U by ranking its elements according to the personalization score computed. The *Ranked lists combination* phase uses the Borda fusion algorithm [220] to merge τ_1, and τ_2 into the final ranked list, τ, that is proposed to user. The Borda fusion method is very straightforward, it is a voting system in which each voter ranks the list of

n candidates in order of preference. Within this list the first candidate scores n points, the second scores $n-1$, down to the last one who scores 1 point. The final score of each candidate is given as the sum of each score in each voter's ranked list. Considering our problem, suppose U is composed of four pages $U = \{a, b, c, d\}$, suppose $\tau_1 = (abcd)$, and $\tau_2 = (acdb)$. According to the Borda fusion method the final list $\tau = (acbd)$ with scores, 8, 5, 4, and 3 respectively. The Borda scoring system is useful whenever the original ranking scores from the search engine are not available. Otherwise, the ranking could have been computed using one of the three methods (i.e. Combo1, Combo2, Combo3) shown above. In the *Evaluation of personalization effectiveness* step the personalization is evaluated in a totally automatic manner. The assumption we make is that results are consistent with those actually seen by the users who submitted the queries in their used query log. The log was referring to queries submitted in August 2006, and the study was conducted in September 2006 therefore they were able to ignore the effect of the index updates. Personalization is analyzed under two different perspectives. A *person-level* re-ranking strategy considers the history of a single user to carry out personalization. A *group-level* re-ranking, instead, focuses on queries and results of a community of (typically homogeneous) people.

Person level re-ranking strategies include **P-Click**, **L-Profile S-Profile**, and **LS-Profile**. Group level re-ranking is computed using the **G-Click** score. Let q be a query submitted by user u. The personalized score for the page p is computed by the different methods in the following ways.

P-Click. The score $S^{\mathrm{P-click}}(q, p, u)$ is computed using the count of clicks on p by u on query q, namely $|\mathrm{Clicks}(q, p, u)|$, and the total number of clicks made by the same user on the same query q, i.e. $|\mathrm{Clicks}(q, \bullet, u)|$

$$S^{\mathrm{P-click}}(q, p, u) = \frac{|\mathrm{Clicks}(q, p, u)|}{|\mathrm{Clicks}(q, \bullet, u)| + \beta}$$

where β is a smoothing factor ($\beta = 0.5$ in the paper by Dou *et al.* [73]) that together with $|\mathrm{Clicks}(q, \bullet, u)|$ is used to normalize the score. Actually, in fact, only $|\mathrm{Clicks}(q, p, u)|$ matters for the purpose of scoring the triple (q, p, u). The main drawback of this approach is that whenever a user submit a query not previously seen, personalization does not take place. As seen in User Action chapter, two-thirds of queries are submitted only once, thus, the method does not impact too much on personalized results.

L-Profile. The score $S^{\mathrm{L-Profile}}(q, p, u)$ is, in this case, computed by using a user profile specified as a vector $c_l(u)$ of 67 *pre-defined* topic categories defined by the 2005 KDD Cup [137]. Let $\mathcal{U}(p)$ be the number of users that have ever clicked on p, and $\mathcal{U}$ the total number of users that have ever clicked on some pages. Let $w(p)$ be the weight of page p within the user's history computed as

$$w(p) = \log \frac{|\mathcal{U}|}{\mathcal{U}(p)}$$

Furthermore, let $|\mathrm{Clicks}(\bullet, \bullet, u)|$ the total number of clicks of u, and let $|\mathrm{Clicks}(\bullet, p, u)|$ be the number of clicks made by user u on page p. The probability that user u clicks on page p, $P(p|u)$ is computed as

$$P(p|u) = \frac{|\mathrm{Clicks}(\bullet, p, u)|}{|\mathrm{Clicks}(\bullet, \bullet, u)|}$$

. The category vector of a web page p, namely $c(p)$ is computed by means of a page classifier developed by Shen *et al.* for the KDD-cup2005 [193]. Each component $c(p)_i$ is the classification confidence returned by Q^{2}C@UST that indicates the probability for a page p to be in the i-th category. If category i is not returned by the tool then we set $c(p)_i = 0$. Let $\mathcal{P}(u)$ be the collection of pages visited by u, the user profile $c_l(u)$ is automatically learnt by past user's clicked pages by using the following formula

$$c_l(u) = \sum_{p \in \mathcal{P}(u)} P(p|u) w(p) c(p)$$

and the final personalization score is computed as

$$S^{\mathrm{L-Profile}}(q, p, u) = \frac{c_l(u) \cdot c(p)}{\|c_l(u)\| \|c(p)\|}$$

S-Profile. The previous method has the characteristic of accumulating the stratified knowledge about *all* the queries submitted in the past. Sometimes it is better, for personalization purposes, to consider only the most recently seen pages by u: $\mathcal{P}_s(q)$ is the set of pages visited in the current session with respect to query q. The vector $c_s(u)$ is a user short-term profile and is computed as

$$c_s(u) = \frac{1}{|\mathcal{P}_s(q)|} \sum_{p \in \mathcal{P}_s(q)} c(p)$$

The personalization **S-Profile** is then computed as

$$S^{\mathrm{S-Profile}}(q, p, u) = \frac{c_s(u) \cdot c(p)}{\|c_s(u)\| \|c(p)\|}$$

where $c(p)$ is computed as in the case of the **L-Profile** scoring formula described above.

LS-Profile. The score, in this case, is obtained by a linear combination of the previous two methods and is given by

$$S^{\mathrm{LS-Profile}}(q, p, u) = \theta S^{\mathrm{L-Profile}}(q, p, u) + (1 - \theta) S^{\mathrm{S-Profile}}(q, p, u)$$

G-Click. To test group-based personalization a kNN approach is used. In this case the personalization is based on the k users having closest preferences with the current user. The similarity between to users u and u' is given by

$$\mathrm{Sim(u, u')} = \frac{c_l(u) \cdot c_l(u')}{\|c_l(u)\| \|c_l(u')\|}$$

and it is used to compute the k nearest neighbors of u as follows

$$\mathcal{S}_u(u) = \{u' | rank(sim(u, u')) \leq k\}$$

The final re-ranking score is, then, computed as

$$S^{\mathrm{G-Click}}(q, p, u) = \frac{\sum_{u' \in \mathcal{S}_u(u)} \mathrm{Sim(u, u')} |\mathrm{Clicks}(q, p, u')|}{\beta + \sum_{u' \in \mathcal{S}_u(u)} |\mathrm{Clicks}(q, \bullet, u')|}$$

To test the performance of the various strategies, a MSN query logs collecting 12 days worth of queries submitted in August 2006 was used. The main important difference between the evaluation performed in this paper and those of previous papers is that authors do not make use of any user study. Instead, they use information about past clicks done by users to evaluate the relevance of the personalized ranking computed. In particular, evaluation is done through the use of two measurements: *Rank Scoring*; *Average Rank*.

Rank Scoring. The method has been proposed by Breese [46] to evaluate the performance of a recommender system using collaborative filtering techniques to produce a ranked list of suggestions to users. The method is aimed at approximating the expected utility of a ranked list of pages for a particular user. Let j be the rank of a page in the evaluated re-ranking, let $\delta(s, j)$ be a boolean function evaluating to 1 if and only if j is clicked when returned in answer to the query s, let α be a normalizing factor: the *expected utility* for page s in the ranked list is given by

$$R_s = \sum_j \frac{\delta(s, j)}{2^{(j-1)/(\alpha-1)}}$$

The final rank scoring is the sum (normalized by the sum of maximum utilities $R_s^{\max}$) of the utilities of all entries of the list

$$R = 100 \frac{\sum_s R_s}{\sum_s R_s^{\max}}$$

Larger values of R mean better performance of the personalization algorithm.

Average Rank. The average rank has been used in other papers [176, 208] to evaluate the effectiveness of the personalization strategy. The average rank for a ranked list of results is defined in terms of the sum of its items. Let $\mathcal{P}_s$ be the set of clicked pages on test query s, let $R(p)$ the rank of page p, the *average rank for a page s* is defined as

$$\text{AvgRank}_s = \frac{1}{|\mathcal{P}_s|} \sum_{p \in \mathcal{P}_s} R(p)$$

and the final average rank on the query set $\mathcal{S}$ is computed as:

$$\text{AvgRank} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \text{AvgRank}_s$$

Table 4.2 shows a comparison of the various methods seen thus far on the MSN query log. The method defined as WEB represents the performance of the considered web search engine without any personalization and, thus, it represents the baseline for comparisons.

Results under the column *all*, correspond to the entire query log, while the *not-optimal* column corresponds to the performance of personalization on queries whose top result was not the one selected by users. That is the queries on which the search engine performed poorly. Click-based methods always outperform the baseline showing that, in general, click-through data can bring benefits to personalization on the web.

The results shown in Table 4.2 are computed over the whole query log and represent the aggregate, and final, effectiveness figure for the proposed methods. As it has been said many times,

method	all		not-optimal	
	Rank Similarity	Average Rank	Rank Similarity	Average Rank
WEB	69.4669	3.9240	47.2623	7.7879
P-Click	**70.4350**	**3.7338**	**49.0051**	**7.3380**
L-Profile	66.7378	4.5466	45.8485	8.3861
S-Profile	66.7822	4.4244	45.1679	8.3222
LS-Profile	68.5958	4.1322	46.6518	8.0445
G-Click	70.4168	3.7361	48.9728	7.3433

Table 4.2: Overall performance of personalization methods shown by Dou *et al.* [73]. For the method G-Click K is set to 50, whereas for the method LS-Profile $\theta = 0.3$.

personalization is only effective whenever the variance in results clicked for a query is high. This, in fact, means that for a single query there are many topics associated with a single result. A measurement that can be computed to evaluate the degree of result variance for a given query is the *Query Click Entropy*.

Let q be a query, let p be a page, and let $\mathcal{P}(q)$ be the set of web pages clicked on query q. Let $P(p|q)$ be the percentage of clicks on page p when returned as an answer to query q, i.e.

$$P(p|q) = \frac{|\text{Clicks}(q, p, \bullet)|}{|\text{Clicks}(q, \bullet, \bullet)|}$$

The Query Click Entropy is defined as

$$\text{ClickEntropy}(q) = \sum_{p \in \mathcal{P}(q)} -P(p|q) \log_2 P(p|q)$$

Obviously $\text{ClickEntropy}(q) = 0$ if and only if $\log_2 P(p|q) = 0$, i.e. $P(p|q) = 1$. Therefore, the minimum entropy is obtained when clicks are *always* on the same page. Personalization, in this case, is of little (or no) utility. Personalization might help in the case of zero entropy, only if the clicked page is not the top ranked one, but this is very unlikely to happen.

Figure 4.5 shows the distribution of the click entropy for the MSN query log. The majority of the queries (about 70%) exhibits a very low entropy value (0 - 0.5) meaning that clicks, in this case, were almost all referred to the same page. In terms of personalization this means that in, almost, 70% of the cases personalization does not help. This fact is also confirmed in the overall results appearing in Table 4.2 where the improvements of the various methods over the baseline are sensitive.

Considering this low entropy in query clicks, remain important to evaluate the variation of the accuracy measurements when entropy varies.

Figure 4.6a plots the variation in *Ranking scoring*, and Figure 4.6b plots the variation of the *Average rank* when varying the entropy level.

The first observation is that the greater the query click entropy the better the performance. Both Ranking scoring, and Average Rank perform better at higher entropy levels. Roughly speaking, this means that whenever accuracy improvement is needed (on high variance query results) personalization is of great help.

The second observation is that, as in the case of the overall results shown above, the click-based methods sensibly outperformed the profile-based ones. This seems to be in contrast with

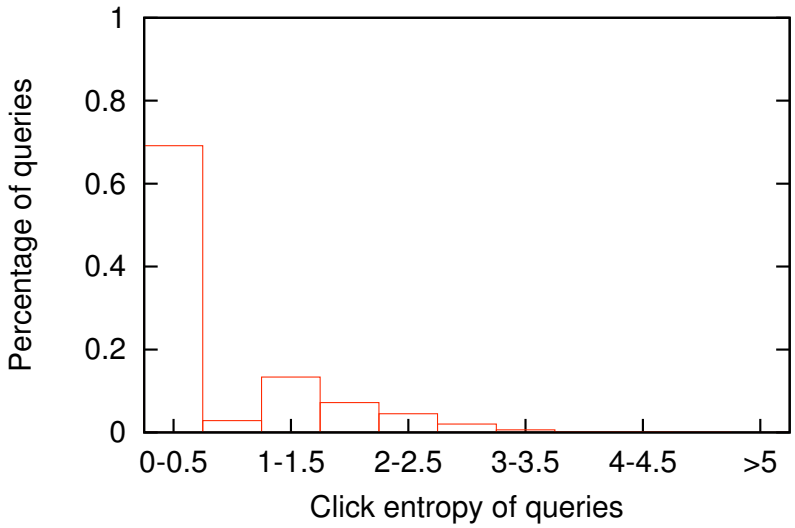


Fig. 4.5: Click entropy distribution for the MSN query log [73].

the results shown in literature so far. Dou *et al.* [73] state that this might have been due to a "*rough implementation*" of their system. Actually, a deeper analysis have shown that profile based strategies, especially the L-Profile, suffer of an inability to adapt to variation of users' information needs.

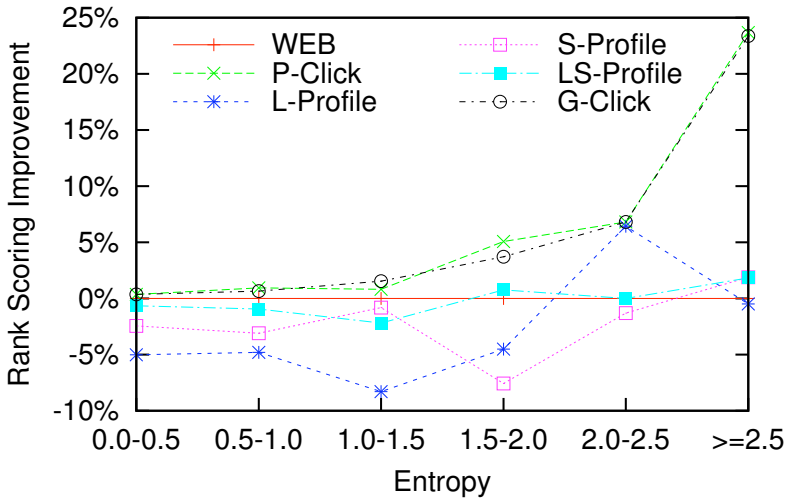
Figure 4.7 shows that: (i) profile-based methods perform better when the number of queries submitted by users is around 70-80, this is due to the fact that such a number of queries forms a good repository of knowledge that can be effectively exploited by the system; (ii) in all the other cases click-based methods outperform profile-based methods and improve over the baseline; (iii) when the number of queries submitted by each user increase and becomes greater than (approximately) 90 the profile methods collapse. This last phenomenon might be explained by the fact that the higher the number of queries, the longer the period within which they have been submitted, the higher the probability that user needs has been changed.

The literature on personalization is quite rich. We have not analyzed, in this work, relevant papers such as [218, 78, 194, 144, 62, 225]. We trust in the keen reader and we leave them the pleasure of reading them.

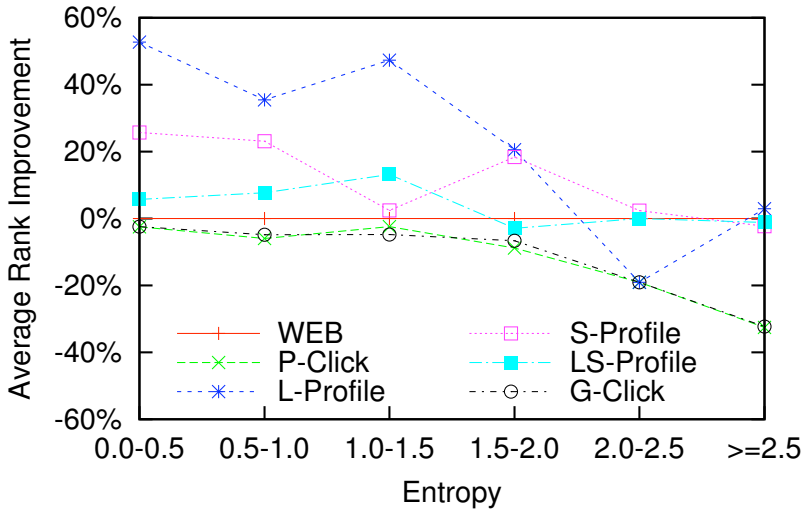
4.5 Learning to Rank

Using machine learning techniques [150] in text categorization (thus in search engine documents characterization) has a long tradition [192]. Starting more than ten years ago [44], machine learning techniques have been extensively studied to derive ranking functions in web search engines.

Differently from personalized ranking, the aim of "learning to rank" techniques [117] is to compute a *global* (i.e. independent from user), model to compute relevance scores for each page. Basically, it works by firstly selecting the best features to be used to identify the importance of a page, and then by training a machine learning algorithm using these features on a subset (i.e. the



(a) Ranking scoring.



(b) Average rank.

Fig. 4.6: Search accuracy improvement over the baseline method (WEB) on varying the entropy measure [73].

training set corpus) of the web pages. The focus in this survey is on using query logs to generate training data for learning to rank algorithms. To this extent, learning to rank algorithms will be discussed briefly, for context, but remember that the focus is on using logs to generate training data.

More specifically the aim of learning to rank is aimed to *learn* (as in the machine learning meaning) a function for ranking objects. Learning to rank is useful for document retrieval, collaborative filtering, and many other applications. We are concerned, in this work, to learning functions that are able to evaluate the *importance* of a document d in answer to a query q. Defined this way, it resembles much the definition of general ranking [221], the main difference is that the ranking function is not user generated but it is *learnt* by using a set of features that generally takes into ac-

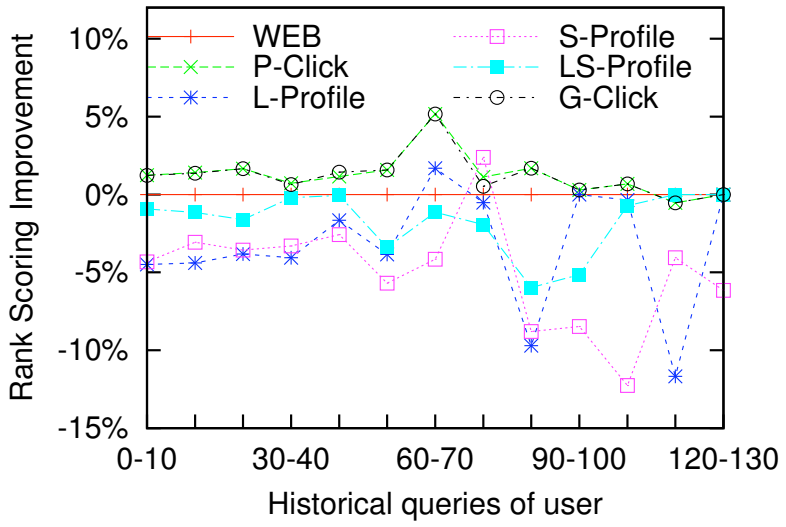


Fig. 4.7: Rank scoring over the WEB method on varying the number of queries submitted by each user [73].

count more than term or document frequencies (that are, instead, typical of traditional IR ranking functions). Furthermore, many methods do not make use of any information coming out of query logs, we do not include those studies in this survey.

One of the seminal papers on this topic has been presented in 1996 at an AAAI workshop on Internet-Based Information Systems. In their paper Boyen *et al.* [44] present LASER: a Learning Architecture for search engine Retrieval. The system is based on machine learning techniques for the ranking module.

As a side note, not really on topic with the current Text, it is worth mentioning that in 1996, when neither PageRank [47] nor HITS [125] were yet proposed, the paper started from the observation that the web was composed of hypertext and links. LASER was the first (as far as we are aware of) to introduce the concept of *reward propagation through the hypertext graph*. Given a *retrieval status value* ($\text{rsv}_0(q,d)$) measuring the likeliness of the document d being relevant for the query q, by a *value iteration* process [39] the rsv score after t link traversals is given by

$$\text{rsv}_{t+1}(q,d) = \text{rsv}_0(q,d) + \gamma \sum_{d' \in \text{links}(d)} \frac{\text{rsv}_t(q,d')}{|\text{links}(d)|^\nu}$$

where γ is a discounting factor governing the propagation of page weights through links, $\text{links}(d)$ is the neighboring set of d, ν is used for normalization purposes. This formula resembles very much that of PageRank except that instead of considering *inlinks* it propagates weights through *outlinks* (much more in the spirit of Marchiori's paper on hyper search engines [146]).

In traditional IR experiments, ranking precision has been measured with the help of a popular benchmark: the TREC collection [97]. Relevance judgements were provided for the precision to be evaluated in a scientific (i.e. reproducible) way.

The ranking precision of a web search engine, instead, is very difficult to evaluate. Basically, in

shortage of humans devoted to evaluate the quality of results for queries the only way that can be followed is to evaluate how results are clicked by users on query results. Click-through information is thus used to infer relevance information: *if a document receives a click it is relevant for the query it has answered*. Therefore, if f is a ranking function we can define its performance as the average rank of the clicked results, i.e.

$$\text{Perf}(f) = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{|D_i|}\sum_{j=1}^{|D_i|}\text{rank}(f, Q_i, D_{ij})$$

where $Q_1 \dots Q_{|Q|}$ are the queries over which click-through data has been collected, D_i is the set of documents clicked in answer to Q_i. The performance metric is very straightforward: if for query q_1 user clicks on the first, the second and the fourth, and for the query q_2 user clicks on the second and the third, than $\text{Perf}(f) = \frac{1}{2}\left(\frac{7}{3} + \frac{6}{2}\right) = 2.67$

Getting back to LASER, the method we are considering, its aim is to learn from users (implicit) feedback on past queries. Fixing the objective of finding the argument f minimizing $\text{Perf}(f)$ a variant of the simulated annealing [170] is used. Actually, the performance of the learning algorithm using click-through data as a feature in learning is only assessed for a single experiment. Preliminary results, though, were very promising thus, motivating further investigations (see Table 4.3).

standard TF-IDF	**automatically learned parameters**
1. Vegetarian Chili Recipes	1. **Eating "Indian" in Pittsburgh**
2. Vegetarian Recipes	2. **Restaurant Reviews**
3. **Eating "Indian" in Pittsburgh**	3. A list of food and cooking sites
4. **Restaurant Reviews**	4. Duane's Home Page & Gay Lists
5. Greek Dishes	5. For the Professional Cook
6. Focus on Vegetarian	6. Eating & Shopping Green in Pittsburgh
7. For the Professional Cook	7. Vegetarian Recipes
Score: 3.5	*Score: 1.5*

Table 4.3: Rankings produced by the standard TF-IDF scoring, and the learnt f function for the query "*vegetarian restaurant*"[44].

To be precise, it is worth saying that the technique does not only use click-through information for learning the ranking. Among its features, in fact, the containment in some special headlines (H1, H2, etc.), the containment within title, bold, italic, blink modifiers, and the appearance as anchor text, are also considered.

After this seminal work many other papers have been published on this topic. In particular, works in [114, 115, 131, 177, 5, 6, 4, 118, 237, 178], show techniques operating on different features extracted from query logs.

Among the others two popular approaches emerged in these last years: *RankSVM* [114], and *RankNet* [51].

Joachims [114] lays the foundations of the RankSVM technique. Stemming from the observation that a click on a result is not an unbiased estimator for the importance of the relative web page, Joachims looks for a set of query log features that could give an unbiased estimate of user's perceived relevance for a web page [114, 116].

The fact that users click more often on the first result than on the others seem to be related with a trust feeling with the search engine ranking. Looking back at Figure 4.2 at page 34 it can be observed that, even if the first and the second results are swapped, the bars denoting their percentage of clicks are not swapped as well.

The key observation is that:

> *click is not an unbiased estimator of the absolute importance of a page, yet, since users scan a page from top to bottom, clicking on a result is likely to be a sign that the user retains that result more important than the previous ones.*

In other words: a click is not an unbiased indication of the absolute importance of a page. However, because people usually scan search results from top to bottom, a click on a result is evidence that it may be more important than unclicked results that appeared before it in the ranking.

Starting from the previous key observation Joachims *et al.* [116] propose a series of strategies to extract relevance feedback from click-through data. All the strategies are better explained through an example: let q be a query returning result pages p_1 to p_7, suppose a user clicks on pages p_1, p_2, p_4, and p_7, i.e.:

$$p_1^*, p_2^*, p_3, p_4^*, p_5, p_6, p_7^*$$

The following strategies for extracting feedback can be defined.

