Università Ca' Foscari di Venezia

Dottorato di Ricerca in Informatica
Scuola di Dottorato in Scienze e Tecnologie - XXIII Ciclo
(A.A. 2010–2011)

Tesi di Dottorato
Settore Scientifico Disciplinare: INF/01

Enhancing Web Search User Experience: from Document Retrieval to Task Recommendation

Gabriele Tolomei
matricola n. 955515

Tutore del Dottorando
Prof. Salvatore Orlando

Direttore della Scuola
Prof. Paolo Ugo

Ottobre 2011

Pagina Web dell'autore: `http://miles.isti.cnr.it/~tolomei`

Posta elettronica dell'autore: tolomei@dsi.unive.it

Indirizzo dell'autore:

Dipartimento di Informatica
Università Ca' Foscari di Venezia
Via Torino, 155
30172 Venezia Mestre – Italia
tel. +39 041 2348411
fax. +39 041 2348419
web: `http://www.dsi.unive.it`

A Nonna Giordana

Abstract

The *World Wide Web* (i.e., Web) is the biggest and most heterogeneous database that humans have ever built, making it the place of choice where people look at whenever they come up with *any* sort of information need. Today, most of Web users put their trust in *Web search engines* for pursuing and satisfying their information needs.

Of course, modern Web search engines radically evolved since their first appearance almost fifteen years ago. Indeed, nowadays they provide a still growing set of capabilities for enhancing users' experience during the whole search process. Nevertheless, they are still in essence Web documents retrieval tools, namely Web-scale information retrieval systems, whereas users' expectations and needs are increasingly becoming more complex and heterogeneous. This trend is actually confirmed by a growing "addiction to Web search": no matter what an information need is, user is anyway brought to ask for it to a Web search engine, which will hopefully give the answer she expects. Moreover, there is a common belief that people are increasingly asking Web search engines also for accomplishing their daily *tasks* (e.g., "*planning holidays*", "*obtaining a visa*", "*organizing a birthday party*", etc.), instead of simply looking for Web pages of interest. Therefore, Web search engines have to face up to a group of emerging research challenges in order to coherently respond to such novel users' requests, thus to enhance search experience of users by really supporting them during their daily "Web lives".

The common aspect of all these challenges is that they strictly rely on an improved understanding of user search behaviors. A possible effective method for capturing how users interact with Web search engines comes from the analysis and mining of *query logs*. These are archives that record user search activities by means of streams of timestamped events (e.g., issued queries, click actions, etc.) and they could reveal the presence of valuable *usage patterns*.

This dissertation sketches two core issues for enabling next-generation Web search, which both represent fruitful applications of query log mining. From a side, we successfully apply query log mining techniques for discovering actual user search sessions from the raw stream of queries stored in Web search engine query logs, whose final aim is to perform *atomic* tasks (e.g., "*booking a hotel room*"). Somehow, for each user we are able to re-build every small search task she performed by partitioning, i.e., clustering, each stream of submitted requests into disjoint sets of

queries issued for achieving the same task. Furthermore, on the basis of the small search tasks discovered, we are able to address a second, more ambitious, challenge. Instead of considering user search process on a "*query-by-query*" perspective, we look at it from a higher level of abstraction, namely on a "*task-by-task*" perspective. To this end, we propose a model of search tasks for representing more complex user behaviors, while interacting with Web search engines. This model describes *what* small tasks are mostly searched for and *how* users typically "combine" them in order to achieve a more complex search task, i.e., a *mission*. Moreover, on top of such user search task model, we design a novel *task-oriented recommender system* that goes beyond traditional query suggestion. Given a portion of the query stream, the Web search engine may first recognize the actual small task behind that subsequence of queries. Finally, it generates lists of recommendations, which are not only related to the subsequence of queries that originated the suggestions, but also to a complex mission, which the small task could be part of, according to our model of users' search tasks.

Sommario

Il *World Wide Web* (Web) costituisce la più grande ed eterogenea sorgente di dati e contenuti realizzati ed usufruibili dall'uomo. Queste insolite caratteristiche, non riscontrabili in nessun'altra base di dati, hanno fatto sì che il Web divenisse sempre più il "luogo" di riferimento cui accedere per soddisfare qualsiasi sorta di bisogno informativo.
Ad oggi, la maggior parte degli utenti Web si affida completamente ai motori di ricerca Web per recuperare informazioni di interesse. A partire dalla loro prima apparizione sul panorama del Web circa quindici anni fa, i motori di ricerca sono certamente evoluti. Infatti, i più moderni motori di ricerca possono vantare un incredibile e crescente insieme di funzionalità, continuamente teso a migliorare l'esperienza di ricerca di ogni singolo utente. Nonostante ciò, i motori di ricerca Web restano essenzialmente "classici" sistemi di recupero di documenti, proiettati però su larga scala. D'altra parte, allo stesso modo le aspettative e le necessità informative degli utenti sono divenute sempre più complesse ed eterogenee. Questo andamento trova conferma in quella che si può definire un'autentica "dipendenza da ricerca Web": indipendentemente dalla specifica connotazione di un certo bisogno informativo, l'utente, prima o poi, è portato a rivolgersi al motore di ricerca di propria fiducia che, sperabilmente, fornirà la risposta attesa sia per genere che per contenuto. Inoltre, è opinione ormai ampiamente diffusa che le persone si rivolgano ai motori di ricerca Web non solo per accedere semplicemente a pagine Web di loro interesse, ma anche per eseguire le attività quotidiane più disparate (ad es., "*pianificare una vacanza*", "*ottenere un visto*", "*organizzare una festa di compleanno*", etc.). Di conseguenza, i motori di ricerca Web devono affrontare una serie di emergenti sfide di ricerca per poter rispondere in modo efficace ed esauriente alle nuove richieste dei propri utenti, cioè per poter fornire un supporto allo svolgimento di tutte quelle attività che ciascun individuo esegue quotidianamente sul Web.
Il fattor comune di tutte queste sfide è che esse dipendono fortemente da una decisa comprensione dei comportamenti di ricerca degli utenti. A questo proposito, un metodo per identificare i meccanismi di interazione tra gli utenti ed i motori di ricerca Web viene fornito dall'analisi dei *query logs*. I query logs sono archivi che memorizzano le attività di ricerca degli utenti sotto forma di sequenze di eventi (ad es., queries sottomesse al sistema, eventuali clicks su pagine risultato, etc.). Se opportunamente analizzati tramite note tecniche di data mining (*query log mining*),

essi possono rivelare la presenza di interessanti schemi d'uso, ovvero di consentire l'acquisizione di "nuova conoscenza".
Questa tesi affronta due temi fondamentali per realizzare i motori di ricerca Web di prossima generazione. Entrambi i temi rappresentano due vantaggiosi esempi di applicazioni del query log mining. Da un lato, abbiamo applicato con successo alcune tecniche di query log mining per estrarre sessioni di ricerca, memorizzate nei query logs, il cui scopo fosse quello di eseguire un'attività *atomica* (ad es., "*prenotazione camera albergo*"). Per ogni utente, siamo stati capaci di ricostruire ogni attività di ricerca che questo voleva eseguire attraverso il partizionamento (clustering) delle queries sottomesse e finalizzate a compiere una singola attività. Inoltre, sulla base delle attività di ricerca identificate con la tecnica descritta in precedenza, abbiamo affrontato la seconda e, per certi versi più ambiziosa, sfida. Anziché considerare il processo di ricerca dell'utente come una mera sequenza di queries successive ("*query-by-query*"), abbiamo cercato di analizzarlo ad un livello di astrazione più alto, ovvero come un insieme di attività ("*task-by-task*"). A questo scopo, abbiamo proposto un modello di attività di ricerca che rappresentasse comportamenti degli utenti anche complessi durante le interazioni con i motori di ricerca Web. Questo modello è in grado di descrivere *quali* attività atomiche sono più diffuse durante le ricerche degli utenti e *come* gli stessi utenti sono soliti "combinare" queste attività per eseguirne altre più complesse (*missioni*). Sulla base di questo modello delle attività di ricerca degli utenti, abbiamo progettato un nuovo sistema di raccomandazione *task-oriented* che supera il tradizionale schema del suggerimento di queries. Data una certa sottosequenza di queries, il motore di ricerca può essere in grado di riconoscere l'attività atomica che sottende la sottosequenza stessa. Infine, il sistema è in grado di generare insiemi di raccomandazioni che non sono collegati solamente all'attività atomica cui si riferisce la sottosequenza di queries iniziale, ma anche ad una o più attività che sono ad essa correlate e che concorrono alla realizzazione di una missione più complessa, sulla base del modello estratto a partire dallo storico delle attività di ricerca degli utenti.