Strategy 4.1 (*Click* > *Skip Above*). For a ranking $(p_1, p_2, \ldots)$ and a set C containing the ranks of the clicked-on links, extract a preference example $\text{rel}(p_i) > \text{rel}(p_j)$ for all pairs $1 \leq j < i$ with $i \in C$ and $j \notin C$[9].

From the running example, using Strategy 4.1, the features $\text{rel}(p_4) > \text{rel}(p_3)$, $\text{rel}(p_7) > \text{rel}(p_5)$, $\text{rel}(p_7) > \text{rel}(p_3)$, and $\text{rel}(p_7) > \text{rel}(p_6)$ are extracted. In other words, the strategy assumes that when a user clicks on a result retains all the previous results not relevant.

Strategy 4.2 (*Last Click* > *Skip Above*). For a ranking $(p_1, p_2, \ldots)$ and a set C containing the ranks of the clicked-on links, let $i \in C$ be the rank of the link the was clicked temporally last. Extract a preference example $\text{rel}(p_i) > \text{rel}(p_j)$ for all pairs $1 \leq j < i$ with $j \notin C$.

From the running example, using Strategy 4.2, the features $\text{rel}(p_7) > \text{rel}(p_6)$, $\text{rel}(p_7) > \text{rel}(p_5)$, and $\text{rel}(p_7) > \text{rel}(p_3)$ are extracted. In other words, the strategy assumes that only the last click counts and it expresses the relevance of the clicked results with respect to all the previously unclicked ones. The set of relevance features extracted using Strategy 4.2 is a subset of those extracted by Strategy 4.1.

Another possible assumption is that the abstracts that are most reliably evaluated are those immediately above the clicked link. This leads to the following strategy, which generates constraints only between a clicked link and a not-clicked link immediately above.

[9] $\text{rel}(\cdot)$ is the function measuring the relevance of a page: $\text{rel}(p_i) > \text{rel}(p_j)$ means p_i is more relevant than p_j in the click-set C.

Strategy 4.3 (*Click > Earlier Click*). For a ranking $(p_1, p_2, \ldots)$ and a set C containing the ranks of the clicked-on links, let $t(i)$, $i \in C$ be the time when the link was clicked. We extract a preference $\text{rel}(p_i) > \text{rel}(p_j)$ for all pairs j and i with $t(i) > t(j)$.

From the running example, using Strategy 4.3, assume pages are clicked in this order p_4, p_1, p_2, p_7, we can extract the following features: $\text{rel}(p_1) > \text{rel}(p_4)$, $\text{rel}(p_2) > \text{rel}(p_4)$, $\text{rel}(p_2) > \text{rel}(p_1)$, $\text{rel}(p_7) > \text{rel}(p_4)$, $\text{rel}(p_7) > \text{rel}(p_1)$, and $\text{rel}(p_7) > \text{rel}(p_2)$. As in the previous strategy the idea that later clicks are more informed decisions than earlier clicks is followed. But, stronger than the "Last Click > Skip Above", it is now assumed that clicks later in time are on more relevant abstracts than earlier clicks.

Strategy 4.4 (*Click > Skip Previous*). For a ranking $(p_1, p_2, \ldots)$ and a set C containing the ranks of the clicked-on links, extract a preference example $\text{rel}(p_i) > \text{rel}(p_{i-1})$ for all $i \geq 2$ with $i \in C$ and $(i-1) \notin C$.

From the running example, using Strategy 4.4, the features $\text{rel}(p_4) > \text{rel}(p_3)$, and $\text{rel}(p_7) > \text{rel}(p_6)$ are extracted. The strategy is motivated by the fact that the abstracts that are most reliably evaluated are those immediately above the clicked link. This leads to the Click > Skip Previous strategy, which generates constraints only between a clicked link and a not-clicked link immediately above. The same assumption can also lead to the following strategy.

Strategy 4.5 (*Click > No-Click Next*). For a ranking $(p_1, p_2, \ldots)$ and a set C containing the ranks of the clicked-on links, extract a preference example $\text{rel}(p_i) > \text{rel}(p_{i+1})$ for all $i \in C$ and $(i+1) \notin C$.

From the running example, using Strategy 4.4, the features $\text{rel}(p_2) > \text{rel}(p_3)$, and $\text{rel}(p_4) > \text{rel}(p_5)$ are extracted.

Accuracy of relevance samples extracted using these different strategies is very difficult to measure. In Joachims *et al.* [116] an approach tackling, once more, a user study has been used. Table 4.4 shows the percentage of automatically extracted pairs that are in agreement with the user generated explicit relevance judgements.

As it can be seen all the features are performing fairly above the baseline which corresponds to the performance of the random extraction, i.e. 50%, in particular the best performing one is the *Last Click > Skip Above*, which outperforms all of the others. In particular, the three methods *Last Click > Skip Above*, *Click > Skip Previous*, and *Click > Skip Above* performed equally better also when the first two results were swapped[10].

Users usually do not issue just a single query and then stop: whenever they are looking for an information, instead of a precise website, they tend to issue more than a single query until they have satisfied their needs. *Query Chains* can be exploited to infer implicit relevance feedback on document clicks in sequences of user queries.

[10] Recall from the discussion done in the introductory paragraph of this chapter that the number of clicks a given position obtained in two different conditions: *normal* and *swapped*, i.e. the first two results were swapped, rank is, more or less, stable. See Figure 4.2.

Strategy	Features per Query	Normal (%)	Swapped (%)
Inter-Judge Agreement	N/A	89.5	N/A
Click > Skip Above	1.37	88.0 ± 9.5	79.6 ± 8.9
Last Click > Skip Above	1.18	89.7 ± 9.8	77.9 ± 9.9
Click > Earlier Click	0.20	75.0 ± 25.8	36.8 ± 22.9
Click > Skip Previous	0.37	88.9 ± 24.1	80.0 ± 18.00
Click > No Click Next	0.68	75.6 ± 14.1	66.7 ± 13.1

Table 4.4: Accuracy of the strategies shown above for generating pairwise preferences from clicks within a single query. Feature-Per-Query shows the average number of features extracted per each query, the Swapped Column represents the experiment with the first two results swapped. These figures have been extracted from the paper by Joachims *et al.* [116]. The Inter-Judge Agreement column correspond to the average agreement with which judges have scored the different results.

To exploit sequentiality in submissions of different queries for the same information need, other six strategies are proposed.

Let us consider the following example used, as in the description of the strategies above, to better explain the query chain-related strategies. Consider four subsequent lists of pages returned in answer to four queries pertaining to the same chain. I.e.

$$p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}, p_{17}$$

$$p^*_{21}, p_{22}, p^*_{23}, p_{24}, p^*_{25}, p_{26}, p_{27}$$

$$p_{31}, p^*_{32}, p_{33}, p_{34}, p_{35}, p_{26}, p_{37}$$

$$p^*_{41}, p_{42}, p_{43}, p_{44}, p_{45}, p_{36}, p_{47}$$

again the asterisk, *, means that the result has been clicked by the user.

Strategy 4.6 (*Click > Skip Earlier QC*). For a ranking $(p_1, p_2, \ldots)$ followed (not necessarily immediately) by ranking $(p'_1, p'_2, \ldots)$ within the same query chain and sets C, and C' containing the ranks of the clicked-on links in either ranking, extract a preference example $\text{rel}(p'_i) > \text{rel}(p_j)$ for all pairs $i \in C'$ and $j < \max(C)$ with $j \notin C$.

For the above example strategy 4.6 produces the following set of examples: $\text{rel}(p_{32}) > \text{rel}(p_{22})$, $\text{rel}(p_{32}) > \text{rel}(p_{24})$, $\text{rel}(p_{41}) > \text{rel}(p_{22})$, $\text{rel}(p_{41}) > \text{rel}(p_{24})$, and $\text{rel}(p_{41}) > \text{rel}(p_{31})$. The strategy is, thus, an analogous extension of "Click > Skip Above" to multiple result sets. A preference is generated between two links from different result sets within the same query chain, if a link in an earlier result set was skipped and a link in a later result set was clicked.

To improve the accuracy of the preferences, we may consider the subset of preferences generated only by the last click in a query chain.

Strategy 4.7 (*Last Click > Skip Earlier QC*). For a ranking $(p_1, p_2, \ldots)$ and a set C containing the ranks of the clicked-on links. If the last ranking $(p'_1, p'_2, \ldots)$ within the same query chain received a click, the let i the the temporally last click in this ranking and extract a preference example $\text{rel}(p'_i) > \text{rel}(p_j)$ for all pairs $j < \max(C)$ with $j \notin C$.

Strategy 4.7 produces the following set of features for our running example: $\text{rel}(p_{41}) > \text{rel}(p_{22})$, $\text{rel}(p_{41}) > \text{rel}(p_{24})$, and $\text{rel}(p_{41}) > \text{rel}(p_{31})$.

In analogy to "Click > Earlier Click" for within query preferences, the following strategy explores the relationship between pairs of clicked links between queries. In particular, it generates a preference between a clicked link of an earlier query and a clicked link of a later query in the same query chain.

Strategy 4.8 (*Click > Click Earlier QC*). For a ranking $(p_1, p_2, \ldots)$ followed (not necessarily immediately) by ranking $(p'_1, p'_2, \ldots)$ within the same query chain and sets C, and C' containing the ranks of the clicked-on links in either ranking, extract a preference example $\text{rel}(p'_i) > \text{rel}(p_j)$ for all pairs $i \in C'$ and $j \in C$.

Strategy 4.8 produces the following set of features for our running example: $\text{rel}(p_{32}) > \text{rel}(p_{21})$, $\text{rel}(p_{32}) > \text{rel}(p_{23})$, $\text{rel}(p_{32}) > \text{rel}(p_{25})$, $\text{rel}(p_{41}) > \text{rel}(p_{21})$, $\text{rel}(p_{41}) > \text{rel}(p_{23})$, $\text{rel}(p_{41}) > \text{rel}(p_{25})$, and $\text{rel}(p_{41}) > \text{rel}(p_{32})$.

One shortcoming of the two strategies "Click > Skip Earlier QC" and "Last Click > Skip Earlier QC" is that they generate preferences only if an earlier query within the chain drew a click. However, about 40% of all queries does not receive any clicks. For such queries without clicks, Joachims *et al.* [116] observed using eye-tracking techniques that show that users typically view the top links [90]. For queries without clicks, it is therefore assumed that the user evaluated the top two links and decided to not click on them, but rather to reformulate the query. This leads to the following two strategies, where a preference is generated between a clicked link in a later query, and the first (or second) link in a earlier query that received no clicks.

Strategy 4.9 (*Click > TopOne NoClickEarlier QC*). For a ranking $(p_1, p_2, \ldots)$ that received no clicks followed (not necessarily immediately) by ranking $(p'_1, p'_2, \ldots)$ within the same query chain having clicks on ranks in C', extract a preference example $\text{rel}(p'_i) > \text{rel}(p_1)$ for all $i \in C'$.

Strategy 4.10 (*Click > TopTwo NoClickEarlier QC*). For a ranking $(p_1, p_2, \ldots)$ that received no clicks followed (not necessarily immediately) by ranking $(p'_1, p'_2, \ldots)$ within the same query chain having clicks on ranks in C', extract a preference example $\text{rel}(p'_i) > \text{rel}(p_1)$, and $\text{rel}(p'_i) > \text{rel}(p_2)$, for all $i \in C'$.

Strategy 4.9 produces the following set of features for our running example: $\text{rel}(p_{21}) > \text{rel}(p_{11})$, $\text{rel}(p_{23}) > \text{rel}(p_{11})$, $\text{rel}(p_{25}) > \text{rel}(p_{11})$, $\text{rel}(p_{32}) > \text{rel}(p_{11})$, $\text{rel}(p_{41}) > \text{rel}(p_{11})$. Additionally to the

previous features, Strategy 4.10 produces also rel(p_{21}) > rel(p_{12}), rel(p_{23}) > rel(p_{12}), rel(p_{25}) > rel(p_{12}), rel(p_{32}) > rel(p_{12}), rel(p_{41}) > rel(p_{12}).

The accuracy of the previous strategies "Click > TopOne NoClickEarlier QC" and "Click > TopTwo NoClickEarlier QC" suggests that users not only give negative feedback about the result set by not clicking on any link, but also that they learn from the result set how to formulate a better query. In particular, a user might discover an unanticipated ambiguity of the original query, which is avoided in a query reformulation. To capture the concept of a user trying to improve their queries within a chain of reformulations it has been considered how often the top result of a later query is more relevant than the top result of an earlier query.

Strategy 4.11 (*TopOne > TopOne Earlier QC*). For a ranking $(p_1, p_2, \ldots)$ that received no clicks followed (not necessarily immediately) by ranking $(p'_1, p'_2, \ldots)$ within the same query chain having clicks on ranks in C', extract a preference example rel(p'_i) > rel(p_1), and rel(p'_i) > rel(p_2), for all $i \in C'$.

Strategy 4.11 produces the following set of features for our running example: rel(p_{21}) > rel(p_{11}), rel(p_{31}) > rel(p_{11}), rel(p_{41}) > rel(p_{11}), rel(p_{31}) > rel(p_{21}), rel(p_{41}) > rel(p_{21}), rel(p_{41}) > rel(p_{31}).

Strategy	Features per Query	Normal (%)	Swapped (%)
Click > Skip Earlier QC	0.49	84.5 ± 16.4	71.7 ± 17.0
Last Click > Skip Earlier QC	0.33	77.3 ± 20.6	80.8 ± 20.2
Click > Click Earlier QC	0.30	61.9 ± 23.5	51.2 ± 17.1
Click > TopOne NoClickEarlier QC	0.35	86.4 ± 21.2	77.3 ± 15.1
Click > TopTwo NoClickEarlier QC	0.70	88.9 ± 12.9	80.0 ± 10.1
TopOne > TopOne Earlier QC	0.84	65.3 ± 15.2	68.2 ± 12.8

Table 4.5: Accuracy of the *Query Chain* strategies shown above for generating pairwise preferences from clicks within a single query. Feature-Per-Query shows the average number of features extracted per each query, the Swapped Column represents the experiment with the first two results swapped [116].

As in the non-QC methods, Table 4.5 shows the accuracy of the methods proposed for the Query Chains. In particular, it is evident that Strategy 4.10 – *Click > TopTwo NoClickEarlier QC* – produces the best results. Indeed, results of *Click > TopTwo NoClickEarlier QC* are better than those obtained by all of the non-QC strategies. Note that the average accuracy of this method is the same of the "Click > Skip Previous" one. The standard deviation, though, is smaller leading to a smaller number of incorrect rankings. Furthermore, 88.9% is close to the theoretical optimum of 89.5% corresponding to the Inter-judgements agreement shown in Table 4.4. For this reason considering the history of the queries submitted by the same user on the same topic, improve the estimate of the relevance of a page.

For any of the QC strategies discussed above to be applicable, it is necessary to specify the algorithm used to detect Query Chains, that is segmenting query submission histories of users into Query Chains automatically. Radlinski and Joachims [177] and Joachims *et al.* [116] devise the

following way of extracting Query Chains. They use a machine learning approach based on features like the *overlap of query words*, *overlap and similarity of the retrieved results*, and *time between queries*.

Anyway:

> "*it remains an open question whether this segmentation can be done equally accurately in a web search setting, and in how far the information need* drifts *within long query chains.*" [116]

For instance, rules 4.2, and 4.3 assume that when a user clicks on several results, it is because the first clicked results did not satisfy the information need. This assumption is not always true. For example, it may be correct for navigational queries but not necessarily true for informational [48] ones.

Getting back to how to use query log features to learn to rank, in [114] a formalization of the Information Retrieval problem is given in order to be able to state it as a machine learning problem. For a query q and a document collection $D = \{d_1, \ldots, d_m\}$, the optimal retrieval system aims at returning a ranking r^* that orders the documents in D according to their relevance to the query. Obviously the ordering r^* cannot be formalized specifically (otherwise it would be a *Sorting Problem*) therefore, usually, an IR system returns an ordering $r_{f(q)}$ that is obtained by sorting documents in D according to scores computed by a function f over the query q, i.e. $f(q)$. Formally, both r^*, and $r_{f(q)}$ are binary relations over $D \times D$ that fulfill the properties of a weak ordering (i.e. asymmetric, and negatively transitive). A relation r contains pairs (d_i, d_j) such that d_j is ranked higher than d_i, i.e. $d_i <_r d_j$. To optimize $f(q)$ in order to produce a ranking as close as possible to the optimal one r^*, we need to define the similarity between two orderings: r^* and $r_{f(q)}$. In the literature one of the most used metric to measure similarity between two ranked lists is the Kendall's τ distance metric [123]. Basically it consists of counting the number of concordant – P – and discordant – Q – pairs in r^* and $r_{f(q)}$. In a finite domain of m total documents in the collection, i.e. $|D| = m$, and the total number of pairs is, thus, $\binom{m}{2}$. Kendall's τ can be defined as

$$\tau(r_a, r_b) = \frac{P - Q}{P + Q} = 1 - \frac{2Q}{\binom{m}{2}}$$

Maximizing $\tau(r^*, r_{f(q)})$ is equivalent to minimize the average rank of relevant documents. Furthermore it is proven the following theorem relating the Kendall's τ with the Average Precision [25].

Theorem 4.1. Let $f(q)$ be a ranking function returning the ranking $r_{f(q)}$ for query q. Let R be the number of relevant documents, and let Q be the number of discordant pairs with respect to the optimal rank. The Average Precision AvgPrec of the scoring function f is bounded by

$$\text{AvgPrec}\left(r_{f(q)}\right) \geq \frac{1}{R}\left[Q + \binom{R+1}{2}\right]^{-1}\left(\sum_{i=1}^{R}\sqrt{i}\right)^2$$

The above argument shows that maximizing $\tau(r^*, r_{f(q)})$ is connected to improved retrieval quality in multiple frameworks.

Therefore, we are able to define the problem of learning a ranking function as an optimization problem.

***Learning a Ranking Function Problem* [114].** For a fixed and unknown distribution $\Pr(q, r^*)$ of queries and target rankings on a document collection D with m documents, the goal is to learn a retrieval function $f(q)$ for which the expected Kendall's τ

$$\tau_{\Pr}(f) = \int \tau(r^*, r_{f(q)}) d\Pr(q, r^*)$$

is maximal [114].

The learner we are seeking selects a ranking function f from a family of ranking function F maximizing the empirical expected τ on the training sample set S, an independently and identically distributed training sample set containing n queries q_i with their rankings r_i^*, $i = 1, ..., n$

$$\tau_S(f) = \frac{1}{n} \sum_{i=1}^{n} \tau\left(r_{f(q_i)}, r_i^*\right)$$

We consider a class of linear ranking functions satisfying

$$(d_i, d_j) \in f_{\vec{w}}(q) \Leftrightarrow \vec{w}\Phi(q, d_i) > \vec{w}\Phi(q, d_j)$$

where, $\vec{w}$ is a weight vector that is the one learnt by the learning algorithm, and $\Phi(q, d)$ is a mapping onto features describing the matching of query q and document d: much in the spirit of Fuhr [84] and Fuhr *et al.* [83]. By introducing (non-negative) slack variables $\xi_{i,j,k}$ we can formulate the problem as an optimization problem known as *Ranking SVM* [114]:

Definition 4.1. *Ranking SVM*

minimize: $V\left(\vec{w}, \vec{\xi}\right) = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k}$

subject to:

$$\begin{aligned}
&\forall (d_i, d_j) \in r_1^* : \vec{w}\left(\Phi(q_1, d_i) - \Phi(q_1, d_j)\right) \geq 1 - \xi_{i,j,1} \\
&\dots \\
&\forall (d_i, d_j) \in r_n^* : \vec{w}\left(\Phi(q_1, d_i) - \Phi(q_1, d_j)\right) \geq 1 - \xi_{i,j,1} \\
&\forall i \forall j \forall k \ : \ \xi_{i,j,k} \geq 0
\end{aligned}$$

In the definition above, C is a parameter used to allow the trade-off of margin size against training error. This problem can be solved using a SVM, and Joachims [114] shows how it can be extended to include also *non-linear* ranking functions.

The learnt ranking function is, then, used to sort the documents by their values of the *retrieval status value* $\text{rsv}(q, d_i) = \vec{w}\Phi(q, d_i)$.

Actually, there is a little trick to adopt in the implementation of the Ranking SVM problem. Since the whole feedback is not available for each query, i.e. we do not have ranking information for the whole collection, we must adapt the Ranking SVM to partial data by replacing r^* with the observed preferences r'. Given a training set S

$$(q_1, r_1'), (q_2, r_2'), \dots, (q_n, r_n')$$

with partial information about the target ranking, this results in the following problem

Definition 4.2. *Ranking SVM (partial)*
minimize: $V\left(\vec{w}, \vec{\xi}\right) = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k}$

subject to:

$$\forall (d_i, d_j) \in r'_1 : \vec{w}\left(\Phi(q_1, d_i) - \Phi(q_1, d_j)\right) \geq 1 - \xi_{i,j,1}$$
$$\ldots$$
$$\forall (d_i, d_j) \in r'_n : \vec{w}\left(\Phi(q_1, d_i) - \Phi(q_1, d_j)\right) \geq 1 - \xi_{i,j,1}$$
$$\forall i \forall j \forall k \ : \ \xi_{i,j,k} \geq 0$$

The resulting retrieval function is, thus, defined using the same SVM approach of the non-partial problem. The function chosen is the one that has the lowest number of discordant pairs with respect to the observed parts of the target ranking.

Radlinski and Joachims [177] and Joachims *et al.* [116] show some experimental results. In particular, Radlinski and Joachims [177] show that using Query Chains helps in improving the retrieval quality. Through a user study made on training data from the Cornell University Library's search engine they showed that 32% of people preferred the rankSVM performance trained over QC over a 20% of people preferring the non-rankSVM version of ranking (48% of people remained indifferent). Furthermore, 17% against 13% of users preferred the rankSVM using QC than the rankSVM not using QC (here, 70% of people remained indifferent).

Other approaches to learn to rank use different learning algorithms. *RankNet*, for instance, is said to be used by the Microsoft's Live search engine [139], it adopts a neural network approach to tackle the problem of learning a ranking function [51]. Several other approaches have been proposed during these last years: RankBoost [82], GBRank [244], LambdaRank [52], NetRank [3], just to name a few. An interesting source of information for this kind of algorithms is the Learning to Rank[11] workshop that, usually, makes his proceedings available online.

For readers interested in deepening their knowledge on learning to rank, a very interesting survey has been published in the same series of the present survey [143].

4.6 Query Spelling Correction

One of the neatest features of a search engine is the ability of "*magically*" detect we are mistyping queries. There is no magic, obviously, and the use of information on past queries is of utmost importance to infer *Spelling Corrections* in mistyped queries.

At a first glance, this may look pretty much similar to the problem of Query Suggestion. Indeed Query Spelling Correction is a little bit subtler. This is particularly true in web search engines where queries are composed by using terms drawn from a vocabulary of conversational words and people

[11] Google for "workshop learning to rank" for a list of URLs of past workshops' editions.

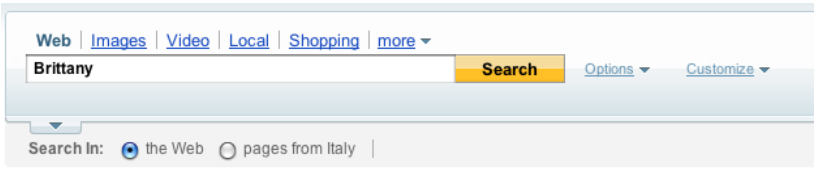


Fig. 4.8: The word Brittany representing the name of a person is correctly recognized as valid.

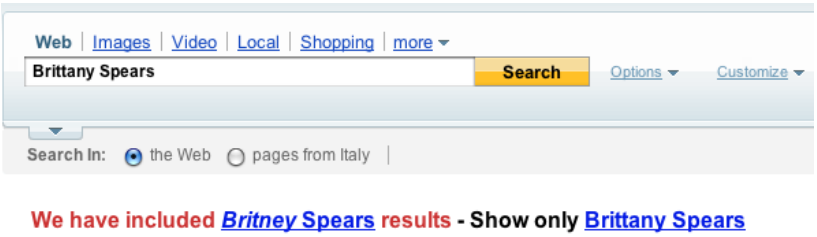


Fig. 4.9: An example of how a popular search engine helps a user that mistyped the word Britney in Brittany.

names (i.e. Brittany Spears[12]). Furthermore, contextual information are also extremely important. For instance, consider the two queries "*flash grdn*", and "*imperial grdn*". It is straightforward to correct the first occurrence of "*grdn*" with "*gordon*", whereas the second one with "*garden*". This simplicity derives from the term before "*flash gordon*" is a popular comics character, for instance. Query spelling correction is not free from errors. Someone looking for "*maurizio marin*" might not look for "*marino marin*" like, instead, it is suggested[13]. Figure 4.9 shows the correction presented for the query "*Brittany Spears*" into the query "*Britney Spears*".

Intuitively, Query Logs constitutes a very comprehensive knowledge base for building spelling correction models that have to be based on the actual usage of a language and not (only) on a prebuilt vocabulary of terms. Indeed, recently, there have been proposed works dealing with spelling correction using web search engines' query logs [67, 8, 136, 164, 60]. Note that the naïve approach of extracting from query logs all the queries whose frequencies are above a certain threshold and consider them valid is not correct. In fact, it can be the case that a misspelled query like "*britny spears*" is far more frequent than correctly spelled queries like "*relational operator*" or "*bayesian net*": shall "*britny spears*" be suggested as a possible correction in those cases?

Cucerzan and Brill [67] develop a very powerful technique to deal with query correction using information out of query logs. Authors of the paper very nicely introduce the problem formulation by reviewing the prior work and by iteratively refining the model to include the most interesting case of web search.

The most classical methodology followed for spelling correction dates back to 1964 when Dam-

[12] Note that we have mistyped the name Britney to demonstrate the subtleties of query spelling. The word Brittany is, in fact, recognized as correct by many spell checkers (see Figure 4.8) whereas the word Britney is not. This is true on our particular spell checker, at least.

[13] At least at the time this survey was written.

erau [71] published the seminal work on spelling correction. Basically the idea behind traditional spelling correction is the following. Use a lexicon/dictionary made up of ∼100K words. Flag words not found in lexicon as misspellings. Suggest lexicon words that are small edit distance[14] from unrecognized word. More formally, let Σ be the alphabet of a language and $L \subset \Sigma^*$ a broad coverage lexicon of the language. The definition given by Damerau [71] of lexicon-based spelling correction is:

Definition 4.3. Given an unknown word $w \in \Sigma^* \setminus L$, *lexicon-based spelling correction* finds $w' \in L$ such that $\text{dist}(w, w') = \min_{v \in L} \text{dist}(w, v)$.

Here, the function $\text{dist}(\cdot, \cdot)$ is a string distance function. Damerau [71] proposes to use the Edit distance to evaluate how similar two strings are. The major drawback of the problem formulation given in Definition 4.3 is that it does not consider the frequency of words in a language. Let us, then, refine the above formulation by thresholding the maximum distance allowed and to get the term with the maximum probability of occurrence. Formally:

Definition 4.4. Given an unknown word $w \in \Sigma^* \setminus L$, find $w' \in L$ such that $\text{dist}(w, w') \leq \delta$ and $P(w') = \max_{v \in L,\, \text{dist}(w,v) \leq \delta} P(v)$.

In this formulation, all distances are set to be δ at maximum, and within the terms at that maximum distance the most likely word is suggested. The most important fact is that prior probabilities are computed over a given language. Therefore, it allows the conditioning on the basis of a given language. Also this formulation is not free from drawbacks: it does not consider, for instance, the actual distances between each candidate and the input word. The following definition does consider that distance by conditioning the probability of a correction on the original spelling $P(v|w)$:

Definition 4.5. Given an unknown word $w \in \Sigma^* \setminus L$, find $w' \in L$ such that $\text{dist}(w, w') \leq \delta$ and $P(w'|w) = \max_{v \in L,\, \text{dist}(w,v) \leq \delta} P(v|w) = \max_{v \in L,\, \text{dist}(w,v) \leq \delta} P(w|v)\, P(v)$.

The skilled reader shall recognize the application of the Bayes' theorem to rewrite the objective function as $P(v|w) = \frac{P(w|v)P(v)}{P(w)}$ where $P(v)$ is the *language model*, and $P(w|v)$ is the *error model*. The term $P(w)$ can be omitted since it does not depend on v thus does not influence the computation of the minimum.

In general, the above three formulations consider words to be corrected in isolation. This means situations like the one shown above when the two queries "flash grdn", and "imperial grdn" had to be corrected in a different way are not taken into consideration. A formulation taking into consideration this issue of *contextual spelling correction* is the following:

Definition 4.6. Given a string $s \in \Sigma^*$, $s = c_l w c_r$, with $w \in \Sigma^* \setminus L$ and $c_l, c_r \in L^*$, find $w' \in L$ such that $\text{dist}(w, w') \leq \delta$ and $P(w'|c_l w c_r) = \max_{v \in L,\, \text{dist}(w,v) \leq \delta} P(v|c_l w c_r)$.

---

[14] Edit distance is usually defined as a measure over the number of characters that need to be changed, added, removed or transposed to convert one word to another.

That is, we are considering contextual information like the words preceding (c_l) and following (c_r) the given word to be corrected. The last formulation cannot consider query corrections where two *valid* words have to be concatenated into an out of lexicon word. For instance, the query "*robert louis steven son*" is composed by all valid queries, yet it is very likely that actually the correct query should have been "*robert louis stevenson*" despite the fact that both "*steven*", and "*son*" are two valid words.

The above observations lead to a very general formulation of the problem that is the following:

Definition 4.7. Given $s \in \Sigma^*$, find $s' \in \Sigma^*$ such that $\text{dist}(w, v) \leq \delta$ and $P(s'|s) = \max_{t \in \Sigma^*, \text{dist}(s,t) \leq \delta} P(t|s)$.

In the above definition it is important to remark that the formulation does not make use of any explicit lexicon of the language considered. In a sense, it is the query log induced language that matters and string probabilities are extracted from the query log itself.

Definition 4.7 is general and correct, yet it cannot be used to derive any algorithm to perform spelling corrections. First of all, it has to be noticed that query spelling correction can be formulated as an iterative process. Consider this query: "*roberl louis steven son*". The correct query formulation is, very likely, "*robert louis stevenson*" and the steps that can be followed to converge to the correct formulation are: "roberl louis steven son" $\rightarrow$ "*robert louis steven son*" $\rightarrow$ "*robert louis stevenson*". How to pass from string s to string s_1? Using the query log information to observe that relatively frequently s_1 appears in the log. Therefore, if s_0 is a misspelled query, the algorithm aims at using query log information to find a succession of queries $s_1, \ldots, s_n$ such that s_i into s_j $(0 \leq i < j \leq n)$, s_n is the correct spelling. Formally this leads to the following definition:

Definition 4.8. Given a string $s_0 \in \Sigma^*$, find a sequence $s_1, \ldots, s_n \in \Sigma^*$ such that for each $i \in 0..n-1$ there exist the decompositions $s_i = w_{i,0}^1 \ldots w_{i,0}^{l_i}, s_{i+1} = w_{i+1,1}^1 \ldots w_{i+1,1}^{l_i}$, where $w_{j,h}^k$ are words or groups of words such that $\text{dist}\left(w_{i,0}^k, w_{i+1,1}^k\right) \leq \delta$, $\forall i \in 0..n-1$, $\forall k \in 1..l_i$ and $P(s_{i+1}|s_i) = \max_{t \in \Sigma^*, \text{dist}(s_i,t) \leq \delta} P(t|s_i)$, $\forall i \in 0..n-1$, and $P(s_n|s_{n-1}) = \max_{t \in \Sigma^*, \text{dist}(s_n,t) \leq \delta} P(t|s_n)$.

A misspelled query that can be corrected by a method applying the above definition is, for example, s_0 ="*britenetspaer inconcert*" can be transformed into s_1 = "*britneyspears in concert*" and successively into s_2 = "*britney spears in concert*", and finally "*britney spears in concert*". Obviously, for the above method to work some assumptions have to be done. First of all we must fix a maximum number of tokens into which a single word can be split, Cucerzan and Brill [67] choose to split into bigrams at maximum. Furthermore, it is essential to such an approach to work correctly that query logs adhere to three properties:

(1) words in the query logs are misspelled in various ways, from relatively easy-to-correct misspellings to very-difficult-to-correct ones, that make the user's intent almost impossible to recognize;
(2) the less difficult to correct a misspelling is the more frequent it is;
(3) the correct spellings tend to be more frequent than misspellings.

Cucerzan and Brill [67] implement a very nice and efficient algorithm to correct misspelled queries. Basically, what they do is to explore through a viterbi search algorithm the space of all possible corrections selecting from time to time the most likely correction out of a trusted lexicon of words and a lexicon of words built over a query log. The algorithm is made efficient since it does not allow simultaneous correction of two adjacent words. Therefore, the query "*log wood*" would not be corrected into "*dog food*" by mistake only because the first is less frequent than the second in the query log.

Cucerzan and Brill [67] also propose a quite accurate evaluation of the effectiveness of the method they propose. The evaluation has been performed over a set of 1044 unique and randomly sampled queries from a daily query log, which were annotated by two annotators whose inter-agreement rate was 91.3%. For those queries considered misspellings the annotators provided also corrections. The overall precision of the system was 81.8%, that is in the 81.8% of the cases the system either classified a query as valid, or recognized a misspelled query and proposed a valid correction. The first case occurred in the 84.8% of the cases, whereas in the 67.2% of the cases the system either did not recognize a misspelling or it proposed an invalid correction.

Something that has not very deeply studied by authors of the work is the error model, that is the probability distribution $P(s|t)$ of misspelling the word s with t. This important issue, crucial for letting the spell-checking algorithm to work with high precision, is thoroughly studied by Ahmad and Kondrak [8]. Basically, they run an Expectation-Maximization (EM) algorithm to update iteratively the error model and to detect "frequent" substitution patterns. These frequent substitution patterns are then plugged into the error model to boost edit distance computations and to make that more effective to correct errors typical in query logs. To enhance the precision of these methods based on statistics from query logs, Chen *et al.* [60] have recently proposed to adopt information from target pages. This way, a richer text repository is available to be able to detect a greater number of patterns to be "injected" into the edit distance computation function.

4.7 Summary

One of the possible uses of knowledge mined from query logs is to enhance the effectiveness of the search engine. By effectiveness we mean the capability of the search engine to answer with the best possible results to the query issued by a *particular* user. That is, we aim at presenting each user the most suitable possible list of results for his specific needs. This technique, known as *personalization*, is one of the major topics presented in this chapter.

When a query is badly formulated, or user is too generic or too specific, *query suggestion* and *query expansion* are the other two techniques that are used to improve the search experience. The former is an explicit help request from the search engine to the user. That is, the user is presented with a list of query suggestions among which we aim at including queries steering the user towards his specific information need. The latter is an implicit technique used by the search engine to modify (by adding search terms) the query in order to make it more expressive.

Learning to rank is used to "*learn*" *static scores*, i.e. *query-independent*, for web pages. Some of the most recently proposed techniques make use of information about how users click on results for a query. One of the most interesting claims (that has been empirically evaluated) is that, instead of considering a click as a distinguishing sign of importance, it is more effective to consider the relative clicking order among results. A click on the i-th result demote all the previous results, that

is clicks are considered as a sort of preference vote.

The last part of the present chapter presented the very recent literature about how search engines exploit information about queries submitted in the past to spell-check (and correct) submitted queries. This prevent them to correct the query "*Brittany*" (Figure 4.8) and allow to correctly detect the spelling error in the query "*Brittany Spears*" (Figure 4.9).

5

Enhancing Efficiency of Search Systems

Quoting a passage of Baeza-Yates *et al.* [14]:

> "*The scale and complexity of web search engines, as well as the volume of queries submitted every day by users, make query logs a critical source of information to optimize precision of results and efficiency of different parts of search engines. Features such as the query distribution, the arrival time of each query, the results that users click on, are a few possible examples of information extracted form query logs. The important question to consider is : can we use, exploit, or transform this information to enable partitioning the document collection and routing queries more efficiently and effectively in distributed web search engines?*"

This means that dealing with efficiency in web search engines is as important as it is dealing with user preferences and feedback to enhance effectiveness. Literature works show that usage patterns in web search engine logs can be exploited to design effective methods for enhancing efficiency in different directions.

In these last years, the majority of research studied how to exploit usage information to make *caching*, and *resource allocation*, effective in highly distributed and parallel search systems. In addition, there is a novel trend to exploit usage information from query logs also for Crawling [57] purposes. In particular two recently published papers deal with a novel user-centric notion of repository quality [160], and a novel prioritization scheme for page crawling ordering based on usage information [161]. We do not enter too much into details of this novel research activity, still we strongly encourage readers interested in crawling to look through those papers since they represent a nice view point exploiting query log information to enhance the crawling process.

5.1 Caching

Caching is the main mean with which systems exploit memory hierarchies. There is a whole body of literature on systems where caching is extensively studied [207, 101]. Furthermore, in web ar-

chitectures caching is exploited to enhance the user's browsing experience, and to reduce network congestion [167].

Caching is a well-known concept in systems with multiple tiers of storage. For simplicity, consider a system storing N objects in relatively slow memory, that also has a smaller but faster memory buffer of capacity k which can store copies of k of the N objects ($N >> k$). This fast memory buffer is called the *cache*. The storage system is presented with a continuous stream of queries, each requesting one of the N objects. If the object is stored in the cache, a *cache hit* occurs and the object is quickly retrieved. Otherwise, a *cache miss* occurs, and the object is retrieved from the slower memory. At this point, the storage system can opt to save the newly retrieved object in the cache. When the cache is full (i.e. already contains k objects), this entails *evicting* some currently cached object. Such decisions are handled by a *replacement policy*, whose goal is to maximize the *cache hit ratio* (or *rate*) - the proportion of queries resulting in cache hits.

Often, access patterns to objects, as found in query streams, are temporally correlated. For example, object y might often be requested shortly after object x has been requested. This motivates *prefetching* - the storage system can opt to retrieve and cache y upon encountering a query for x, anticipating the probable future query for y.

Caching in web search engine [135] is, basically, a matter of stocking either results, partial results, or raw posting lists, into a smaller, and faster to lookup, buffer memory. Usually, in real systems, caching uses a combination of the three above kinds, and the final system appears as in Figure 5.1. The figure is only a slight modification of the architecture depicted in Figure 1.2 at page 4 and shows that the placement of this cache modules within the architecture does not require massive modifications.

The right to decide what results are to be kept in cache is acknowledged to the *caching policy*. In case of the cache running out of space, the caching policy is responsible for expunging a result in favor of another retained to be more likely to be requested in future.

In web search engines caching has been studied since 2000 when Markatos presented the (probably) first work specifically targeted on exploiting caching possibilities in web search engines [75].

The setting with respect to search result caches in web search engines consists of result pages of *search queries* to be cached. A search query is defined as a triplet $q = (qs, from, n)$ where qs is a query string made up of *query terms*, *from* denotes the relevance score of the first result requested, and n denotes the number of requested results. The result page corresponding to q would contain the results whose relevance score with respect to qs are $from, from + 1, \ldots, from + n - 1$. The value of n is typically 10. Search engines set aside some storage to cache such result pages. Indeed, a search engine is not a typical two-tiered storage structure. Results not found in the cache, in fact, are not stored in slower storage but rather need to be generated through the query evaluation process of the search engine.

Prefetching of search results occurs when the engine computes and caches $p \cdot n$ results for the query $(qs, from, n)$, with p being some small integer constant, in anticipation of follow-up queries requesting additional result pages for the same query string.

Caching (often with the name of *paging*) has also been studied theoretically in the past, within the formal context of *Competitive Analysis*, by many researchers. The seminal paper of Sleator and Tarjan [206] showed that the LRU (Least Recently Used) paging strategy is optimal with respect to a competitive point of view.

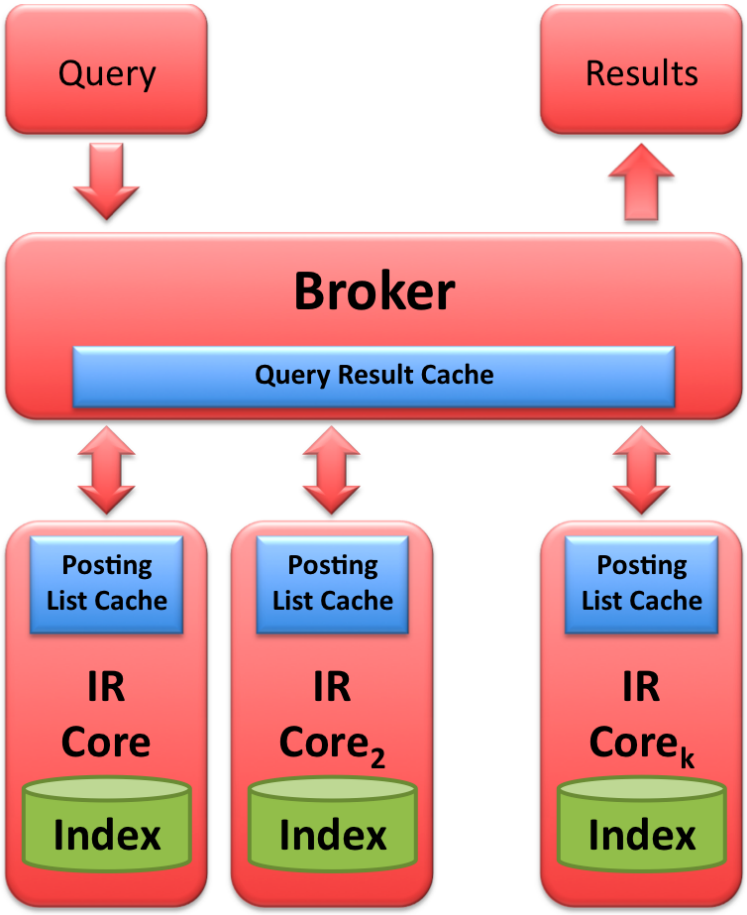


Fig. 5.1: Caching module placement within the typical structure of a web search engine shown in Figure 1.3.

Projected onto search engines, competitive analysis has been performed by Lempel and Moran [133] to show that it is possible to design an online paging scheme, tailored to search engine query workloads, that incurs in an expected number of cache misses no greater that 4 times the expected number of misses that any online caching algorithm would experience. In the following, we are going to present some of the most notably results in web search engine caching technology at all levels.

5.1.1 Caching and Prefetching of Query Results

We already showed in Query Nature chapter that Power-law arises in the distributions regulating usage patterns in real-world web search engines: TodoBR [188], Yahoo! [15], Tiscali [76], and so on and so forth.

Caching in web search engines immediately recalls the storing of *results* of previously computed queries in a *faster* memory area. This kind of caching is know as: *caching of search results*. A good deal of research has been (and it is currently) conducted on how efficiently manage the (small)

space dedicated to the cache in an optimal way. Obviously, caching policies depends heavily on how requests are distributed. The idea behind the ideal (and theoretically optimal) Belady's OPT cache policy: "*in case of miss and cache full, replace the page that will be accessed farthest in the future*" [38]. Unfortunately, this would require the cache to be "*Clairvoyant*" that is, it would require the cache to know in advance the rest of the query stream. Therefore, caching policies can only aim at approximating OPT as better as possible.

Roughly speaking, locality of accesses means that queries repeat themselves within (relatively) small periods of time, and caching should exploit this kind of regularities to keep copies of "*likely-to-be-accessed-in-the-future*" queries.

The caching design space is two-dimensional: choosing the caching policy in order to increase as much as possible the *hit-ratio* [75, 132, 134, 76, 202, 24], and optimizing the architecture of the caching system [188, 26, 140, 76, 202, 15] to improve *response time* and *throughput* (number of queries per second) of the search engine. Both dimensions are important since higher hit-ratios often correspond to lower response times and higher throughput.

Markatos [75] describes different, state-of-the-art, caching policies and compares the hit-ratio obtained on an Excite log. The paper does not propose any explicit policy tailored to specific statistical properties of a query log. Furthermore, it does not consider the possibility of exploiting a prefetching strategy in order to prepare the cache to answer possible requests for following pages. Nevertheless, the research shows the feasibility of caching in search engines.

The four different policies tested on a query stream coming out of the Excite query logs were: *LRU*, *FBR*, *LRU/2*, *SLRU*.

LRU is among the most famous algorithms for cache replacement policies. LRU, also known as *move-to-front*, works by using a First-In First-Out (FIFO) queue to store query results. When the submitted query is present in the cache buffer it is shifted to the front of the queue. If the query is not in cache, then it is forwarded to the underlying search level and when the results are back, the new query entry is pushed to the top of the queue. If the cache was full the last element of the cache is evicted before the new element is put on top.

The FBR replacement algorithm [182] maintains the LRU ordering of all blocks in the cache, but replaces the block in the cache that is least frequently used (LFU) and in case of more than one, the LRU.

LRU/2 is a replacement policy that opts for evicting queries whose second-to-last access is least recent among all penultimate accesses [154].

SLRU combines both recency and frequency of access when making a replacement decision. In [122] it is stated "An SLRU cache is divided into two segments, a probationary segment and a protected segment. Lines in each segment are ordered from the most to the least recently accessed. Data from misses is added to the cache at the most recently accessed end of the probationary segment. Hits are removed from wherever they currently reside and added to the most recently accessed end of the protected segment. Lines in the protected segment have thus been accessed at least twice. The protected segment is finite, so migration of a line from the probationary segment to the protected segment may force the migration of the LRU line in the protected segment to the most recently used (MRU) end of the probationary segment, giving this line another chance to be accessed before being replaced. The size limit on the protected segment is an SLRU parameter that varies according to the I/O workload patterns. Whenever data must be discarded from the cache,

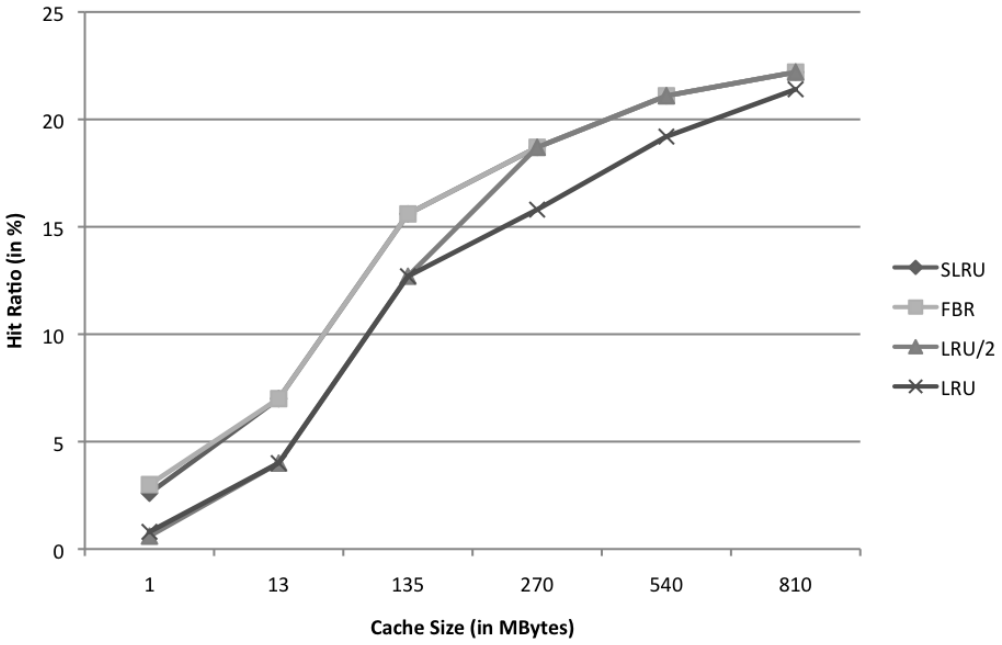


Fig. 5.2: Comparison of different policies hit-ratios on varying the cache size (in MB). Results are computed over queries in the Excite log [75].

lines are obtained from the LRU end of the probationary segment."

Figure 5.2 reports result obtained by the four policies over the whole log by varying the cache size. The different replacement policies behaves differently depending on the cache size. For very small cache sizes (<100 Mbytes) FBR performs better than LRU/2. For large caches LRU/2 performs a little better than FBR. However, in all cases, SLRU performs very close to the best of FBR and LRU/2. Suggesting that SLRU might be the policy of choice.

Furthermore, since LRU and SLRU seem to be the best policies in terms of management complexity and hit-ratio, from now on we restrict our discussion mainly on these two policies. Furthermore, to better highlight the differences of LRU and SLRU policies and to analyze the performance of the two policies for very large caches, the histogram in Figure 5.3 compares LRU and SLRU for the same Excite log (2.2GB of data) on caches whose size varies between 1MB and 2,160MB.