Acknowledgments

If there wouldn't have been some old and new lovable people staying so close to me during these last four years, I truly couldn't have been able to come up with any of the ideas presented in this work.

Now, that we're all going to relax a little bit, I like thinking of all those people spreading around me, just waiting for me to speak something.

In the following few lines, I'll try to hug them all, one by one.

First, I'd like to thank my supervisor Salvatore Orlando who always supported my work even when it apparently seemed to be unfruitful. In the same way, Fabrizio Silvestri unconditionally encouraged me with his friendly and fraternal habits, thus convincing me of reaching the goals we planned to achieve.

My special thought goes to Domenico Laforenza and Raffaele Perego, which are the former and current heads of the High Performance Computing Lab at ISTI-CNR, respectively. They welcomed me on board and now I'm so proud to be part of this "troop". To this end, I'd like also to thank other colleagues of mine, especially those who shared the most with me other than work troubles and pains: Franco Maria, Gabriele, and Diego have been, and I hope they always will be, much more than colleagues. I want to say them: "Hey guys, C-64 will be forever with me!".

Anyway, if today I'm here, I've to be grateful to Fabio, a schoolmate, a friend, and lastly a colleague. He perfectly understood my pains coming from my past job and he suddenly translated them into precious advices for my future.

I want to hug my parents and all my family. Together, they have been able to support my choice simply because they blindly put their trust on me and because they wanted to see me really happy. In particular, I dedicate all my efforts to my grandmother Giordana, who unfortunately is not be able to share this joy with me due to her conditions. Besides, during these years I've been delighted by the happiness that only a small baby could give: my niece Martina.

And now that this path seems to come to an end, I'm looking forward to starting a brand new journey, walking hand in hand with someone who appeared when everything seemed to be broken... I don't simply give my thanks to Sara, I'm gonna give her all the days of my life.

Ringraziamenti

Nessuna delle idee contenute in queste pagine si sarebbe potuta materializzare se, durante questi ultimi quattro anni di cammino, non avessi avuto vecchi e nuovi compagni di viaggio disposti a sopportare i ritmi e l'incedere dei miei passi. Adesso che insieme ci stiamo fermando per riprendere fiato ed assaporare un po' di meritato riposo, mi piace immaginarli tutti lì, sparsi qua e là intorno a me, pronti ad aspettare che sia io a dire la prima parola o a fare il primo gesto.

In queste poche righe cerco di abbracciarli tutti, uno ad uno.

Desidero innanzitutto ringraziare il mio supervisore Salvatore Orlando che si è adoperato con infaticabile dedizione per sostenere costantemente il mio lavoro, anche quando questo sembrava non trovare una collocazione che fosse riconosciuta ed apprezzata dalla comunità scientifica. Allo stesso modo, seppur con un approccio più amichevole e fraterno, ma non per questo meno duro ed efficace, sono consapevole che non avrei raggiunto questo traguardo se Fabrizio Silvestri, con i suoi incondizionati incoraggiamenti, non avesse quotidianamente tentato di controbilanciare la mia innata tendenza ad abbattermi quando le cose non andavano come avrebbero dovuto, donandomi quell'equilibrio e quella serenità indispensabili per raggiungere gli obiettivi che insieme ci eravamo prefissati.