The first observation is that both LRU and SLRU perform roughly the same for caches bigger than 1GB. Caches of that size, anyway, are not of significant interest for such a small log. Instead, it is interesting to notice the slight advantage of SLRU over LRU for mid-sized caches. For example, for a 270MB cache, SLRU scored a 18.8% against a 16.4% of LRU's hit-ratio.

The work of Markatos only proposes the analysis of *existing* caching policies over a real-world search engine workload. Even though Markatos timely address the problem of result caching in web search engines, it does not present any new policies specifically tailored on such a workload.

The first two policies that have been proposed and that exploit the characteristics of search engine query logs are: Lempel and Moran's *PDC* (Probabilistic Driven Caching) [132], and Fagni *et al. SDC* (Static Dynamic Caching) [76]. Furthermore, incidentally both papers introduce the

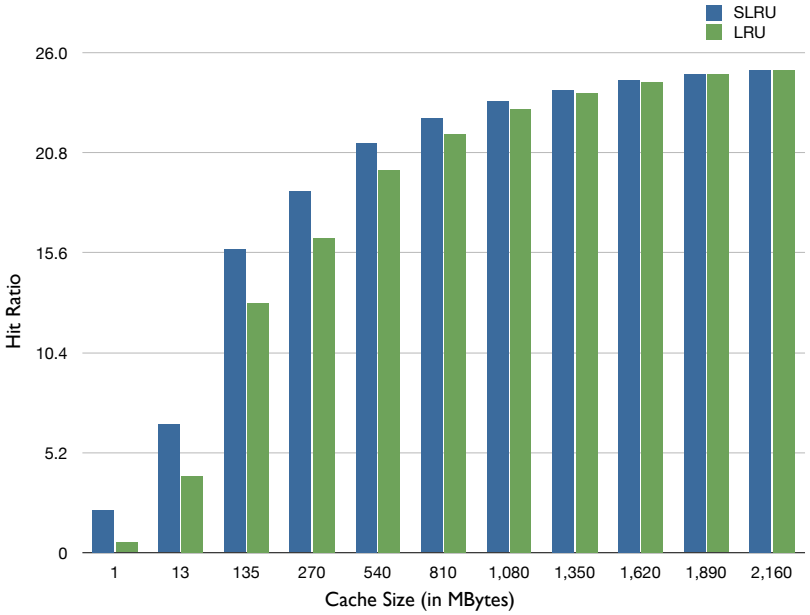


Fig. 5.3: SLRU vs. LRU in Result Caching: Excite log [75].

concept of prefetching in web search engine result caches. Prefetching [132, 134, 76] is another possibility for increasing the hit-ratio of result caching policies. As said briefly in the initial part of this section, it, basically, consists of anticipating user requests by caching not only the result page requested but also the successive p (where p is the prefetching parameter). Information extracted from query logs can be used to tune finely the prefetching policy as shown by Fagni *et al.* [76].

The idea behind *PDC* is to associate a probability distribution to all the possible query that can be submitted to a WSE. The distribution is built over the statistics computed on the previously submitted queries. For all the queries that have not previously seen, the distribution function evaluates to zero. This probability distribution is used to compute a priority value that is exploited to maintain an importance ordering among the entries of the cache. In practice, the higher the probability of a query to be submitted the higher it is ranked within the cache once it is actually appeared. Indeed, the probability distribution is used only in the case of queries requesting the pages subsequent the first. For the first page of results a simple *SLRU* policy is used. Note that *PDC*, also consider a model of users' behavior. Within this model, a query-session starts with a request to the WSE. At this point two ways are possible: he can either submit a *follow-up* query (i.e. a query requesting the successive page of results), or he can give up and possibly start a new session by submitting a different query. A session is considered over, if no follow-up query appears within τ seconds. This model is respected in *PDC* by demoting the priorities of the entries of the cache referring to the queries submitted more than τ seconds ago. Note that the caching policy for the queries requesting the second, and over, page of results are ordered following a priority computed using the statistical data available. A priority queue is used to keep $2^{\text{nd}}+$ page of results sorted according to their priority. They evict from the section the entry which has the lower priority only if the ready-to-enter query has a priority greater than that. Conversely, the queries referring

to the first page of results are managed by a separate *SLRU* cache. The results on a query log of Altavista containing queries submitted during a week of 2001, are very good. Using a cache of $256,000$[1] elements using *PDC* and prefetching 10 pages of results, the authors obtained a hit-ratio of about $53,5\%$. For a more comprehensive picture, Figure 5.4 from the paper shows a comparison of PDC with LRU and SLRU on varying cache size and probationary segments size. Unfortunately the policy seems to be quite expensive in terms of time-to-serve for each request (in particular those causing a cache miss). Due to the priority queue used to keep queries sorted, an amortized complexity of $O(n \log n)$ dominates the overall complexity.

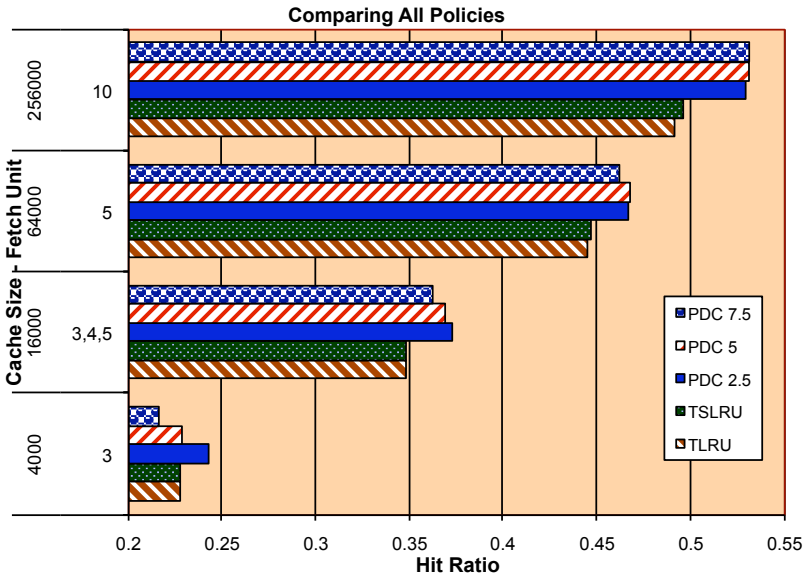


Fig. 5.4: Comparison of PDC with SLRU and LRU on varying cache size and the probationary segment size of SLRU [132].

Interestingly, as far as we are aware of, PDC is the first caching policy which may not apply the eviction policy in case of full cache and miss. In fact, the result currently considered may not be stored in cache if all the priorities of the current entries are higher.

Also the *SDC* policy, proposed by Fagni *et al.* [76] is an effective exploitation of historical usage data. As in PDC, SDC integrates both caching and prefetching at the same time. *SDC* is the acronym of Static Dynamic Caching since it basically integrates two types of caching: static, and dynamic.

Static cache was previously analyzed by Markatos [75] where it is shown that a static-only cache of query results badly performs on the tested log. The merit of SDC, instead, is to have mixed the two concept of Static and Dynamic caching trying to balance the benefits in terms of capturing frequent queries by means of the Static caching, and recent queries, by means of the Dynamic policy.

[1] Differently from Markatos' paper, results are expressed as number of entries. Estimating an entry of size 4 KBytes, $256,000$ elements correspond, roughly, to 1 GBytes of memory.

In the static cache, the set of the *most-frequently-submitted-in-the-past* queries is kept. The dynamic cache is a cache managed through a traditional replacement policy. Differently from PDC, SDC management complexity is constant, i.e. $O(1)$, whenever a constant policy is used in the dynamic section. PDC, instead, exhibit a $O(\log k)$ amortized management complexity. Furthermore, SDC introduces a novel kind of prefetching that exploits a particular and peculiar characteristics of web search users behavior: *Adaptive Prefetching*. It has been observed that when a user go through the i-th ($i \geq 2$) page of results then he will, with high probability, explore also the $i+1$-th page. Therefore, adaptive prefetching simply consists of performing prefetching whenever a request refers to the i-th ($i \geq 2$) result pages. That is prefetching is performed only in case of requests for the second, or greater, page of results.

Figure 5.5 shows that SDC resulted to be superior on hit-ratios achieved by a search system with a traditional completely dynamic policy. For example, on a cache table containing 256,000 results from the Altavista log, the hit-ratio of a pure LRU is about 32%, while SDC with LRU as the dynamic policy on the 40% of the cache size (i.e. about 100,000 results) obtains a hit-ratio of about 55% when a prefetching factor of 10 is used.

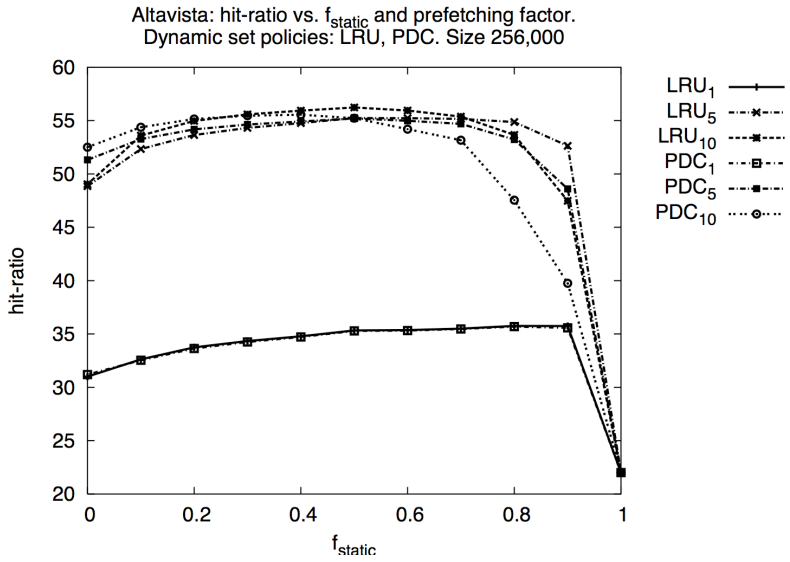


Fig. 5.5: Comparison of SDC adopting LRU and PDC as dynamic policies on varying static cache size and prefetching factor. The case of $f_{static} = 0$ corresponds to the pure (all dynamic) policy [76].

In their paper Fagni *et al.* [76] claim the superiority of SDC being due to the way it exploits the power law in the query log. Queries that frequently appear, not necessarily present the recency property (i.e. the same frequent query might be submitted after an arbitrary large number of distinct queries). If this assumption is true, and if the LRU queue is not large enough, then some frequent query could be evicted before they will be requested again[2]. To assess this, Figures 5.6a, and 5.6b report the cumulative number of occurrences of each distance, measured as the number of distinct

[2] Cache policies not affected by this problem are said to be *scan-resistant*. LRU is not scan-resistant

queries received by AltaVista and Yahoo! in the interval between two successive submissions of each frequent query [76, 15]. From the two figures in 5.6 we can conclude that even if we set the size of a LRU cache to a relatively large number of entries, the miss rate results to be high anyway.

Policies exploiting historical usage information like PDC or SDC might suffer of data model staling. The model built over a period might be not valid anymore in the next period. Baeza-Yates *et al.* [15, 16] use a static cache containing the 128,000 most frequent queries from a Yahoo! log has been used to test the hit-ratio trends on an hourly basis. Figure 5.7 shows that the hit-ratio is quit stable (ranging from 0.25 to 0.35) within a period of at least a week (for this particular query log, obviously). Indeed, there is a slight downward trend from left to right indicating a very limited entry staling problem. Also, there is a clear periodic trend in the plot indicating that for certain kind of queries, repetitions are more frequent during certain times of the day. These two observation need a more careful analysis, that indeed has not been done in the paper by Baeza-Yates *et al.* [15, 16], that could highlight fine tuned caching policies for different periods of the day (or year).

Recently Baeza-Yates *et al.* [15, 16] showed a caching algorithm exploiting an admission policy to prevent infrequent queries from taking space of more frequent queries in the cache [24]. The admission policy checks query features like, for instance, length in characters, in words, etc., and decides whether considering or not the query for being cached. Results have shown the superiority of the approach over SDC from a hit-ratio point of view. Unfortunately, the computational complexity of the policy is high, thus (likely), jeopardizing the benefits of a higher hit ratio from a throughput point of view.

Caching of Posting Lists

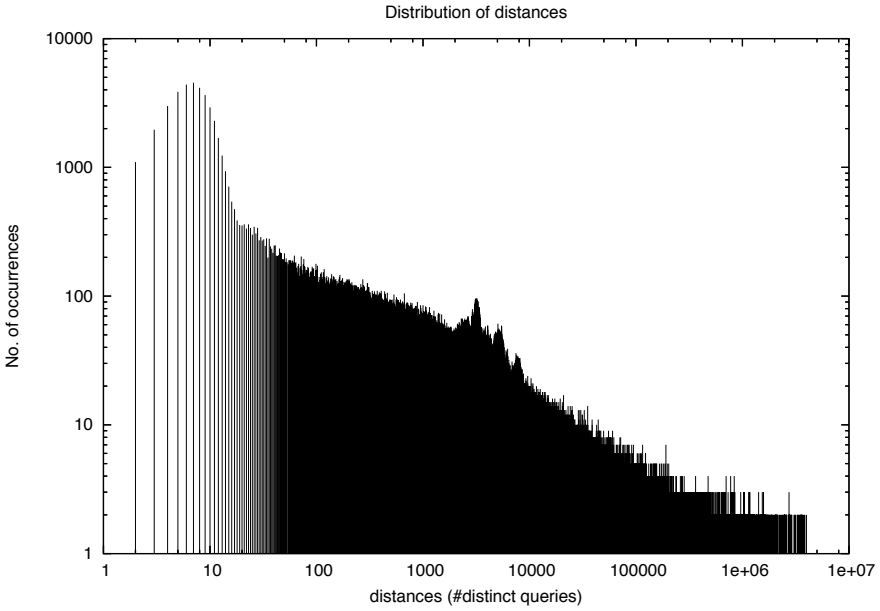
As said above, caching of query results is, most likely, the first thing coming to your mind when speaking about caching in web search engines. Posting list caching also exists , and it is as important as result caching.

As it has been seen in the Introduction, posting lists are used during the computation of query results. In less recent search systems, posting lists were thought to be stored on disk. In more modern web search systems, anyway, due to partitioning of data in large scale distributed search engines, it is very likely that the entire posting lists would be stored on memory. In both cases, anyway, having a more compact and faster structure storing frequently accessed lists might be of help for improving query answering time. This is the idea behind posting list caching: *storing frequently accessed lists to reduce delays due to list seek and retrieval operations.*

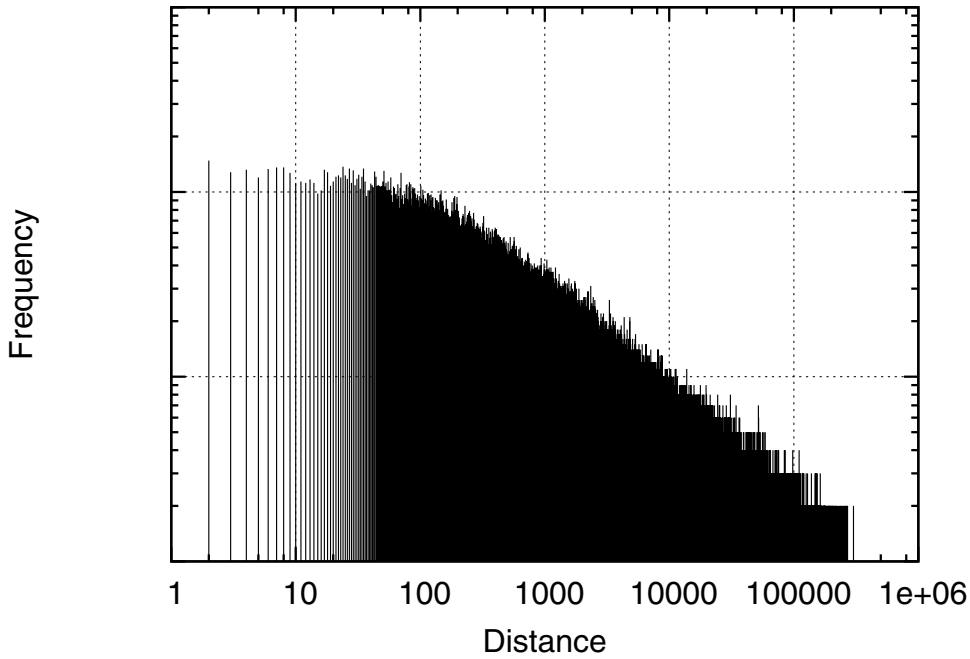
Furthermore, posting list caching and query result caching are not exclusive: if both used "*cum grano salis*", overall performance might end up being sensitively improved [15, 16].

Posting list caching has not received a lot of attention in the past, yet in these papers [188, 26, 140, 15], it has been shown to be an effective way to increase an Index server performance.

Basically, the potential effectiveness of posting list caching comes from the high recurrence rate of a few query terms. Think for instance to queries like "*britney spears*", "*britney spears scandal*", "*britney spears songs*", "*britney spears home page*", etc. They all share the common terms "*britney*" and "*spears*" and caching those lists will save retrieval time when processing queries in the examples. It has been shown that a high percentage of query terms are submitted repeatedly. Therefore, at least in theory, an infinite posting list cache would obtain a very high hit-ratio. For instance, Baeza-



(a) Altavista (from [76]).



(b) Yahoo! (from [15]).

Fig. 5.6: Cumulative number of occurrences of each distance, measured as the number of distinct queries received by AltaVista and Yahoo! in the interval between two successive submissions of each frequent query.

Yates *et al.* [15, 16] showed that a posting list cache using LRU as the replacement policy would reach a terrific hit-ratio of more than 90% for relatively large caches.

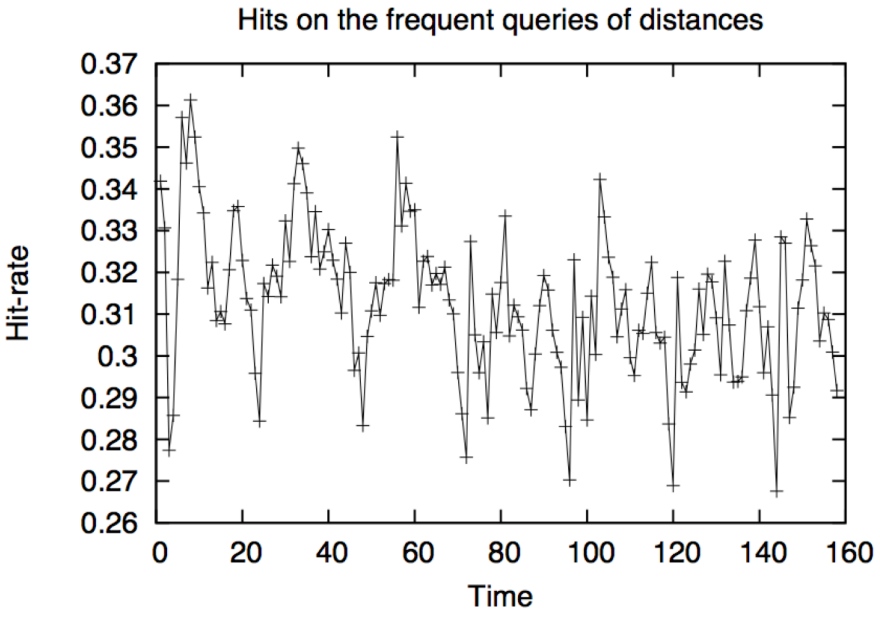


Fig. 5.7: Hourly hit-ratio for a static cache holding 128,000 answers during the period of a week. Figures drawn from a Yahoo! query log [15].

Again, as in the case of query results, traditional caching policies can still be used for posting lists as well. Anyway, more sophisticated policies can be devised.

In fact, caching posting lists is fundamentally different from caching query results because posting lists are of variable lengths, whereas query results are of fixed-size. For example, the posting list of a very common term is extremely longer than a list referring to an uncommon term like, for instance, a typo. Starting from this observation successful posting cache policies consider also the size of the posting lists among the features used to decide upon evictions. Baeza-Yates *et al.* [15, 16] authors consider both *dynamic* and *static* caching. For dynamic caching, they use two well-known policies, LRU and LFU, as well as a modified algorithm that takes posting-list size into account.

Before discussing the static caching strategies, let us recall some notation used by authors in the original paper: let $f_{q(t)}$ denote the query-term frequency of a term t, that is, the number of queries containing t in the query log, and $f_{d(t)}$ to denote the document frequency of t, that is, the number of documents in the collection in which the term t appears.

The first strategy considered, is the algorithm proposed by Baeza-Yates and Saint-Jean [26], which consists in selecting the posting lists of the terms with the highest query-term frequencies $f_{q(t)}$. This algorithm is called Q_{tf}. Interestingly, there is a trade-off between $f_{q(t)}$ and $f_{d(t)}$. Terms with high $f_{q(t)}$ are useful to keep in the cache because they are queried often. On the other hand, terms with high $f_{d(t)}$ are not good candidates because they correspond to long posting lists and consume a substantial amount of space. In fact, the problem of selecting the best posting lists for the static cache can be seen as a standard *Knapsack* [54] problem: given a knapsack of fixed capacity, and a set of n items, such as the i-th item has value c_i and size s_i, select the set of items that fit in the knapsack and maximize the overall value. In our case, "value" corresponds to $f_{q(t)}$

and "size" corresponds to $f_{d(t)}$. Therefore, a simple greedy algorithm for the knapsack problem is employed: select the posting lists of the terms with the highest values of the ratio $\frac{f_{q(t)}}{f_{d(t)}}$. This algorithm is called $Q_{tf}D_f$.

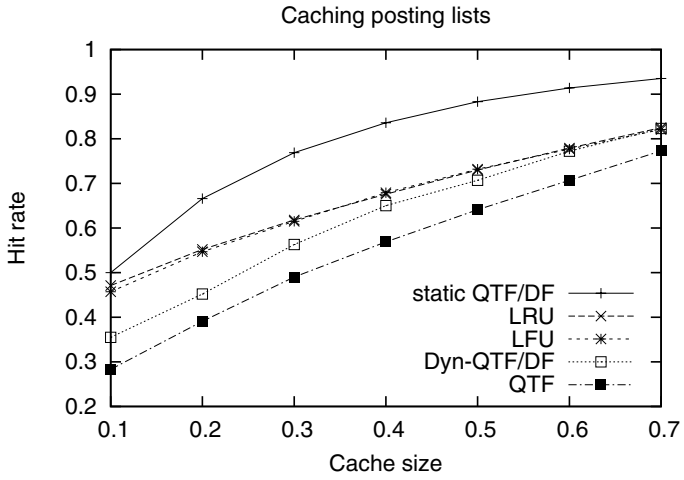


Fig. 5.8: Hit rate of different strategies for caching posting lists on queries in the Yahoo! log [15]. Cache size is expressed as the percentage of total memory used to store the whole index in memory.

Two different variations of $Q_{tf}D_f$ has been tested: static $Q_{tf}D_f$, and dynamic $Q_{tf}D_f$. In static $Q_{tf}D_f$ a static cache has been filled in with the knapsack-like algorithm, whereas in dynamic $Q_{tf}D_f$ the entry with the lowest $\frac{f_{q(t)}}{f_{d(t)}}$ is evicted, in case of cache-miss and cache full.

Results, in terms of hit-ratio, of different caching policies have been reported in Figure 5.8. The cache size, in this experiment, is measured as a fraction of the total space required to store the posting lists of all terms. The most important observation from the results is that the static $Q_{tf}D_f$ algorithm has a better hit rate than all the dynamic algorithms. An important benefit that static cache has, is that it requires no eviction and it is hence more efficient when evaluating queries. However, if the characteristics of the query traffic change frequently over time, then it requires re-populating the cache often or there will be a significant impact on hit rate.

To estimate the impacts of this sort of *topic shift* on query logs, Figure 5.9 reports measures on the effect on the $Q_{tf}D_f$ algorithm of the changes in a 15-week Yahoo! query log.

The query term frequencies is computed over the whole stream in order to select which terms to cache, and then compute the hit-ratio on the whole query stream. Obviously, since this computation assumes perfect knowledge of the query term frequencies, this hit-ratio is to be considered as an upper bound to the maximum hit-ratio attainable. A realistic scenario is simulated using the first 6(3) weeks of the query stream for computing query term frequencies and the following 9(12) weeks to estimate the hit-ratio. As Figure 5.9 shows, the hit-ratio decreases by less than 2% from the upper bound. Therefore, the static $Q_{tf}D_f$ strategy can be considered as a very good approximation of the behavior of the optimal policy for a long time.

Caching is not just a matter of improving hit-ratio of its policy. Indeed, as it has also been

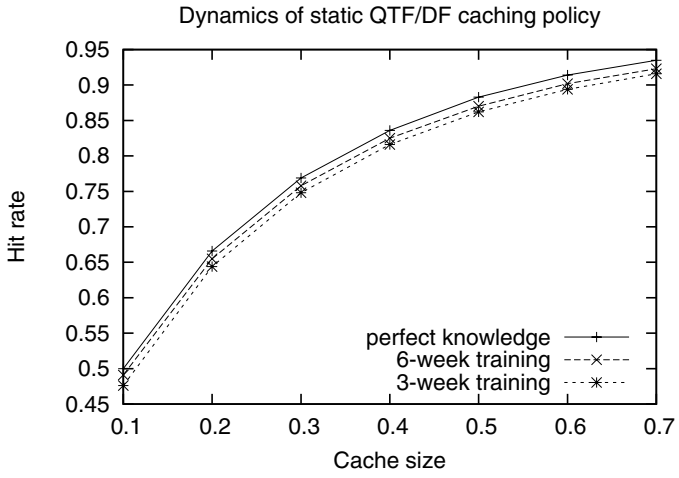


Fig. 5.9: Impact of distribution changes on the static caching of posting lists [15]. Cache size is expressed as the percentage of total memory used to store the whole index in memory.

pointed out and shown by Lempel and Moran [133], optimal policies for graph based workload models are possible at the cost of a linear time in the number of cache entries. Obviously, a linear time cache management is useless since it is prohibitively expensive. PDC exploits a reduced model that allows a logarithmic, i.e. $O(\log k)$, management policy. A high management cost may jeopardize the benefit of a high hit-ratio. Furthermore, in a real setting on the same machine many different index server instances run in parallel (via multithreading) and it is inconceivable to think of several *private* copies of the cache for every index server. While a shared cache reduces the space occupancy, it introduces the need of a regulated concurrent access to the cache structure by each thread, i.e. a spin-lock around the cache. Therefore, each access implies: lock acquisition, cache management (in case of miss the management time is higher due to the querying phase), lock release. The more efficient the cache management phase, the higher the scalability of the querying system as more concurrent tasks can run in parallel.

Therefore, having a static caching policy is important since, being the buffer read-only, it does not require any management operation and therefore, no lock is needed around the shared cache. This impacts heavily on the cache performance. Fagni *et al.* [76] assessed the throughput of a SDC cache under two different conditions: a lock around all the cache, a lock only on the dynamic set. Note that this last setting is the one which realistically should be used to manage concurrent accesses to a SDC cache.

Figure 5.10 reports the results of some of the tests conducted. In particular, the figure plots, for $f_{static} = 0.6$ and no prefetching, the throughput of an SDC caching system (i.e., the number of queries answered per second) as a function of the number of concurrent threads contemporary and concurrently accessing the cache. The two curves show the throughput of a system when each thread accesses in a critical section either the whole cache (dashed line) or just the Dynamic Set (solid line). Note that locking the whole cache is exactly the mandatory behavior of threads when accessing a purely dynamic cache (i.e., $f_{static} = 0$). Throughput was measured by considering that

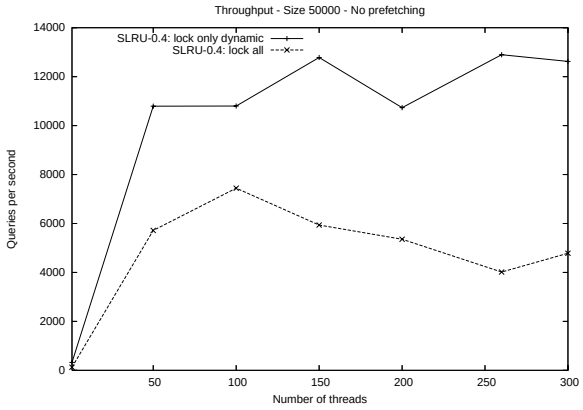


Fig. 5.10: Throughput of a caching system (in number of queries served per second) for $f_{static} = 0.6$ as a function of the number of concurrent threads used and different locking policies [76].

a large bunch of $500,000$ queries (from the *Tiscali* log) arrives in a burst. The size of the cache was $50,000$ blocks (quite small, though), while the replacement policy considered was *SLRU*.

Therefore, the presence of the Static Set, which does not need to be accessed using a mutex, permits to approximately double the number of queries served per second. Moreover, the caching system does not only provides high throughput but can also sustain a large number of concurrent queries. Performance starts degrading only when more than 200 queries are served concurrently.

Obviously the same argument holds also in case of posting list caching.

5.2 Index Partitioning and Querying in Distributed Web search Systems

As it have been pointed out by Baeza-Yates *et al.* [14] data distribution is one of the most important aspect to optimize in a modern distributed web search system. Engineering a fully distributed index server can be as easy as just randomly spreading data on the servers, or can be tricky as carefully partitioning data into topically consistent document partitions. Moreover, a still open problem is to devise whether document partitioning, or term partitioning is the index partitioning method of choice.

In this paragraph, we revise the problem of selecting a collection in a distributed information retrieval system. Then, we analyze the first efforts towards using knowledge extracted from answers to submitted queries. Finally, we review how to apply these methods to search systems, and we show how such methods can improve greatly the performance and reduce the costs of search engines.

5.2.1 Partitioning and Querying in Federated Distributed IR Systems

Parallel and distributed computing techniques have been used since many years, so far [184]. The majority of the methods that have been proposed so far makes use of knowledge acquired from past users' activity models obtained by mining web search engine query logs. The reader interested in such techniques shall find many juicy details in literature works such as [91, 92, 55, 226, 238, 93,

85, 100, 233, 169, 168, 141, 195]. In the following we review some of the basic techniques used just to set the ground for the future discussion on methods using users' feedback.

Nowadays, the mostly used parallel architectures are: *Cluster of PC*, *Grid* platforms, and the more modern *Cloud* computing facilities. The former consists of a collection of interconnected stand-alone PCs working together as a single, integrated computing resource [53, 40]. The latter is a distributed computing infrastructure for advanced science and engineering. The underlining problems of Grid concepts consist of coordinating resource sharing and problem solving in dynamic, multi–institutional, virtual organizations [81, 106]. Yahoo! Grid [235], for instance, is an example of how distributed computing is used within a large search company to improve their performance.

Since realistic web search engines usually manage distinct indexes, the only way to ensure timely and economic retrieval is designing the broker module so that it forwards a given query only to the workers managing documents related to the query topic. *Collection Selection* plays a fundamental role in the reduction of the search space. Particular attention should be paid, though, when using such a technique in a real web search system since the loss of relevant documents resulting form the exclusion of some Index servers, could impact dramatically, by degrading the overall effectiveness, on the system's performance.

A *collection selection index (CSI)* summarizing each collection as a whole, is used to decide which collections are most likely to contain relevant documents for a submitted user's query. Document retrieval, actually, only takes place at such collections.

Hawking and Thistlewaite [100] and Craswell *et al.* [65] compare several selection methods. Authors showed that the method of using only a naïve collection selection index, may lacks of effectiveness. This implies that many proposals try to improve both the effectiveness and the efficiency of the previous schema.

Moffat *et al* [153] use a centralized index on blocks of B documents. For example, each block might be obtained by concatenating documents. A query first retrieves block identifiers from the centralized index, then searches the highly ranked blocks to retrieve single documents. This approach works well for small collections, but causes a significant decrease in precision and recall when large collections have to be searched.

Garcia–Molina *et. al.* [91, 92, 93] propose GlOSS, a broker for a distributed IR system based on the boolean IR model that uses statistics over the collections to choose the ones which better fits the user's requests. The authors of GlOSS made the assumption of independence among terms in documents so, for example, if term A occurs f_A times and the term B occurs f_B times in a collection with D documents, than they estimated that $\frac{f_A}{D} \cdot \frac{f_B}{D} \cdot D$ documents contain both A and B. Authors of GlOSS generalize their ideas to vector space IR systems (gGlOSS), and propose a new kind of server, called hGlOSS, that collects information for a hierarchy of several GlOSS servers and select the best GlOSS server for a given query.

Xu and Croft [233] analyze collection selection strategies using cluster-based language models. Xu *et al.* propose three new methods of organizing a distributed retrieval system, called *global clustering*, *local clustering*, and *multiple-topic representation*. In the first method, assuming that all documents are made available in one central repository, a clustering of the collection is created; each cluster is a separate collection that contains only one topic. Selecting the right collections for a query is the same as selecting the right topics for the query. The next method is *local clustering* and it is very close to the previous one except for the assumption of a central repository of documents. This

method can provide competitive distributed retrieval without assuming full cooperation among the subsystems. The last method is *multiple-topic representation*. In addition to the constraints in local clustering, the authors assume that subsystems do not want to physically partition their documents into several collections. The advantage of this approach is that it assumes minimum cooperation from the subsystem. The disadvantage is that it is less effective than both global and local clustering.

Callan *et al.* [55] compare the retrieval effectiveness of searching a set of distributed collections with that of searching a centralized one using an inference networks in which leaves represent document collections, and nodes represent terms that occur in the collection. The probabilities that flow along the arcs can be based upon statistics that are analogous to *tf* and *idf* in classical document retrieval: document frequency *df* (the number of documents containing the term) and inverse collection frequency *icf* (the number of collections containing the term). They call this type of inference network a *collection retrieval inference network*, or *CORI* for short. They found no significant differences in retrieval performance between distributed and centralized searching when about half of the collections on average were searched for a query.

5.2.2 Query-based Partitioning and Collection Selection

The use of queries information on "*traditional*", i.e. non web, Distributed IR systems has been proposed in the past [56, 197, 198, 58, 196]. Only recently, it has been started to investigate the opportunities offered by collection selection architectures in web IR systems [173, 172, 175, 171].

Puppin [171] shows that combining different methods (*collection prioritization*, *incremental caching*, and *load balancing*) it is possible to reduce the load of each query server up to 20% of the maximum by loosing only a fraction (up to 5%) of the precision attained by the centralized system.

At the core of the technique shown by Puppin [171] there is a *collection partitioning strategy* whose goal is to cluster the most relevant documents for each query. The cluster hypothesis states that *closely associated documents tend to be relevant to the same requests* [221]. Clustering algorithms, like k-means [107], for instance, exploit this claim by grouping documents on the basis of their content.

The partitioning method used by Puppin *et al.* [173] is, instead, based on the novel *query-vector (QV) document model*, introduced by Puppin and Silvestri [172], instead exploits the cluster hypothesis the other way around. It clusters queries and successively devises a document clustering. In the QV model, documents are represented by the weighted list of queries (out of a training set) that recall them: the QV representation of document d is a vector where each dimension is the score that d gets for each query in the query set. The set of the QVs of all the documents in a collection can be used to build a query-document matrix, which can be normalized and considered as an empirical joint distribution of queries and documents in the collection. Our goal is to co-cluster queries and documents, to identify queries recalling similar documents, and groups of documents related to similar queries. The algorithm adopted is by Dhillon *et al.* [72] based on a model exploiting the empirical joint probability of picking up a query/document pair. The results of the co-clustering algorithm are then used to build the Collection Selection Index and to subsequently perform collection selection.[3]

[3] The implementation of the co-clustering algorithm used in [171] is available at `http://hpc.isti.cnr.it/~diego/phd`.

In alternative to the two popular ways of modeling documents, *bag-of-words*, and *vector space*, QV can be used to represent a document by recording which documents are returned as answers to each query. The query-vector representation of a document is built out of a query log. The actual search engine is used in the building phase: for every query in a training query set, the system stores the first N results along with their score.

Table 5.1 gives an example. The first query $q1$ recalls, in order, $d3$ with score 0.8, $d2$ with score 0.5 and so on. Query $q2$ recalls $d1$ with score 0.3, $d3$ with score 0.2 and so on. We may have empty columns, when a document is never recalled by any query (in this example $d5$). Also, we can have empty rows when a query returns no results ($q3$).

Query/Doc	d1	d2	d3	d4	*d5*	d6	...	dn
q1	-	0.5	0.8	0.4	-	0.1	...	-
q2	0.3	-	0.2	-	-	-	...	0.1
q3	-	-	-	-	-	-	...	-
q4	-	0.4	-	0.2	-	0.5	...	0.3
...	...	...	...	...	...	...	...	...
qm	0.1	0.5	0.8	-	-	-	...	-

Table 5.1: In the query-vector model, every document is represented by the query it matches (weighted with the score) [173].

This concept can be stated more formally by the following definition of the Query-vector model [172].

Definition 5.1. Query-vector model. Let Q be a query log containing queries $q_1, q_2, \ldots, q_m$. Let $d_{i_1}, d_{i_2}, \ldots, d_{i_{n_i}}$ be the list of documents returned, by a reference search engine, as results to query q_i. Furthermore, let r_{ij} be the score that document d_j gets as result of query q_i (0 if the document is not a match).

A document d_j is represented as an m-dimensional *query-vector* $\overline{d}_j = [\overline{r_{ij}}]^T$, where $\overline{r_{ij}} \in [0, 1]$ is the normalized value of r_{ij}:

$$\overline{r_{ij}} = \frac{r_{ij}}{\sum_{i \in Q} \sum_{j \in D} r_{ij}} \tag{5.1}$$

In QV model, the underlying reference search engine is treated as a black box, with no particular assumptions on its behavior. Internally, the engine could use any metric, algorithm and document representation. The QV model is simply built out of the results recalled by the engine using a given query log.

Definition 5.2. Silent documents. A silent document is a document never recalled by any query from the query log Q. Silent documents are represented by *null query-vectors*.

Incidentally, this is another important benefit granted by the use of historical query log information. The ability of identifying silent documents is a very important feature of the model because

it allows to determine a set of documents that can safely be moved to a supplemental index. Obviously, a silent document can become "*audible*" again. For example, whenever it is about a topic that become suddenly popular due to a news event. In this case, a document re-distribution will be needed and partitions will be created again.

The $\overline{r_{ij}}$ values defined according to 5.1, form a contingency matrix R, which can be seen as an empirical joint probability distribution and used by the cited co-clustering algorithm. This approach creates, simultaneously, clusters of rows (queries) and columns (documents) out of an initial matrix, with the goal of minimizing the loss of information.

Co-clustering considers both documents and queries. We, therefore, have two different sets of results: (i) groups made of documents answering to similar queries, and (ii) groups of queries with similar results. The first group of results is used to build the document partitioning strategy, while the second is the key to the collection selection strategy (see below).

More formally, the result of co-clustering is a matrix $\widehat{P}$ defined as:

$$\widehat{P}(qc_a, dc_b) = \sum_{i \in qc_b} \sum_{j \in dc_a} \overline{r_{ij}}$$

In other words, each entry $\widehat{P}(qc_a, dc_b)$ sums the contributions of $\overline{r_{ij}}$ for the queries in the query cluster a and the documents in document cluster b. Authors call this matrix simply PCAP[4] and its entries measure the relevance of a document cluster to a given query cluster, thus, forming a Collection Selection Index that naturally induces a simple, but effective, collection selection algorithm.

The queries belonging to each query cluster are chained together into *query dictionary* files. Each dictionary file stores the text of each query belonging to a cluster, as a single text file. For instance, if the four queries "*hotel in Texas*", "*resort*", "*accommodation in Dallas*", and "*hotel downtown Dallas Texas*" are clustered together as the first query cluster, the first query dictionary is qc_1="*hotel in Texas resort accommodation in Dallas hotel downtown Dallas Texas*". The second query dictionary file could be, for instance qc_2 ="*car dealer Texas buy used cars in Dallas automobile retailer Dallas TX*". A third query cluster could be qc_3 ="*restaurant chinese restaurant eating chinese Cambridge*".

When a new query q is submitted to the IR system, the BM25 metric [180] is used to find which clusters are the best matches: each dictionary file is considered as a document, which is indexed using the vector-space model, and then queried with the usual BM25 technique. This way, each query cluster qc_i receives a score relative to the query q, say $r_q(qc_i)$. In our example, if a user asks the query *"used Ford retailers in Dallas"*, $r_q(qc_2)$ is higher than $r_q(qc_1)$ and $r_q(qc_3)$.

This is used to weight the contribution of PCAP $\widehat{P}(i,j)$ for the document cluster dc_j, as follows:

$$r_q(dc_j) = \sum_i r_q(qc_i) \cdot \widehat{P}(i,j)$$

Table 5.2 gives an example. The top table shows the PCAP matrix for three query clusters and five document clusters. Suppose BM25 scores the query-clusters respectively 0.2, 0.8 and 0, for a given query q. We compute the vector $r_q(dc_i)$ by multiplying the matrix PCAP by $r_q(qc_i)$, and the collections dc3, dc1, dc2, dc5, dc4 are chosen in this order.

[4] Because authors erroneously though that the LaTeX command to typeset $\widehat{P}$ was `\cap{P}`

PCAP	dc1	dc2	dc3	dc4	dc5	$r_q(qc_i)$
qc1		0.5	0.8	0.1		0.2
qc2	0.3		0.2		0.1	0.8
qc3	0.1	0.5	0.8			0

$$
\begin{aligned}
r_q(dc_1) &= 0 + 0.3 \times 0.8 + 0 = 0.24 \\
r_q(dc_2) &= 0.5 \times 0.2 + 0 + 0 = 0.10 \\
r_q(dc_3) &= 0.8 \times 0.2 + 0.2 \times 0.8 + 0 = 0.32 \\
r_q(dc_4) &= 0.1 \times 0.2 + 0 + 0 = 0.02 \\
r_q(dc_5) &= 0 + 0.1 \times 0.8 + 0 = 0.08
\end{aligned}
$$

Table 5.2: Example of PCAP to perform collection selection. We have three query clusters: qc_1= *"hotel in Texas resort accommodation in Dallas hotel downtown Dallas Texas"*, qc_2 = *"car dealer Texas buy used cars in Dallas automobile retailer Dallas TX"* and qc_3 = *"restaurant chinese restaurant eating chinese Cambridge"*. The second cluster is the best match for the query *"used Ford retailers in Dallas"*. The third document cluster is expected to have the best answers.

The QV model and the PCAP selection function together are able to create very robust document partitions. In addition, they allow the search engine to identify, with great confidence, what the most authoritative servers are for any query. These ideas, in fact, can be used to design a distributed IR system for web pages. The strategy is as follows. First, we train the system with the query log relative to a *training period*, by using a reference centralized index to answer all the queries submitted to the system. The top-ranking results are recorded for each query. Then, the co-clustering step is applied on the resulting query-document matrix representing the QV formalization of the problem. The documents are then partitioned onto several IR cores according to the results of clustering.

In the experiments conducted by Puppin *et al.* [173], $\sim 6M$ documents are partitioned into 17 clusters: the first 16 clusters are the clusters returned by co-clustering, and the last one holds the silent documents, i.e. the documents that are not returned by any query, represented by null query-vectors (the 17-th cluster is used as a sort of *supplemental index*). Authors partitioned the collection into 16 clusters because empirically they observed that a smaller number of document clusters would have brought to a situation with a very simple selection process, while a bigger number would have created artificially small collections.

After the training, collection selection is performed using the method shown above. In this experiment, the broker actively chooses which cores are going to be polled for every query. Actually, more responsibility can be given to each core resulting in a more balanced load and in a better precision (see load balancing strategy paragraph below). The index servers holding the selected collections receive the query, and return their results, eventually merged by the broker. In order to have comparable document ranking within each index core, the global collection statistics are distributed to each IR server.

Authors were not able to obtain a list of human-chosen relevant documents for each query (as it happens with the TREC data[5]). Nevertheless, following the example of previous works by Xu

[5] The TREC web Corpus: WT10g is available at `http://www.ted.cmis.csiro.au/TRECWeb/wt10g.html`.

and Callan [232] they compare the results coming from collection selection with the results coming from a centralized index. In particular, they use the *intersection* and *competitive similarity* metrics, adapted from Panconesi *et al.* [61] and briefly recalled below.

Let G_q^N denote the top N results returned for q by a centralized index (*ground truth*), and let H_q^N be the top N results returned for q by the set of servers chosen by the collection selection strategy. The *intersection at N*, $INTER_N(q)$, for a query q is the fraction of results retrieved by the collection selection algorithm that appear among the top N documents in the centralized index:

$$INTER_N(q) = \frac{|H_q^N \cap G_q^N|}{|G_q^N|}$$

Given a set D of documents, we call *total score* the value:

$$S_q(D) = \sum_{d \in D} r_q(d)$$

with $r_q(d)$ the score of d for query q. The competitive similarity at N, $COMP_N(q)$, is measured as:

$$COMP_N(q) = \frac{S_q(H_q^N)}{S_q(G_q^N)}$$

This value measures the relative quality of results coming from collection selection with respect to the best results from the central index. In both cases, if $|G_q^N| = 0$ or $S_q(G_q^N) = 0$, the query q is not used to compute average quality values.

This strategy is tested on a simulated distributed web search engine. They use the WBR99 web document collection[6], of 5,939,061 documents, i.e. web pages, representing a snapshot of the Brazilian web (domains .br) as spidered by the crawler of the TodoBR search engine. The collection consists of about 22 GB of uncompressed data, mostly in Portuguese and English, and comprises about 2,700,000 different terms after stemming.

Along with the collection, a query log of queries submitted to TodoBR has been used, in the period January through October 2003. The first three weeks of the log has been selected as the training set and it is composed of about half a million queries, of which 190,000 are distinct. The main test set is composed by the fourth week of the log, comprising 194,200 queries. The main features of test setup are summarized in Table 5.3.

Zettair[7] was used as the central search engine. Zettair is a compact and fast text search engine designed and written by the search engine Group at RMIT University. This IR system has been modified so to implement different collection selection strategies (CORI and PCAP) in the Query Broker front end.

To assess the quality of the approach, Puppin *et al.* [174] perform a clustering task aimed at document partitioning and collection selection, for a parallel information retrieval system. To be as complete as possible, different approaches to partitioning and selection have been compared. First of all, documents are partitioned into 17 clusters. For partitions created with co-clustering, the 17-th cluster, or *overflow* cluster (OVR), holds the *supplemental index*, which stores the *silent documents*. The tested approaches are:

[6] Thanks to Nivio Ziviani and his group at UFMG, Brazil, who kindly provided the collection, along with logs and evaluated queries.

[7] Available under a BSD-style license at `http://www.seg.rmit.edu.au/zettair/`.

d	5,939,061 documents taking (uncompressed) 22 GB
t	2,700,000 unique terms
t'	74,767 unique terms in queries
tq	494,113 (190,057 unique) queries in the training set
$q1$	194,200 queries in the main test set (first week - TodoBR)

Table 5.3: Main features of test set.

- *Random*: a random allocation. This is, to the best of our knowledge, the most popular approach to document partitioning among commercial search engines [33].
- *Shingles*: k-means clustering of document signatures computed using shingling [50]. Shingles have already been used in the past for clustering text collections, [50] and for detecting duplicate pages [63, 104]. It has also been shown to be a very effective document representation for identifying near-duplicate documents.
- *URL-sorting*: it is a very simple heuristics, which assign documents block-wise, after sorting them by their URL; *this is the first time URL-sorting is used to perform document clustering*; this simple technique, already used for other IR tasks [179, 43, 203], can offer a remarkable improvement over a random assignment.
- *K-means*: k-means clustering over the document collection, represented by query-vectors.
- *Co-clustering*: co-clustering algorithm is used to compute documents and query clusters. 16 document clusters and 128 query clusters have been created through 10 iterations of the co-clustering algorithm.

CORI is the collection selection function of choice in all the tests performed, excepting the last one, where PCAP has been used. Results are shown in Table 5.4. Intersection at 5, 10, 20 ($INTER_5$, $INTER_{10}$, $INTER_{20}$) is computed, when using only a subset of servers is chosen on the basis of the ranking returned by the collection selection function used. The first column shows the value of the $INTER_k$ measure when only the most promising server is used to answer each query. The following observation can be made.

Shingles offer only a moderate improvement over a random allocation, and a bigger improvement when a large number of collections, about half, are chosen. Shingles are not able to cope effectively with the curse of dimensionality. Experimental results show that URL-sorting is actually a good clustering heuristic, better than k-means on shingles when a little number of servers is polled. URL-sorting is even better if we consider that sorting a list of a billion URLs is not as complex as computing clustering over one billion documents. *This method, thus, could become the only one feasible in a reasonable time within large scale web search engines if no usage information can be used.*

Results improve dramatically when we shift to clustering strategies based on the query-vector representation. The results of using CORI over partitions created with k-means on query-vectors (a value of $INTER_k$ of about 29% when a single partition is queried) are much better than the results obtained by other clustering strategies that do not exploit usage information (obtaining an $INTER_k$ score up to 18%).

Even better does co-clustering behave. Both CORI and PCAP on co-clustered documents are

$INTER_5$

	1	2	4	8	16	OVR
CORI on random	6	11	25	52	91	100
CORI on shingles	11	21	38	66	100	100
CORI on URL sorting	18	25	37	59	95	100
CORI on kmeans qv	29	41	57	73	98	100
CORI on co-clustering	31	45	59	76	97	100
PCAP on co-clustering	34	45	59	76	96	100

$INTER_{10}$

	1	2	4	8	16	OVR
CORI on random	5	11	25	50	93	100
CORI on shingles	11	21	39	67	100	100
CORI on URL sorting	18	25	37	59	95	100
CORI on kmeans qv	29	41	56	74	98	100
CORI on co-clustering	30	44	58	75	97	100
PCAP on co-clustering	34	45	58	76	96	100

$INTER_{20}$

	1	2	4	8	16	OVR
CORI on random	6	12	25	48	93	100
CORI on shingles	11	21	40	67	100	100
CORI on URL sorting	18	24	36	57	95	100
CORI on kmeans qv	29	41	56	74	98	100
CORI on co-clustering	30	43	58	75	97	100
PCAP on co-clustering	34	45	58	75	96	100

Table 5.4: Comparison of different clustering and selection strategies: intersection (percentage) at 5, 10, 20.

superior to previous techniques, with PCAP also outperforming CORI by about 10% (from 30% to 34%). This result is even stronger when we watch at the footprint of the collection representation, which is about 5 times smaller for PCAP [173].

We can conclude, thus, that by choosing a fixed, limited number of servers to be polled for each query, the system can return a very high fraction of the relevant results.

We report results on $COMP_N(q)$ later on this section. This strategy can cause a strong difference in the relative computing load of the underlying IR cores, if one server happens to be hit more often than another: the IR system results to be slowed down by the performance of the most loaded server. In Figure 5.11, we show a sample of the peak load reached by the Index servers when we use the four most authoritative collections for every query (i.e. the *FIXED* strategy by Puppin [171]), with

the presence of a LRU result cache of 4,000 entries: it varies from 100 to about 250 queries out of a rotating window of 1000 queries.

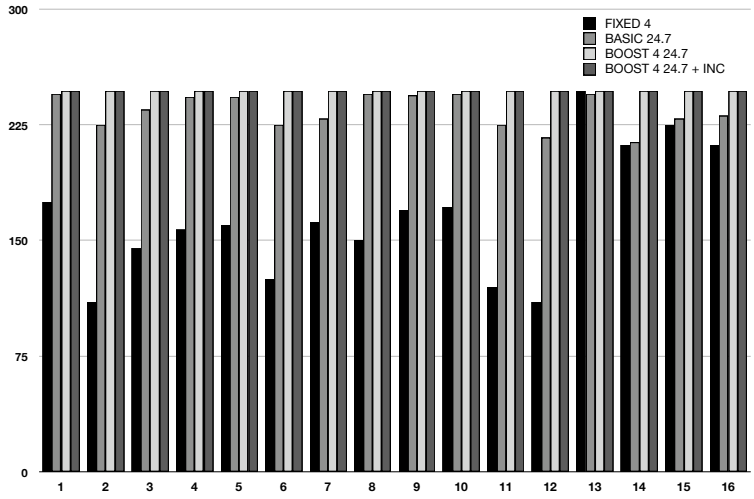


Fig. 5.11: Computing pressure (sample) on the cores when using different routing strategies: routing to the 4 most promising servers for each query (FIXED), load-driven routing capped to 24.7% load (BASIC), boost with 4 servers, cap to 24.7% (BOOST) and boost with incremental cache (INC). The load is measured as the number of queries that are served by each core from a window of 1,000 queries. BASIC, BOOST and INC are able to improve results by utilizing the idle resources: there is no need for additional computing power [171].

The LRU cache is used to make experiments as realistic as possible. Not filtering queries through an LRU cache would make things even worse. Roughly speaking, we can design a load-driven collection selection system where more servers can choose to answer a given query, even if they are not the most authoritative for it, if they happen to be momentarily under-loaded. Furthermore, this way, the system can exploit the load differences among servers to gather more possible relevant results. We can instruct the IR cores about the maximum allowed computing load, and let them drop the queries they cannot serve. In this configuration, the broker still performs collection selection, and ranks the collections according to the expected relevance for the query at hand. The query is broadcasted, but now every server is informed about the relevance it is expected to have w.r.t. the given query. At this point, each Index server can choose to either serve, or drop the query, according to its instant load[8].

The most promising core receives a query tagged with top priority, equal to 1. The other cores c receive a query q tagged with linearly decreasing priority $p_{q,c}$ (down to $1/N$, with N cores). At time t, a core c with current load $\ell_{c,t}$ serves the query q if:

$$\ell_{c,t} \times p_{q,c} < L$$

[8] An implementation issue arising here, is to give the query broker the ability to determine if a query has been answered or not by a core. If the query is accepted, the core answers with its local results. Otherwise, it can send a negative acknowledgment. Alternatively, the broker can use a time-out, so to guarantee a chosen query response time.

where L is a load threshold that represents the computing power available to the system. This is done to give preferred access to queries on the most promising cores: if two queries involve a core c, and the load in c is high, only the query for which c is very promising is served by c, and the other one is dropped. This way, overloaded cores are not hit by queries for which they represent only a second choice.

If the condition at the index server is met, the core computes its local results and returns them to the broker. In this model, thus, the broker, instead of simply performing collection selection, performs a process of *prioritization*, i.e. chooses the priority that a query should get at every Index server.

In the experimental evaluation three query routing strategies are devised and compared:

- *Fixed* $< T >$: the query is routed to the T most relevant servers, according to a collection selection function, with T given once for all. This allows us to measure the computing power required to have at least T servers answer a query (and to have a guaranteed average result quality).
- *Load-driven basic (LOAD)* $< L >$: the system contacts all servers, with different *priority*. Priority ranges from 0 down to $1/N$ (on a system with N cores). The load threshold on cores is fixed to L.
- *Load-driven boost (BOOST)* $< L, T >$: same as load-driven, but here we contact the first T servers with maximum priority, and then the other ones with linearly decreasing priority. By boosting, we are able to keep the lower loaded servers closer to the load threshold. Boosting is valuable when the available load is higher, as it enables us to use the lower loaded servers more intensively. If the threshold L is equal to the load reached by FIXED $< T >$, we know that we can poll T servers every time without problems. The lower-priority queries are dropped when we get closer to the threshold.

Using the load-driven strategy, we are able to keep busy all the cores in the system by asking them to answer also the queries for which they are less authoritative (see Figure 5.11).

The last optimization shown by Puppin [171] is *incremental caching*. Basically, it is a caching scheme, i.e. all the consideration above on caching still hold, plus it has an additional information per query that records which servers have answered back with results. Therefore, *the incremental cache holds the best r results returned thus far by the underlying search service*. Performance, obviously, depends on the policy used. In this case, however, hit-ratio and throughput are not the only measurable results: precision of results returned for each query is also important due to the fact that the higher the number of times a query is requested the higher the precision gets. Since queries follow a power-law, we expect precision of frequent queries to stabilize to the maximum attainable possible, after a very few references to those queries.

To formalize in detail incremental caching, we need to redefine the type of entries stored in a cache line.

Definition 5.3. A *query-result record* is a quadruple of the form $< q, p, \overline{r}, \overline{s} >$, where: q is the query string, p is the number of the page of results requested, $\overline{r}$ is the ordered list of results associated with q and p, $\overline{s}$ is the set of servers from which results in $\overline{r}$ are returned. Each result is represented by a pair $< doc\_id, score >$.

If we cache an entry $< q, p, \overline{r}, \overline{s} >$, this means that only the servers in $\overline{s}$ answered, and that they returned $\overline{r}$. Also, since in an incremental cache the stored results might be updated, we need to store the score (along with the document identifier) to compare the new results with the old ones.

The complete algorithm is formally shown in Table 5.5. Results in an incremental cache are continuously modified by adding results from the servers that have not been queried yet. The set $\overline{s}$ serves to this purpose and keeps track of the servers that have answered so far.

For a query with topic q, result page p, with a selection function $\rho(q)$.

1. Look up (q,p).
2. If not found:
    1. tag the query with priorities $\rho(q)$;
    2. let $\overline{s}$ be the set of servers that accepted the query;
    3. let $\overline{r}$ be the result set coming from $\overline{s}$;
    4. select and remove the best candidate for replacement;
    5. store $< q, p, \overline{r}, \overline{s} >$;
    6. return $\overline{r}$ to user.
3. If found $< q, p, \overline{r}, \overline{s} >$:
    1. tag the query with priorities $\rho(q)$ for the remaining servers;
    2. let $\overline{s_2}$ be the set of servers that accepted the query;
    3. let $\overline{r_2}$ be the result set coming from $\overline{s_2}$;
    4. add $\overline{s_2}$ to $\overline{s}$;
    5. merge $\overline{r_2}$ with $\overline{r}$ (sorting by score);
    6. return $\overline{r}$ to user.

Table 5.5: The incremental cache algorithm, as performed by the search engine broker. The case when the cache is not full is straightforward and not shown [171]

When load-driven selection is used, only the results coming from the polled servers are available for caching. In case of a subsequent hit, the selection function gives top priority to the first server that was not polled before. Let's say, for example, that for a query q, the cores are ranked in this order: $s4$, $s5$, $s1$, $s3$ and $s2$. In other words, $s4$ has priority 1. Now, let's say that only $s4$ and $s1$ are actually polled, due to their lower load. When q hits the system a second time, $s5$ has the top priority, followed by $s3$ and $s2$. Their results are eventually added to the incremental cache.

This strategy does not add computing pressure to the system w.r.t. to boost (see Figure 5.11). The advantage comes to the fact that repeated queries get higher priority also for the low-relevance servers, at the cost of other non-repeated queries.

It is important to emphasize that load-driven routing and incremental caching strategy work independently from the selection function, which is used as a black box. Puppin [171] uses PCAP,

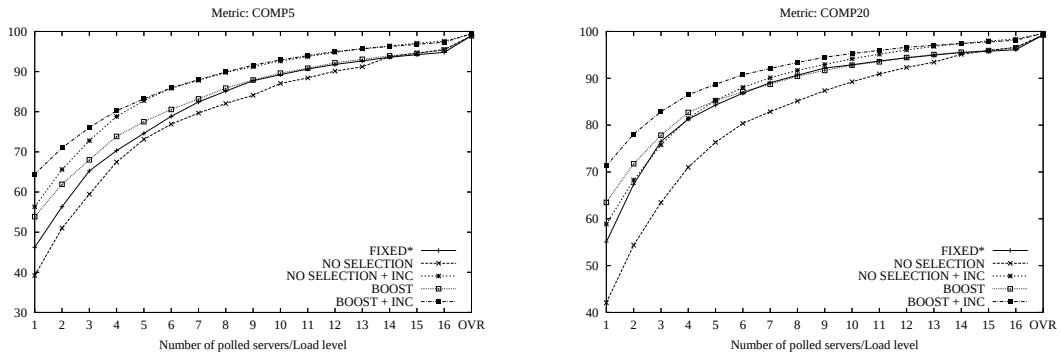


Fig. 5.12: Comparison with real query timing information on TodoBR [171].

yet these concepts can be successfully utilized with any other collection selection algorithm. Across all configurations, load-driven routing, combined with incremental cache, clearly surpasses the other strategies. Furthermore, changing from fixed to load-driven routing and incremental caching does not add computing pressure: results can be improved with the same computing requirements in the IR cores, at the cost of a negligibly higher cache complexity.

In Figure 5.12, we show the competitive similarity of different strategies, under this model, for the tests performed with the TodoBR log. With a peak load set to the *average* load of FIXED $< 1 >$, i.e. the average load needed to always poll only the most promising server for every query, the strategy based on load-driven routing and incremental caching surpasses a competitive similarity of 65% (with an 80% $COMP_{10}$).

Moreover, in order to assess how the strategy exploits the underlying cluster, in Table 5.6 the average number of servers per query is shown. While the configuration without collection selection polls more servers on average (see Table 5.6), they are less relevant for the queries, and the final results are better for the approaches based on collection selection, which poll fewer, but more relevant, servers.

5.2.3 Partitioning and Load Balancing in Term-partition based Search Engines

Term-partitioning has always been considered not as effective as document partitioning in real search settings. Nonetheless, some recent works have made this fact not as solid as before. Due to recent proposals of a pipelined architecture, the *term partitioning* approach is now attracting some attention again [152]. According to this partitioning strategy, the set of terms occurring in the index, i.e. the *lexicon*, is partitioned among the Index servers, and each server is able to discover only the documents containing a subset of the lexicon. The strategies proposed so far suffer from a significant load imbalance, due, again, to the power-law distribution of terms in user queries and indexed documents.

In the classical term partitioning approach, the query is segmented into several disjoint sub-queries. Each sub-query must be sent to the server that is responsible for the corresponding terms.

	1	2	3	4	5	6	7	8	
FIXED*	0.86	1.77	2.75	3.64	4.58	5.56	6.50	7.41	
NO COLL	5.72	7.57	9.17	10.43	11.43	12.08	12.59	13.04	
NO COLL + INC	8.72	10.35	11.57	12.65	13.31	13.90	14.26	14.61	
BOOST	2.04	2.88	3.78	4.68	5.57	6.47	7.30	8.23	
BOOST + INC	5.32	6.52	7.69	8.78	9.70	10.57	11.34	12.05	

	9	10	11	12	13	14	15	16	16+OVR
FIXED*	8.33	9.26	10.23	11.15	12.06	13.04	14.01	15.08	16.34
NO COLL	13.45	13.86	14.16	14.57	14.86	15.19	15.40	15.63	16.34
NO COLL + INC	14.88	15.14	15.37	15.58	15.77	15.92	16.06	16.17	16.62
BOOST	9.14	9.98	10.91	11.93	12.85	13.83	14.71	15.50	16.34
BOOST + INC	12.65	13.17	13.67	14.19	14.68	15.16	15.64	15.98	16.62

Table 5.6: Average number of servers polled per query with different strategies, for different load levels. FIXED* polls a fixed number of servers, but queries can be dropped by overloaded servers. Even if Boost and Boost + Incremental Caching are utilizing, on average, a smaller number of servers than broadcast (i.e. no collection selection), the choice is more focused and gives better results [171].

Note that each sub-query can be served locally by each server that has to return, in principle, the whole result list. Then, local results must be merged in order to produce the list of relevant DocIDs to be returned to the user. In the pipelined approach, the broker, after dividing the query into sub-queries, packs them up into a structure denoting the list of servers to invoke. The first server of the list receives the whole query-pack, resolves its sub-query, sends its own results, along with the remainder of the query-pack, to the second server in the list. This, in turn, resolves its sub-query and merges the results with those received from the previous index server. This process goes on until no more sub-queries are left and the last index server in the list is able to select the r best documents that are sent back to the broker. Finally, the broker returns the rendered result page to the user.

Term-partitioning is easily explained through an example. Suppose we have an index built on a toy collection made up of three documents: $d_1 = \{t_1, t_2\}$, $d_2 = \{t_1, t_3\}$, $d_3 = \{t_2, t_3\}$. Suppose, we have two index servers and we store t_1, and t_2 on server 1, t_3 on server 2. First server's index contains $t_1 \rightarrow \{d_1, d_2\}$, and $t_2 \rightarrow \{d_1, d_3\}$. Server number two contains $t_3 \rightarrow \{d_2, d_3\}$. If a user submits a query consisting of terms t_1 and t_2 only the first server is involved in the resolution. For the query t_1, t_3, instead, both servers are involved. In traditional term partitioning the broker sends t_1, and t_2 separately to server 1 and 2, respectively. In pipelined term-partitioning the broker packs t_1, and t_2, sends the packet to server 1, which extracts t_1's postings. Afterwards, server 1 forwards t_2 along with the list $\{d_2, d_3\}$ to server 2. Finally, server 2 sends the list consisting of the document d_2 back to the broker.

From the above example, it is quite clear why term-partitioning suffers a lot from the imbalance problem. If the query stream consists of all resubmissions of query t_1, t_2, then server two is never involved in a query resolution. Balancing the load is, therefore, the first objective if we want to

prove the applicability of term partitioning in real-world web search engines.

Results of preliminary studies made [151, 240, 142] are good and promising. The aim at balancing the load in a term-partitioned distributed search system using information about how terms are distributed across requests in the past. Anyway, none of the proposed studies have shown, yet, that term-partitioning can be better than document-partitioning.

The load balancing problem is also addressed by Moffat *et al.* [151], where the authors exploited both term frequency information and postings list replication to improve load balancing in their pipelined WSE. Although they showed a strong improvement of about 30% in the throughput of the system, still the document partitioning approach behaves better. The term distribution strategy used is called *fill smallest*. It basically consists in applying a bin-packing heuristic to fill up the partitions by weighting each term with its frequency of occurrence within a stream of queries. This technique is enhanced by exploiting partial replications of terms and associated postings lists. As replication strategy, they proposed to replicate up to $1,000$ most frequent terms. They also tested a multi-replicate strategy, in which they replicated the 100 most frequent terms only: the most frequent one is placed on all their eight servers, then the following nine frequent ones are placed on four of them, and then the other ninety terms are placed on two servers.

Nevertheless, believing in the potential of the pipelined query evaluation, authors envision in term partitioning a great potential. The authors indicate as future directions of research the exploration of load balancing methodologies based on the exploitation of term usage patterns present in the query stream. Such patterns can drive both the dynamic reassignment of lists while the query stream is being processed, and the selective replication of the most accessed inverted lists.

Such techniques are studied by Lucchese *et al.* [142]. They demonstrate the feasibility and efficacy of exploiting a frequent-patterns driven partitioning of the vocabulary of terms among the servers, in order to enhance the performance of a term-partitioned, large-scale, pipelined WSE. The model they propose takes into account: correlation of terms in user queries, the disk cost as, the OS buffer cache, and the communication and computation overheads. They shown, through simulated results, that the novel model overcomes the performance limits of the previous partitioning strategies.

Without entering too much into the theoretical details of the methods by Lucchese *et al.* [142] we show some results proving that the method effectively reduces the load unbalance and, at the same time, reduces the number of queried servers. These two parameters create, obviously, a trade-off. The parameter α is a value tuning the importance of load balancing over the average number of servers queried in the original formula of Lucchese *et al.* [142].

All the results are drawn from the TodoBR query log but the same conclusions can be drawn also for other query logs. All query logs were preliminarily cleaned and transformed into a transactional form, where each query is simply formed by a query identifier and the list $t_1, t_2, \ldots, t_q$ of terms searched for. The terms were all converted to lower case, but neither stemming nor stopword removal was performed. The first 2/3 of each query log were used to drive the partitioning of the index, while the last part was used to test the partitioning obtained. Since they do not have available a real collections of documents coherent with the query logs, they validated the proposed approach by simulating a broker and assuming constant times for disk, retrieval and network disregarding the lengths of the involved posting lists. It is worth noticing that the model is however sound and as general as possible. The knowledge of the actual values of the above parameters could be easily

Servers	Baseline Cases: random	Baseline Cases: bin packing	Term Ass. $\alpha = 0.9$
1	28	28	50
2	31	30	20
3	17	17	14
> 3	24	25	16

(a) Percentages of queries as a function of the number of servers involved in their processing. The parameter α states the how much load-balancing should be preferred over term-packing [142]

	Replication Factors					
	0.0001		0.0005		0.001	
Servers	*bin pack.*	*term ass.*	*bin pack.*	*term ass.*	*bin pack.*	*term ass.*
TodoBR						
1	42	54	56	62	63	67
2	31	22	22	18	19	16
3	12	10	9	8	8	8
> 3	15	14	12	11	10	9

(b) Effect of replicating the index entries of most frequently queried terms [142].

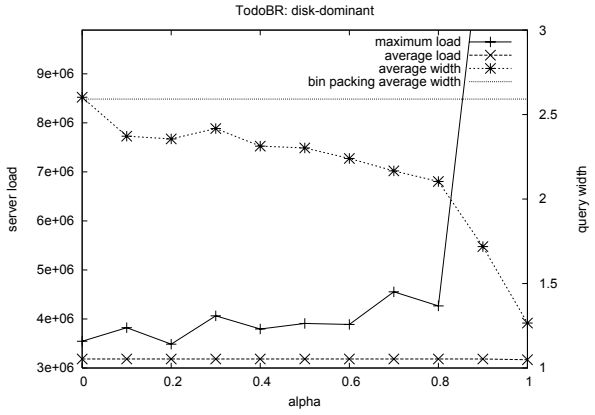
Fig. 5.13: Term assignment results.

taken into account during the term assignment process.

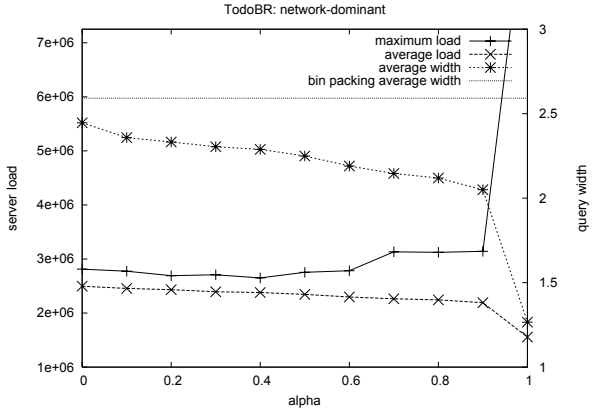
The figures reported in Table 5.13 show the percentage of queries occurring in the test sets of the TodoBR query log as a function of the number of servers queried. Each column of the table refers to a different assignment of terms to the partitions. In particular, the baseline cases are *random* assignment – i.e. the traditional one, and *bin packing* [151]. The term assignment algorithm is executed with values of α equal to 0.8 meaning that we are preferring to lower the number of server per query. The assignment strategy allows to remarkably increase the number of queries involving only one server. Obviously, the less the servers involved in answering a query the lower the query response time and the communication volume. On the *TodoBR* log the number of queries served by a single server almost doubled w.r.t. the *random* and *bin packing* assignments. As a consequence, the number of queries requiring more than one server decreases correspondingly. We can see that the number of queries solved by more than 3 servers is reduced by at least 1/3 on *TodoBR*. The effect of replicating in all the servers the index entries of some of most frequently queried terms was also tested. By introducing only very small percentages of replicated terms, ranging from 0.001% to 0.1% of the the total number of terms, the effect on the average number of per-query servers were very remarkable. Indeed, replication is very effective in reducing the average number of per-query servers also in the baseline case of bin packing. However, the advantages of using term assignment technique are remarkable also in these tests.

Load balancing also improved. Figure 5.14a, and 5.14b report the comparison of the average number of servers vs. load balancing in TodoBR query log when term assignment is performed. In particular, Figure 5.14a reports the case where the load balancing is preferred over reducing the number of servers involved per query. Whereas, Figure 5.14b shows the effect on the load of giving more importance to reducing the average number of servers involved per query.

Therefore, term assignments improved WSE throughput and query response time with respect to random and bin packing [151] term assignments. However, since a web search engine is a complex, highly nonlinear environment, the analysis conducted by Lucchese *et al.* [142] and thus the above results should be confirmed by testing performed on an actual web search engines with an actual index.



(a) Load balancing first



(b) Less servers best

Fig. 5.14: Number of servers vs. load balancing in TodoBR term partitioning experiments. From [142].

5.3 Summary

Differently from the previous chapter, this one presented how the knowledge mined from query logs can be used to speed-up query processing in search engines. We showed two major directions:

- *Caching*, that consists in exploiting past usage information to build cache replacement policies suitable for search engine workloads;
- *Data partitioning*, that is the design of carefully chosen strategies to improve the placement of data within a distributed web search engine. For this topic, we showed how to improve both document and term partitioned search engines.

6

New Directions

As the title says we review some of the open problems and challenges as highlighted by many scholars. Obviously, this section does not contain any consolidated result but only some new ideas that are briefly sketched.

6.1 Eye Tracking

As we have seen in the Enhancing Effectiveness chapter, relevance feedback is a very important mean with which one can improve the users' search experience by adapting the engine's ranking function to particular class of users, or to a particular time period.

Feedback of some sort can also be obtained by observing how people interact with a search result page by tracking eye movements [90, 74, 95, 70].

As an example of what kind of information can be captured by eye tracking techniques, Figure 6.1 shows users' fixation patterns for a page of web search results. It is clearly shown that users clearly read the contextual descriptions, especially on the seventh result.

By using eye tracking some useful feedback information, not previously available with click-through information, can be envisioned. Indeed, a lot of nice applications for this kind of feedback can be thought: learning how to better rank search results, how to place advertisement in a way they capture more attention from users, etc.

For instance, Guan and Cutrell [95] perform the following experiment: for a given search query the more relevant results (i.e. *targets* in their terminology) are alternatively displayed (for different users) at the top, in the middle, at the bottom of search result page. Therefore, this study empirically evaluates how people's attention is distributed across search results when the target is systematically manipulated to be displayed at different positions. As a result, it is shown that people spend more time and are less successful in finding target results when targets were displayed at lower positions in the list. When people could not find the target results for navigational search, they either selected the first result, or switched to a new query. From this study it could be concluded that

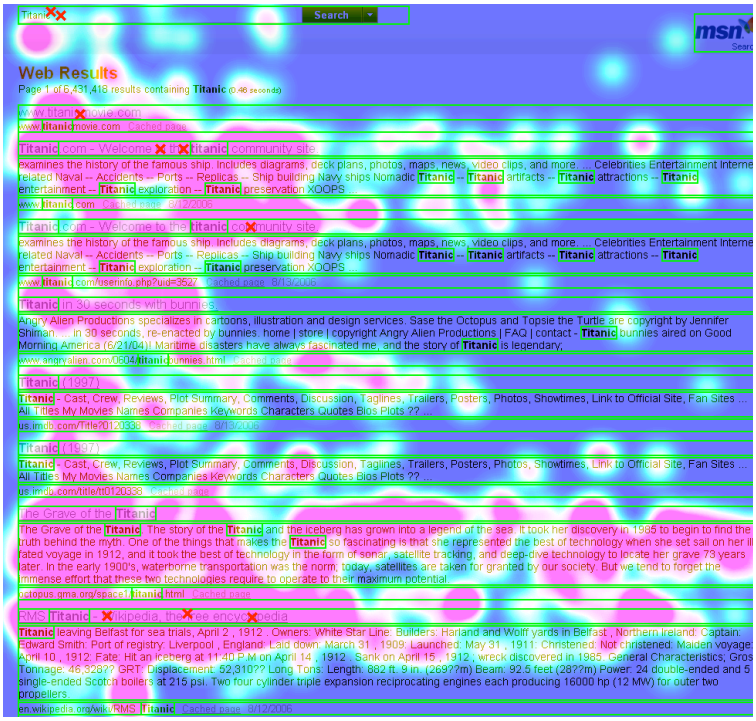


Fig. 6.1: Heat map visualization of the areas across which three different users direct their sights [70].

> "*the search engine could show variety of different search results where "best" ranking is not clear so that users could have an accurate estimation of the relevance of results and then behave accordingly.*"

Cutrell and Guan [70] describe a study that used eye tracking to explore the effects of changes in the presentation of search results. An interesting finding is that making snippets richer significantly improves the performance in terms of attractiveness for informational queries, but degrades performance for navigational queries. These results suggest this difference in performance is due to the fact that as the snippet length increases, users pay more attention to the snippet and less attention to the URL located at the bottom of the search result.

6.2 Web Search Advertisement

Web advertisement is more and more used by a continuously growing number of commercial bodies [89].

Advertisement is another area in which relevance feedback and click-through data can help a lot. Obviously, since the main goal of online advertisement is to attract clicks, analyzing the characteristics of the most clicked advertisement pages could help in determining a better advertisement placement.

To be more precise, an advertisement can be modeled as a quadruple $\langle \mathcal{T}, \mathcal{D}, \mathcal{U}, \mathcal{K} \rangle$, where: $\mathcal{T}$ is the title of the advertisement; $\mathcal{D}$ is the description – usually a few words that effectively describe what is advertised; $\mathcal{U}$ is the URL of the landing page for that advertisement; $\mathcal{K}$ is a set of keywords

related to the advertised business with the relative maximum amount of money the advertiser is willing to pay in case someone clicks the advertisement.

The maximum amount of money willed to pay in case of a click on the relative advertisement, is called *bid*[1], and usually is part of the advertisement ranking process. Indeed, the process of specifying a maximum amount of money for a keyword is commonly referred to as "*to bid for a keyword*".

One of the main factors involved in the estimation of the CTR is the relevance of the advertisement with respect to the user query. In the simplest case this can be done by selecting all those advertisements whose keywords match at least one term in the query. For instance, if a user submit the query "first aid" all those advertisements that have bid either for "first", for "aid", or both are selected. This, indeed, does not keep into account any relevance judgement with respect to query results possibly presenting users advertisements that are not of their interest. Therefore, the number of clicks generated by this naïve technique can be low and should be improved.

For instance, a straightforward way to enhance the CTR for an advertiser is to count the number of clicks it has received in the past. However, this naïve approach may result not as effective as possible. In fact, it penalizes newly inserted, or rarely recalled advertisements. Therefore, a finer analysis of query log entries and click-through data could enable the finding of better ranking functions for advertisers.

6.3 Time-series Analysis of Queries

Here we present an alternative viewpoint on query logs. Better to say, queries can be viewed as signals in the domain of time. In each time unit we record the occurrences of the query. This would result in a sort of signal to which standard temporal series techniques may be applied [222, 223, 86, 59, 243]. The techniques reported in the above papers allow for the discovering 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 news 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 behaviors such as the rise in one curve to the rise in the second, peak to peak, run to run. Figure 6.2 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.

Computing a DTW is simply done by a dynamical programming algorithm minimizing the distance in terms of euclidean distance between to time series points.

The algorithm shown in Figure 6.1 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.

Goal of the project described in the article, was to discover how time-series were correlated in order to be able to use events in one information source to predict those in another. For example as the article reports: "one might expect a ramp-up much earlier for expected events in the query-

[1] This recall the action of bidding in auctions. Actually this auction-based mechanism is part of the advertisement process.

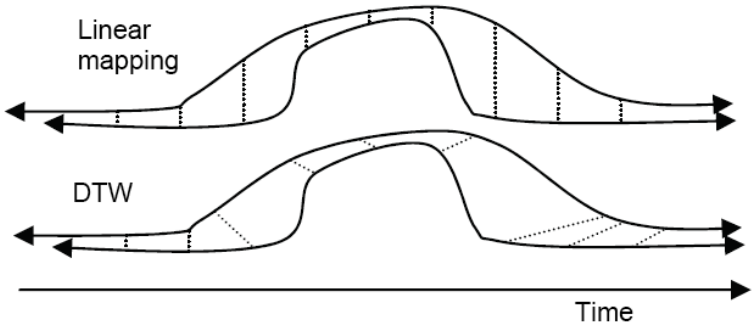


Fig. 6.2: The difference of using DTW against a simple linear mapping for comparing two time series.

Procedure DynamicTimeWarping(x, y).

(1) DTW$[0,0] = 0$;
(2) for $i = 1..\text{length}(x)$
  (a) DTW$[0,i]$,DTW$[i,0] = \infty$;
  (b) for $i = 1..\text{length}(x)$
    i. for $i = 1..\text{length}(y)$
      A. cost $= |x(i) - y(j)|$;
      B. DTW$[i,j]$ = min(DTW$[i-1,j]$ + *cost*, DTW$[i,j-1]$ + *cost*,DTW$[i-1,j-1]$ + *cost*);

Table 6.1: Dynamic Time Warping Algorithm.

logs followed by a burst of posting and news activity in the blog and NEWS datasets around the time of the actual event". The datasets used were two query logs (from MSN, and AOL), a blog dataset, a NEWS dataset. By using human-based tests they devised 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.

There are still many open issues that are interesting to, at least, enumerate:

- The effects of query log analysis on exploratory search
- Complex and interactive question answering;
- Client-side instrumentation (including keystroke and mouse movement analysis)
- Direct mental activity analysis

This shows how this research field is still in its infancy and needs the efforts of many researchers to enable a better search experience for end users. On the other hand, search companies should find a way to give researchers not working in private companies access to query log information. In this case, not only search experience is enhanced, but also REsearch experience will be.

6.4 Summary

This chapter is a sort of "*What's Next?*" for the topic presented in this survey. We have presented some challenging problems whose solutions are trailblazing. These problems have not fully solved, yet. In particular, very few research papers have been presented on how to improve the effectiveness (read "*incomes*") of advertisement placement algorithms. We strongly believe that one of most promising directions is, thus, represented by using query log information.

Conclusions

This work has covered some of the most important issues in web search engine optimization through past query mining. Starting from first order statistics of query distributions (i.e. query frequency, click frequency, page popularity, etc.), we have shown more complex analyses of historical web search usage data such as: query sessions (also known as query chains) and social relations between queries (i.e. folksonomies).

Results and observations from this introductory parts meet the two central chapters where technique for enhancing search effectiveness and efficiency are presented.

Due to the experimental nature of these analyses, results are, to some extent, fragmented into several parts that, in the vast majority of the cases, are taken by the relative articles published in literature. Actually, a comparison of the various techniques would require to re-implement them and this was beyond the scope, and not much in the spirit of this work.

To conclude we really hope to have been able to give readers the basic tools they can use for working on this, still quite young, field of query log mining.

Acknowledgements

This journey has come to an end...

I have enjoyed so much writing this survey for many reasons. I have had the opportunity to discover new material and papers by "crawling" the space of papers in the literature by following the "links" their references showed to me. Apart from kidding, I must thank first of all Jamie Callan, and Fabrizio Sebastiani (in strict alphabetical order) for having helped me so much during the time I spent writing this survey. I want also thank the reviewers for all the useful and invaluable comments.

Finally, thanks to Francesca who "joyfully" supported me during this journey!

References

[1] E. Adar, "User 4xxxxx9: Anonymizing query logs," in *Query Log Analysis: Social And Technological Challenges. A workshop at the 16th International World Wide Web Conference (WWW 2007)*, (E. Amitay, C. G. Murray, and J. Teevan, eds.), May 2007.

[2] E. Adar, D. S. Weld, B. N. Bershad, and S. S. Gribble, "Why we search: visualizing and predicting user behavior," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (New York, NY, USA), pp. 161–170, ACM, 2007.

[3] A. Agarwal and S. Chakrabarti, "Learning random walks to rank nodes in graphs," in *ICML '07: Proceedings of the 24th international conference on Machine learning*, (New York, NY, USA), pp. 9–16, ACM, 2007.

[4] E. Agichtein, E. Brill, and S. Dumais, "Improving web search ranking by incorporating user behavior information," in *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 19–26, ACM, 2006.

[5] E. Agichtein, E. Brill, S. Dumais, and R. Ragno, "Learning user interaction models for predicting web search result preferences," in *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 3–10, ACM, 2006.

[6] E. Agichtein and Z. Zheng, "Identifying "best bet" web search results by mining past user behavior," in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 902–908, ACM, 2006.

[7] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, (P. Buneman and S. Jajodia, eds.), pp. 207–216, ACM Press, 1993.

[8] F. Ahmad and G. Kondrak, "Learning a spelling error model from search query logs," in *Proceedings of the 2005 Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, (Vancouver, Canada), pp. 955–962, Association for Computational Linguistic, October 2005.

[9] C. Anderson, *The Long Tail*. Random House Business, 2006.

[10] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan, "Searching the web," *ACM Trans. Inter. Tech.*, vol. 1, no. 1, pp. 2–43, 2001.

[11] V. Authors, "About web analytics association," Retrieved on August 2009. `http://www.webanalyticsassociation.org/aboutus/`.

[12] R. Baeza-Yates, *Web Mining: Applications and Techniques*, ch. Query Usage Mining in Search Engines, pp. 307–321. Idea Group, 2004.

[13] R. Baeza-Yates, "Algorithmic challenges in web search engines," in *Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, (Valdivia, Chile), pp. 1–7, 2006.

[14] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri, "Challenges in distributed information retrieval," in *International Conference on Data Engineering (ICDE)*, (Istanbul, Turkey), IEEE CS Press, April 2007.

[15] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "The impact of caching on search engines," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 183–190, ACM, 2007.

[16] R. Baeza-Yates, A. Gionis, F. P. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "Design trade-offs for search engine caching," *ACM Trans. Web*, vol. 2, no. 4, pp. 1–28, 2008.

[17] R. Baeza-Yates, C. Hurtado, and M. Mendoza, *Query Recommendation Using Query Logs in Search Engines*, pp. 588–596. Vol. 3268/2004 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, November 2004.

[18] R. Baeza-Yates, C. Hurtado, and M. Mendoza, "Ranking boosting based in query clustering," in *Proceedings of 2004 Atlantic Web Intelligence Conference*, (Cancun, Mexico), 2004.

[19] R. Baeza-Yates and A. Tiberi, "Extracting semantic relations from query logs," in *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 76–85, ACM, 2007.

[20] R. A. Baeza-Yates, "Applications of web query mining," in *Advances in Information Retrieval, 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005, Proceedings*, (D. E. Losada and J. M. Fernández-Luna, eds.), pp. 7–22, Springer, 2005.

[21] R. A. Baeza-Yates, "Graphs from search engine queries," in *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings*, (J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plasil, eds.), pp. 1–8, Springer, 2007.

[22] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza, "Improving search engines by query clustering," *JASIST*, vol. 58, no. 12, pp. 1793–1804, 2007.

[23] R. A. Baeza-Yates, C. A. Hurtado, M. Mendoza, and G. Dupret, "Modeling user search behavior," in *Third Latin American Web Congress (LA-Web 2005), 1 October - 2 November 2005, Buenos Aires, Argentina*, pp. 242–251, IEEE Computer Society, 2005.

[24] R. A. Baeza-Yates, F. Junqueira, V. Plachouras, and H. F. Witschel, "Admission policies for caches of search engine results," in *SPIRE*, pp. 74–85, 2007.

[25] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[26] R. A. Baeza-Yates and F. Saint-Jean, "A three level search engine index based in query log distribution," in *SPIRE*, pp. 56–65, 2003.

[27] J. Bar-Ilan, "Access to query logs – an academic researcher's point of view," in *Query Log Analysis: Social And Technological Challenges. A workshop at the 16th International World Wide Web Conference (WWW 2007)*, (E. Amitay, C. G. Murray, and J. Teevan, eds.), May 2007.

[28] Z. Bar-Yossef and M. Gurevich, "Mining search engine query logs via suggestion sampling," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 54–65, 2008.

[29] R. Baraglia, F. Cacheda, V. Carneiro, F. Diego, V. Formoso, R. Perego, and F. Silvestri, "Search shortcuts: a new approach to the recommendation of queries," in *RecSys '09: Proceedings of the 2009 ACM conference on Recommender systems. To Appear*, (New York, NY, USA), ACM, 2009.

[30] R. Baraglia, F. Cacheda, V. Carneiro, V. Formoso, R. Perego, and F. Silvestri, "Search shortcuts: driving users towards their goals," in *WWW '09: Proceedings of the 18th international conference on World wide web*, (New York, NY, USA), pp. 1073–1074, ACM, 2009.

[31] R. Baraglia, F. Cacheda, V. Carneiro, V. Formoso, R. Perego, and F. Silvestri, "Search shortcuts using click-through data," in *WSCD '09: Proceedings of the 2009 workshop on Web Search Click Data*, (New York, NY, USA), pp. 48–55, ACM, 2009.

[32] R. Baraglia and F. Silvestri, "Dynamic personalization of web sites without user intervention," *Commun. ACM*, vol. 50, no. 2, pp. 63–67, 2007.

[33] L. A. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.

[34] S. M. Beitzel, E. C. Jensen, A. Chowdhury, O. Frieder, and D. Grossman, "Temporal analysis of a very large topically categorized web query log," *J. Am. Soc. Inf. Sci. Technol.*, vol. 58, no. 2, pp. 166–178, 2007.

[35] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder, "Hourly analysis of a very large topically categorized web query log," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 321–328, ACM, 2004.

[36] S. M. Beitzel, E. C. Jensen, O. Frieder, D. D. Lewis, A. Chowdhury, and A. Kolcz, "Improving automatic query classification via semi-supervised learning," in *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, (Washington, DC, USA), pp. 42–49, IEEE Computer Society, 2005.
[37] S. M. Beitzel, E. C. Jensen, D. D. Lewis, A. Chowdhury, and O. Frieder, "Automatic classification of web queries using very large unlabeled query logs," *ACM Trans. Inf. Syst.*, vol. 25, no. 2, p. 9, 2007.
[38] L. A. Belady, "A study of replacement algorithms for a virtual storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.
[39] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.
[40] "Beowulf Project at CESDIS,". http://www.beowulf.org.
[41] M. Bilenko and R. W. White, "Mining the search trails of surfing crowds: identifying relevant websites from user activity," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*, (New York, NY, USA), pp. 51–60, ACM, 2008.
[42] B. Billerbeck, F. Scholer, H. E. Williams, and J. Zobel, "Query expansion using associated queries," in *Proceedings of the twelfth international conference on information and knowledge management*, pp. 2–9, ACM Press, 2003.
[43] P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *WWW '04: Proceedings of the 13th international conference on World Wide Web*, (New York, NY, USA), pp. 595–602, ACM Press, 2004.
[44] J. Boyan, D. Freitag, and T. Joachims, "A machine learning architecture for optimizing web search engines," in *Proceedings of the AAAI Workshop on Internet-Based Information Systems*, 1996.
[45] O. Boydell and B. Smyth, "Capturing community search expertise for personalized web search using snippet-indexes," in *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, (New York, NY, USA), pp. 277–286, ACM, 2006.
[46] J. S. Breese, D. Heckerman, and C. M. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *UAI*, pp. 43–52, 1998.
[47] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *WWW7: Proceedings of the seventh international conference on World Wide Web 7*, (Amsterdam, The Netherlands, The Netherlands), pp. 107–117, Elsevier Science Publishers B. V., 1998.
[48] A. Z. Broder, "A taxonomy of web search," *SIGIR Forum*, vol. 36, no. 2, pp. 3–10, 2002.
[49] A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang, "Robust classification of rare queries using web knowledge," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 231–238, ACM, 2007.
[50] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," in *Selected papers from the sixth international conference on World Wide Web*, (Essex, UK), pp. 1157–1166, Elsevier Science Publishers Ltd., 1997.
[51] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*, (New York, NY, USA), pp. 89–96, ACM, 2005.
[52] C. J. C. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions.," in *NIPS*, (B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 193–200, MIT Press, 2006.
[53] R. Buyya, ed., *High Performance Cluster Computing*. Prentice Hall PTR, 1999.
[54] H. C. by Thomas, E. L. Charles, L. R. Ronald, and S. Clifford, *Introduction to Algorithms*. The MIT Press, 2001.
[55] J. P. Callan, Z. Lu, and W. B. Croft, "Searching distributed collections with inference networks," in *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 21–28, ACM, 1995.
[56] J. Callan and M. Connell, "Query-based sampling of text databases," *ACM Trans. Inf. Syst.*, vol. 19, no. 2, pp. 97–130, 2001.
[57] C. Castillo, *Effective Web Crawling*. PhD thesis, Dept. of Computer Science – University of Chile, Santiago, Chile, November 2004.
[58] J. Caverlee, L. Liu, and J. Bae, "Distributed query sampling: a quality-conscious approach," in *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 340–347, ACM, 2006.
[59] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 554–560, ACM, 2006.

[60] Q. Chen, M. Li, and M. Zhou, "Improving query spelling correction using web search results," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (Prague, Czech Republic), pp. 181–189, Association for Computational Linguistic, June 2007.

[61] F. Chierichetti, A. Panconesi, P. Raghavan, M. Sozio, A. Tiberi, and E. Upfal, "Finding near neighbors through cluster pruning," in *Proceedings of ACM SIGMOD/PODS 2007 Conference*, 2007.

[62] P. A. Chirita, C. S. Firan, and W. Nejdl, "Personalized query expansion for the web," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 7–14, ACM, 2007.

[63] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe, "Collection statistics for fast duplicate document detection," *ACM Trans. Inf. Syst.*, vol. 20, no. 2, pp. 171–191, 2002.

[64] A. Cooper, "A survey of query log privacy-enhancing techniques from a policy perspective," *ACM Trans. Web*, vol. 2, no. 4, pp. 1–27, 2008.

[65] N. Craswell, P. Bailey, and D. Hawking, "Server selection on the world wide web," in *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, (New York, NY, USA), pp. 37–46, ACM, 2000.

[66] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey, "An experimental comparison of click position-bias models," in *WSDM '08: Proceedings of the international conference on Web search and web data mining*, (New York, NY, USA), pp. 87–94, ACM, 2008.

[67] S. Cucerzan and E. Brill, "Spelling correction as an iterative process that exploits the collective knowledge of web users," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, pp. 293–300, July 2004.

[68] S. Cucerzan and R. W. White, "Query suggestion based on user landing pages," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 875–876, ACM Press, 2007.

[69] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma, "Probabilistic query expansion using query logs," in *WWW '02: Proceedings of the 11th international conference on World Wide Web*, (New York, NY, USA), pp. 325–332, ACM, 2002.

[70] E. Cutrell and Z. Guan, "What are you looking for?: an eye-tracking study of information usage in web search," in *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, (New York, NY, USA), pp. 407–416, ACM, 2007.

[71] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[72] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," in *Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2003)*, pp. 89–98, 2003.

[73] Z. Dou, R. Song, and J. Wen, "A large-scale evaluation and analysis of personalized search strategies," in *Proceedings of the 16th international World Wide Web conference (WWW2007)*, pp. 572–581, May 2007.

[74] "Search engine users," White paper, 2005. `http://www.enquiroresearch.com/personalization/`.

[75] M. E.P., "On caching search engine query results," *Computer Communications*, vol. 24, pp. 137–143(7), 1 February 2000.

[76] T. Fagni, R. Perego, F. Silvestri, and S. Orlando, "Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data," *ACM Trans. Inf. Syst.*, vol. 24, no. 1, pp. 51–78, 2006.

[77] C. H. Fenichel, "Online searching: Measures that discriminate among users with different types of experience," *JASIS*, vol. 32, no. 1, pp. 23–32, 1981.

[78] P. Ferragina and A. Gulli, "A personalized search engine based on web-snippet hierarchical clustering," in *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, (New York, NY, USA), pp. 801–810, ACM, 2005.

[79] L. Fitzpatrick and M. Dent, "Automatic feedback using past queries: social searching?," in *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 306–313, ACM, 1997.

[80] B. M. Fonseca, P. B. Golgher, E. S. de Moura, and N. Ziviani, "Using association rules to discover search engines related queries," in *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*, (Washington, DC, USA), p. 66, IEEE Computer Society, 2003.

[81] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan-Kaufmann, 1999.

[82] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *J. Mach. Learn. Res.*, vol. 4, pp. 933–969, 2003.
[83] N. Fuhr, S. Hartmann, G. Knorz, G. Lustig, M. Schwantner, and K. Tzeras, "AIR/X - a rule-based multistage indexing system for large subject fields," in *Proceedings of the RIAO'91, Barcelona, Spain, April 2-5, 1991*, pp. 606–623, 1991.
[84] N. Fuhr, "Optimal polynomial retrieval functions based on the probability ranking principle," *ACM Trans. Inf. Syst.*, vol. 7, no. 3, pp. 183–204, 1989.
[85] N. Fuhr, "A decision-theoretic approach to database selection in networked ir," *ACM Trans. Inf. Syst.*, vol. 17, no. 3, pp. 229–249, 1999.
[86] G. P. C. Fung, J. X. Yu, P. S. Yu, and H. Lu, "Parameter free bursty events detection in text streams," in *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pp. 181–192, VLDB Endowment, 2005.
[87] G. W. Furnas, S. C. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum, "Information retrieval using a singular value decomposition model of latent semantic structure," in *SIGIR*, pp. 465–480, 1988.
[88] G. Galilei, "Discorsi e dimostrazioni matematiche intorno a due nuove scienze," Leida : Appresso gli Elsevirii, 1638.
[89] "The associated press: Internet ad revenue exceeds $21b in 2007," 2008. `http://ap.google.com/article/ALeqM5hccYd6ZuXTns2RWXUgh6br4n1UoQD8V1GGC00`.
[90] L. A. Granka, T. Joachims, and G. Gay, "Eye-tracking analysis of user behavior in www search," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 478–479, ACM, 2004.
[91] L. Gravano, H. Garcia-Molina, and A. Tomasic, "The efficacy of gloss for the text database discovery problem," Tech. Rep., Stanford University, Stanford, CA, USA, 1993.
[92] L. Gravano, H. García-Molina, and A. Tomasic, "The effectiveness of gioss for the text database discovery problem," in *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 126–137, ACM, 1994.
[93] L. Gravano, H. García-Molina, and A. Tomasic, "Gloss: text-source discovery over the internet," *ACM Trans. Database Syst.*, vol. 24, no. 2, pp. 229–264, 1999.
[94] L. Gravano, V. Hatzivassiloglou, and R. Lichtenstein, "Categorizing web queries according to geographical locality," in *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, (New York, NY, USA), pp. 325–333, ACM, 2003.
[95] Z. Guan and E. Cutrell, "An eye tracking study of the effect of target rank on web search," in *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, (New York, NY, USA), pp. 417–420, ACM, 2007.
[96] T. H. Haveliwala, "Topic-sensitive pagerank," in *WWW '02: Proceedings of the 11th international conference on World Wide Web*, (New York, NY, USA), pp. 517–526, ACM, 2002.
[97] D. Hawking, "Overview of the trec-9 web track," in *TREC*, 2000.
[98] D. Hawking, "Web search engines: Part 1," *Computer*, vol. 39, no. 6, pp. 86–88, 2006.
[99] D. Hawking, "Web search engines: Part 2," *Computer*, vol. 39, no. 8, pp. 88–90, 2006.
[100] D. Hawking and P. Thistlewaite, "Methods for information server selection," *ACM Trans. Inf. Syst.*, vol. 17, no. 1, pp. 40–76, 1999.
[101] J. Hennessy and D. Patterson, *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, 2003.
[102] M. R. Henzinger, "Algorithmic challenges in web search engines," *Internet Mathematics*, vol. 1, no. 1, 2003.
[103] M. R. Henzinger, R. Motwani, and C. Silverstein, "Challenges in web search engines," *SIGIR Forum*, vol. 36, no. 2, pp. 11–22, 2002.
[104] T. C. Hoad and J. Zobel, "Methods for identifying versioned and plagiarized documents," *Journal of the American Society for Information Science and Technology*, vol. 54, no. 3, pp. 203–215, 2003.
[105] I. Hsieh-Yee, "Effects of search experience and subject knowledge on the search tactics of novice and experienced searchers," *JASIS*, vol. 44, no. 3, pp. 161–174, 1993.
[106] S. T. I. Foster, C. Kesselman, "The Anatomy of the Grid: Enabling Scalable Virtual Organization," *Int'l Journal on Supercomputer Application*, vol. 3, no. 15.
[107] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.
[108] B. J. Jansen and M. Resnick, "An examination of searcher's perceptions of nonsponsored and sponsored links during ecommerce web searching," *J. Am. Soc. Inf. Sci. Technol.*, vol. 57, no. 14, pp. 1949–1961, 2006.

[109] B. J. Jansen and A. Spink, "An analysis of web searching by european alltheweb.com users," *Inf. Process. Manage.*, vol. 41, no. 2, pp. 361–381, 2005.
[110] B. J. Jansen and A. Spink, "How are we searching the world wide web? a comparison of nine search engine transaction logs," *Inf. Process. Manage.*, vol. 42, no. 1, pp. 248–263, 2006.
[111] B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic, "Real life information retrieval: a study of user queries on the web," *SIGIR Forum*, vol. 32, no. 1, pp. 5–17, 1998.
[112] B. J. Jansen, A. Spink, and S. Koshman, "Web searcher interaction with the dogpile.com metasearch engine," *JASIST*, vol. 58, no. 5, pp. 744–755, 2007.
[113] B. J. J. Jansen, "Understanding user-web interactions via web analytics," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 1, no. 1, pp. 1–102, 2009.
[114] T. Joachims, "Optimizing search engines using clickthrough data," in *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 133–142, ACM Press, 2002.
[115] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay, "Accurately interpreting clickthrough data as implicit feedback," in *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 154–161, ACM, 2005.
[116] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay, "Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search," *ACM Trans. Inf. Syst.*, vol. 25, no. 2, p. 7, 2007.
[117] T. Joachims, H. Li, T.-Y. Liu, and C. Zhai, "Learning to rank for information retrieval (lr4ir 2007)," *SIGIR Forum*, vol. 41, no. 2, pp. 58–62, 2007.
[118] T. Joachims and F. Radlinski, "Search engines that learn from implicit feedback," *Computer*, vol. 40, no. 8, pp. 34–40, 2007.
[119] K. S. Jones, S. Walker, and S. E. Robertson, "A probabilistic model of information retrieval: development and comparative experiments," *Inf. Process. Manage.*, vol. 36, no. 6, pp. 779–808, 2000.
[120] R. Jones, R. Kumar, B. Pang, and A. Tomkins, ""i know what you did last summer": query logs and user privacy," in *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, (New York, NY, USA), pp. 909–914, ACM, 2007.
[121] R. Jones, B. Rey, O. Madani, and W. Greiner, "Generating query substitutions," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 387–396, ACM Press, 2006.
[122] R. Karedla, J. S. Love, and B. G. Wherry, "Caching strategies to improve disk system performance," *Computer*, vol. 27, no. 3, pp. 38–46, 1994.
[123] M. Kendall, *Rank Correlation Methods*. Hafner, 1955.
[124] J. Kleinberg, "Bursty and hierarchical structure in streams," in *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 91–101, ACM, 2002.
[125] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, 1999.
[126] S. Koshman, A. Spink, and B. J. Jansen, "Web searching on the vivisimo search engine," *JASIST*, vol. 57, no. 14, pp. 1875–1887, 2006.
[127] M. Koster, "Aliweb: Archie-like indexing in the web," *Comput. Netw. ISDN Syst.*, vol. 27, no. 2, pp. 175–182, 1994.
[128] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 49–86, 1951.
[129] R. Kumar, J. Novak, B. Pang, and A. Tomkins, "On anonymizing query logs via token-based hashing," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (New York, NY, USA), pp. 629–638, ACM, 2007.
[130] T. Lau and E. Horvitz, "Patterns of search: analyzing and modeling web query refinement," in *UM '99: Proceedings of the seventh international conference on User modeling*, (Secaucus, NJ, USA), pp. 119–128, Springer-Verlag New York, Inc., 1999.
[131] U. Lee, Z. Liu, and J. Cho, "Automatic identification of user goals in web search," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*, (New York, NY, USA), pp. 391–400, ACM, 2005.
[132] R. Lempel and S. Moran, "Predictive caching and prefetching of query results in search engines," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*, (New York, NY, USA), pp. 19–28, ACM, 2003.
[133] R. Lempel and S. Moran, "Competitive caching of query results in search engines," *Theor. Comput. Sci.*, vol. 324, no. 2-3, pp. 253–271, 2004.

[134] R. Lempel and S. Moran, "Optimizing result prefetching in web search engines with segmented indices," *ACM Trans. Inter. Tech.*, vol. 4, no. 1, pp. 31–59, 2004.
[135] R. Lempel and F. Silvestri, "Web search result caching and prefetching," Encyclopedia of Database Systems, Springer Verlag, 2008. To Appear.
[136] M. Li, Y. Zhang, M. Zhu, and M. Zhou, "Exploring distributional similarity based models for query spelling correction," in *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, (Morristown, NJ, USA), pp. 1025–1032, Association for Computational Linguistics, 2006.
[137] Y. Li, Z. Zheng, and H. K. Dai, "Kdd cup-2005 report: facing a great challenge," *SIGKDD Explor. Newsl.*, vol. 7, no. 2, pp. 91–99, 2005.
[138] F. Liu, C. Yu, and W. Meng, "Personalized web search by mapping user queries to categories," in *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, (New York, NY, USA), pp. 558–565, ACM Press, 2002.
[139] Live Search Team at Microsoft, "Local, relevance, and japan!," `http://blogs.msdn.com/livesearch/archive/2005/06/21/431288.aspx`, 2005.
[140] X. Long and T. Suel, "Three-level caching for efficient query processing in large web search engines," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*, (New York, NY, USA), pp. 257–266, ACM, 2005.
[141] R. M. Losee and L. C. Jr., "Information retrieval with distributed databases: Analytic models of performance," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 1, pp. 18–27, 2004.
[142] C. Lucchese, S. Orlando, R. Perego, and F. Silvestri, "Mining query logs to optimize index partitioning in parallel web search engines," in *InfoScale '07: Proceedings of the 2nd international conference on Scalable information systems*, (New York, NY, USA), ACM, 2007.
[143] T.-Y. Lui, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, 2008.
[144] Y. Lv, L. Sun, J. Zhang, J.-Y. Nie, W. Chen, and W. Zhang, "An iterative implicit feedback approach to personalized search," in *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, (Morristown, NJ, USA), pp. 585–592, Association for Computational Linguistics, 2006.
[145] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 1999.
[146] M. Marchiori, "The quest for correct information on the web: Hyper search engines.," *Computer Networks*, vol. 29, no. 8-13, pp. 1225–1236, 1997.
[147] M. Mat-Hassan and M. Levene, "Associating search and navigation behavior through log analysis: Research articles," *J. Am. Soc. Inf. Sci. Technol.*, vol. 56, no. 9, pp. 913–934, 2005.
[148] O. A. McBryan, "Genvl and wwww: Tools for taming the web," in *Proceedings of the first International World Wide Web Conference*, (O. Nierstarsz, ed.), (CERN, Geneva), p. 15, 1994.
[149] S. Melink, S. Raghavan, B. Yang, and H. Garcia-Molina, "Building a distributed full-text index for the web," *ACM Trans. Inf. Syst.*, vol. 19, no. 3, pp. 217–241, 2001.
[150] T. Mitchell, *Machine Learning*. McGraw-Hill International Editions, 1997.
[151] A. Moffat, W. Webber, and J. Zobel, "Load balancing for term-distributed parallel retrieval," in *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 348–355, ACM, 2006.
[152] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates, "A pipelined architecture for distributed text query evaluation," *Inf. Retr.*, vol. 10, no. 3, pp. 205–231, 2007.
[153] A. Moffat and J. Zobel, "Information retrieval systems for large document collections," in *TREC*, pp. 0–, 1994.
[154] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "An optimality proof of the lru-k page replacement algorithm," *J. ACM*, vol. 46, no. 1, pp. 92–112, 1999.
[155] S. Orlando, R. Perego, and F. Silvestri, "Design of a Parallel and Distributed WEB Search Engine," in *Proceedings of Parallel Computing (ParCo) 2001 conference*, Imperial College Press, September 2001.
[156] H. C. Ozmutlu, A. Spink, and S. Ozmutlu, "Analysis of large data logs: an application of poisson sampling on excite web queries," *Inf. Process. Manage.*, vol. 38, no. 4, pp. 473–490, 2002.
[157] S. Ozmutlu, H. C. Ozmutlu, and A. Spink, "Multitasking web searching and implications for design," *JASIST*, vol. 40, no. 1, pp. 416–421, 2003.
[158] S. Ozmutlu, A. Spink, and H. C. Ozmutlu, "A day in the life of web searching: an exploratory study," *Inf. Process. Manage.*, vol. 40, no. 2, pp. 319–345, 2004.

[159] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Tech. Rep., Stanford Digital Library Technologies Project, 1998.
[160] S. Pandey and C. Olston, "User-centric web crawling," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*, (New York, NY, USA), pp. 401–411, ACM, 2005.
[161] S. Pandey and C. Olston, "Crawl ordering by search impact," in *WSDM '08: Proceedings of the international conference on Web search and web data mining*, (New York, NY, USA), pp. 3–14, ACM, 2008.
[162] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," in *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, (New York, NY, USA), p. 1, ACM, 2006.
[163] "Pew research center for the people & the press," WWW page, 2007. `http://people-press.org/`.
[164] J. Piskorski and M. Sydow, "String distance metrics for reference matching and search query correction," in *Business Information Systems, 10th International Conference, BIS 2007, Poznań, Poland, April 2007*, (W. Abramowicz, ed.), pp. 356–368, Springer-Verlag, 2007.
[165] J. Pitkow, H. Schütze, T. Cass, R. Cooley, D. Turnbull, A. Edmonds, E. Adar, and T. Breuel, "Personalized search," *Commun. ACM*, vol. 45, no. 9, pp. 50–55, 2002.
[166] B. Poblete, M. Spiliopoulou, and R. Baeza-Yates, "Website privacy preservation for query log publishing," in *First International Workshop on Privacy, Security, and Trust in KDD (PINKDD'07)*, August 2007.
[167] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.
[168] A. L. Powell and J. C. French, "Comparing the performance of collection selection algorithms," *ACM Trans. Inf. Syst.*, vol. 21, no. 4, pp. 412–456, 2003.
[169] A. L. Powell, J. C. French, J. Callan, M. Connell, and C. L. Viles, "The impact of database selection on distributed searching," in *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 232–239, ACM, 2000.
[170] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*. Cambridge University Press, second ed., 1992.
[171] D. Puppin, *A Search Engine Architecture Based on Collection Selection*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, December 2007.
[172] D. Puppin and F. Silvestri, "The query-vector document model," in *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, (New York, NY, USA), pp. 880–881, ACM, 2006.
[173] D. Puppin, F. Silvestri, and D. Laforenza, "Query-driven document partitioning and collection selection," in *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, (New York, NY, USA), p. 34, ACM, 2006.
[174] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates, "Tuning the capacity of search engines: Load-driven routing and incremental caching to reduce and balance the load," *ACM Trans. Inf. Syst.*
[175] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates, "Load-balancing and caching for collection selection architectures," in *InfoScale '07: Proceedings of the 2nd international conference on Scalable information systems*, (New York, NY, USA), ACM, 2007.
[176] F. Qiu and J. Cho, "Automatic identification of user interest for personalized search," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 727–736, ACM, 2006.
[177] F. Radlinski and T. Joachims, "Query chains: learning to rank from implicit feedback," in *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, (New York, NY, USA), pp. 239–248, ACM Press, 2005.
[178] F. Radlinski and T. Joachims, "Active exploration for learning rankings from clickthrough data," in *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 570–579, ACM, 2007.
[179] K. H. Randall, R. Stata, J. L. Wiener, and R. G. Wickremesinghe, "The link database: Fast access to graphs of the web," in *DCC '02: Proceedings of the Data Compression Conference (DCC '02)*, (Washington, DC, USA), p. 122, IEEE Computer Society, 2002.
[180] S. E. Robertson and S. Walker, "Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval," in *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 232–241, Springer-Verlag New York, Inc., 1994.
[181] S. E. Robertson and S. Walker, "Okapi/keenbow at trec-8," in *TREC*, 1999.
[182] J. T. Robinson and M. V. Devarakonda, "Data cache management using frequency-based replacement," *SIGMETRICS Perform. Eval. Rev.*, vol. 18, no. 1, pp. 134–142, 1990.
[183] J. Rocchio, *Relevance feedback in information retrieval*. Prentice-Hall, 1971.

[184] G. Salton and C. Buckley, "Parallel text search methods," *Commun. ACM*, vol. 31, no. 2, pp. 202–215, 1988.
[185] G. Salton and C. Buckley, "Improving retrieval performance by relevance feedback," *JASIS*, vol. 41, no. 4, pp. 288–297, 1990.
[186] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.
[187] M. Sanderson and S. T. Dumais, "Examining repetition in user search behavior," in *ECIR*, pp. 597–604, 2007.
[188] P. C. Saraiva, E. S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto, "Rank-preserving two-level caching for scalable search engines," in *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 51–58, ACM, 2001.
[189] F. Scholer, H. E. Williams, and A. Turpin, "Query association surrogates for web search: Research articles," *J. Am. Soc. Inf. Sci. Technol.*, vol. 55, no. 7, pp. 637–650, 2004.
[190] "Search engine use shoots up in the past year and edges towards email as the primary internet application," WWW page, 2005. `http://www.pewinternet.org/pdfs/PIP\_SearchData\_1105.pdf`.
[191] "Search engine users," WWW page, 2005. `http://www.pewinternet.org/pdfs/PIP\_Searchengine\_users.pdf`.
[192] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, 2002.
[193] D. Shen, R. Pan, J.-T. Sun, J. J. Pan, K. Wu, J. Yin, and Q. Yang, "Q^2c@ust: our winning solution to query classification in kddcup 2005," *SIGKDD Explor. Newsl.*, vol. 7, no. 2, pp. 100–110, 2005.
[194] X. Shen, B. Tan, and C. Zhai, "Ucair: a personalized search toolbar," in *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 681–681, ACM, 2005.
[195] M. Shokouhi, J. Zobel, and Y. Bernstein, "Distributed text retrieval from overlapping collections," in *ADC '07: Proceedings of the eighteenth conference on Australasian database*, (Darlinghurst, Australia, Australia), pp. 141–150, Australian Computer Society, Inc., 2007.
[196] M. Shokouhi, J. Zobel, S. Tahaghoghi, and F. Scholer, "Using query logs to establish vocabularies in distributed information retrieval," *Inf. Process. Manage.*, vol. 43, no. 1, pp. 169–180, 2007.
[197] L. Si and J. Callan, "Using sampled data and regression to merge search engine results," in *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 19–26, ACM, 2002.
[198] L. Si and J. Callan, "Relevant document distribution estimation method for resource selection," in *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, (New York, NY, USA), pp. 298–305, ACM, 2003.
[199] 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," *JASIS*, vol. 44, no. 5, pp. 273–291, 1993.
[200] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz, "Analysis of a very large altavista query log," Tech. Rep., Systems Research Center – 130 Lytton Avenue – Palo Alto, California 94301, 1998.
[201] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, "Analysis of a very large web search engine query log," *SIGIR Forum*, vol. 33, no. 1, pp. 6–12, 1999.
[202] F. Silvestri, *High Performance Issues in Web Search Engines: Algorithms and Techniques*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, May 2004.
[203] F. Silvestri, "Sorting out the document identifier assignment problem," in *Proceedings of the 29th European Conference on Information Retrieval*, April 2007.
[204] F. Silvestri, S. Orlando, and R. Perego, "Assigning identifiers to documents to enhance the clustering property of fulltext indexes," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 305–312, ACM, 2004.
[205] F. Silvestri, S. Orlando, and R. Perego, "Wings: A parallel indexer for web contents," in *International Conference on Computational Science*, pp. 263–270, 2004.
[206] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, 1985.
[207] A. J. Smith, "Cache memories," *ACM Comput. Surv.*, vol. 14, no. 3, pp. 473–530, 1982.
[208] M. Speretta and S. Gauch, "Personalized search based on user search histories," in *Web Intelligence*, pp. 622–628, 2005.
[209] A. Spink, B. J. Jansen, D. Wolfram, and T. Saracevic, "From e-sex to e-commerce: Web search changes," *Computer*, vol. 35, no. 3, pp. 107–109, 2002.

[210] A. Spink, S. Koshman, M. Park, C. Field, and B. J. Jansen, "Multitasking web search on vivisimo.com," in *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, (Washington, DC, USA), pp. 486–490, IEEE Computer Society, 2005.

[211] A. Spink, H. C. Ozmutlu, and D. P. Lorence, "Web searching for sexual information: an exploratory study," *Inf. Process. Manage.*, vol. 40, no. 1, pp. 113–123, 2004.

[212] A. Spink and T. Saracevic, "Interaction in information retrieval: Selection and effectiveness of search terms," *JASIS*, vol. 48, no. 8, pp. 741–761, 1997.

[213] A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic, "Searching the web: the public and their queries," *J. Am. Soc. Inf. Sci. Technol.*, vol. 52, pp. 226–234, February 2001.

[214] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan, "Web usage mining: Discovery and applications of usage patterns from web data," *SIGKDD Explorations*, vol. 1, no. 2, pp. 12–23, 2000.

[215] J. Teevan, E. Adar, R. Jones, and M. Potts, "History repeats itself: repeat queries in yahoo's logs," in *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 703–704, ACM, 2006.

[216] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts, "Information re-retrieval: repeat queries in yahoo's logs," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 151–158, ACM, 2007.

[217] J. Teevan, S. T. Dumais, and E. Horvitz, "Beyond the commons: Investigating the value of personalizing web search," in *Proceedings of Workshop on New Technologies for Personalized Information Access (PIA '05)*, (Edinburgh, Scotland, UK), 2005.

[218] J. Teevan, S. T. Dumais, and E. Horvitz, "Personalizing search via automated analysis of interests and activities," in *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 449–456, ACM Press, 2005.

[219] H. Turtle and J. Flood, "Query evaluation: strategies and optimizations," *Inf. Process. Manage.*, vol. 31, no. 6, pp. 831–850, 1995.

[220] M. van Erp and L. Schomaker, "Variants of the borda count method for combining ranked classifier hypotheses," in *Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition*, pp. 443 – 452, International Unipen Foundation, 2000.

[221] C. J. van Rijsbergen, *Information Retrieval*. London: Butterworths, 2nd ed., 1979.

[222] M. Vlachos, C. Meek, Z. Vagena, and D. Gunopulos, "Identifying similarities, periodicities and bursts for online search queries," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 131–142, ACM, 2004.

[223] M. Vlachos, P. S. Yu, V. Castelli, and C. Meek, "Structural periodic measures for time-series data," *Data Min. Knowl. Discov.*, vol. 12, no. 1, pp. 1–28, 2006.

[224] D. Vogel, S. Bickel, P. Haider, R. Schimpfky, P. Siemen, S. Bridges, and T. Scheffer, "Classifying search engine queries using the web as background knowledge," *SIGKDD Explor. Newsl.*, vol. 7, no. 2, pp. 117–122, 2005.

[225] X. Wang and C. Zhai, "Learn from web search logs to organize search results," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 87–94, ACM, 2007.

[226] R. Weiss, B. Vélez, and M. A. Sheldon, "Hypursuit: a hierarchical network search engine that exploits content-link hypertext clustering," in *HYPERTEXT '96: Proceedings of the the seventh ACM conference on Hypertext*, (New York, NY, USA), pp. 180–193, ACM, 1996.

[227] R. W. White, M. Bilenko, and S. Cucerzan, "Studying the use of popular destinations to enhance web search interaction," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 159–166, ACM, 2007.

[228] R. W. White, M. Bilenko, and S. Cucerzan, "Leveraging popular destinations to enhance web search interaction," *ACM Trans. Web*, vol. 2, no. 3, pp. 1–30, 2008.

[229] R. W. White and D. Morris, "Investigating the querying and browsing behavior of advanced search engine users," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 255–262, ACM, 2007.

[230] L. Xiong and E. Agichtein, "Towards privacy-preserving query log publishing," in *Query Log Analysis: Social And Technological Challenges. A workshop at the 16th International World Wide Web Conference (WWW 2007)*, (E. Amitay, C. G. Murray, and J. Teevan, eds.), May 2007.

[231] J. L. Xu and A. Spink, "Web research: The excite study," in *WebNet 2000*, pp. 581–585, 2000.

[232] J. Xu and J. Callan, "Effective retrieval with distributed collections," in *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 112–120, ACM, 1998.

[233] J. Xu and W. B. Croft, "Cluster-based language models for distributed retrieval," in *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 254–261, ACM, 1999.

[234] J. Xu and W. B. Croft, "Improving the effectiveness of information retrieval with local context analysis," *ACM Trans. Inf. Syst.*, vol. 18, no. 1, pp. 79–112, 2000.

[235] Yahoo! Grid, "Open source distributed computing: Yahoo's hadoop support," `http://developer.yahoo.net/blog/archives/2007/07/yahoo-hadoop.html`, 2007.

[236] Y. Yang and C. G. Chute, "An example-based mapping method for text categorization and retrieval," *ACM Trans. Inf. Syst.*, vol. 12, no. 3, pp. 252–277, 1994.

[237] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, "A support vector method for optimizing average precision," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 271–278, ACM, 2007.

[238] B. Yuwono and D. L. Lee, "Server ranking for distributed text retrieval systems on the internet," in *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*, pp. 41–50, World Scientific Press, 1997.

[239] O. R. Zaïane and A. Strilets, "Finding similar queries to satisfy searches based on query traces," in *OOIS Workshops*, pp. 207–216, 2002.

[240] J. Zhang and T. Suel, "Optimized inverted list assignment in distributed search engine architectures," in *IPDPS*, pp. 1–10, 2007.

[241] Y. Zhang and A. Moffat, "Some observations on user search behavior," in *Proceedings of the 11th Australasian Document Computing Symposium*, Brisbane, Australia, 2006.

[242] Z. Zhang and O. Nasraoui, "Mining search engine query logs for query recommendation," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 1039–1040, ACM, 2006.

[243] 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 *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 543–552, ACM, 2006.

[244] Z. Zheng, K. Chen, G. Sun, and H. Zha, "A regression framework for learning ranking functions using relative relevance judgments," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 287–294, ACM, 2007.

[245] G. K. Zipf, *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology.* Addison-Wesley, 1949.

[246] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Comput. Surv.*, vol. 38, no. 2, p. 6, 2006.