Dedico un pensiero speciale a Domenico Laforenza e Raffaele Perego, rispettivamente ex ed attuale responsabile dell'High Performance Computing Lab dell'ISTI-CNR, che hanno saputo accogliermi nella loro "truppa" fino al punto di farmi sentire, oggi, parte integrante di questo gruppo. Per questo, ringrazio anche tutti i colleghi di laboratorio e, in particolare, quelli con cui ho maggiormente condiviso momenti di vita che sono andati ben oltre la semplice sfera professionale: Franco Maria, Gabriele e Diego sono stati, e spero rimangano, molto più che semplici colleghi di lavoro quanto piuttosto quegli esempi reali di amicizia che nasce tra i banchi di scuola. A loro voglio dire: "La C-64 resterà sempre con me!".

Se oggi sono qui lo devo soprattutto a Fabio, compagno di università, amico ed infine collega, che ha saputo leggere le frustrazioni e le insoddisfazioni del mio precedente lavoro e le ha tradotte in consigli ed indicazioni per il mio futuro.

Abbraccio i miei genitori e tutta la mia famiglia, che insieme hanno saputo assecondare la mia scelta di tornare studente fidandosi in modo cieco e quasi incosciente delle mie capacità di sapermi costruire un percorso che, per quanto complicato e rischioso, avrebbe potuto rendermi più felice. In particolare, dedico tutti gli sforzi

di questo lavoro a mia nonna Giordana, i cui avversi segni del tempo che suo malgrado è costretta a portare su di sé non le consetiranno di gioire come meriterebbe. D'altra parte, in questi anni ho potuto assaporare il gusto della spensieratezza che solo una piccola creatura sa donare: mia nipote Martina.

E adesso, che questo cammino sembra concluso, non vedo l'ora di iniziarne uno nuovo per mano a colei che aspettavo da sempre e che è arrivata proprio quando tutto sembrava non funzionare... A Sara non vanno solo i miei ringraziamenti, a lei donerò i miei giorni.

Contents

List of Figures

List of Tables

1 Introduction

"The Web is more a social creation than a technical one. I designed it for a social effect to help people work together and not as a technical toy. The ultimate goal of the Web is to support and improve our weblike existence in the world."

Tim Berners-Lee
Weaving The Web (1999)

The *World Wide Web* (i.e., Web) is the biggest and most heterogeneous database that humans have ever built, making it the place of choice where people look at whenever they come up with *any* sort of information need.

However, this great repository of data as we know it today is really far away from the original idea of the Web as it was first conceived in 1989 at the *Conseil Européen pour la Recherche Nucléaire* (CERN) by Sir Tim Berners-Lee. Basically, Berners-Lee presented a document called *"Information Management: A Proposal"* [27], in which he expressed the need for a *simple protocol* that could request information stored in remote systems through networks and for a *scheme* by which such information could be exchanged in a common format and documents could be linked to each other by *hyperlinks*.

More simply, the Web was born as a distributed computer platform that allows academic and research people connecting to each other, thus sharing and exchanging data stored across several computing systems in a *standardized* way via the world-wide computer network, also referred to as the *Internet*.

The *raison d'être* of the Web relies on the structure of its composing *hypertext* documents, i.e., *Web pages*, which allows authors to link their documents to other *related* documents stored in computers anywhere across the Internet.

Quickly, the Web has unexpectedly become an extraordinary *socioeconomic phenomenon* because of two main impact factors: *users* and *contents*.

From a side, an increasing number of end users, which were not necessarily involved in academia or research activities, have rapidly started to have access to the Web as well because personal computers became more affordable and the Internet infrastructure and services spread to wider geographical areas. According to latest

2011 Internet statistics, there are more than two billion of Web users all around the world[1]. Moreover, during the 90's there has been a proliferation of a broad spectrum of applications running on the Web, which were mostly targeted to "common" people and also to business enterprises, thus resulting in a growing number of Web contents.

However, in this first Web era there was still a clear separation of roles between *few* content providers, i.e., typically skilled workers and professionals, and *many* content consumers, i.e., common end users. During the last years, a new trend has gained momentum: new "*social-oriented*" applications that allow easy authoring and content creation have lead to an increased *democratization* and *collaborative involvement* in the Web (e.g., Wikipedia, Flickr, YouTube, Facebook, Twitter, etc.). Somehow, this process caused the end of the first Web era, by bringing down the wall between content providers and consumers, which now can play both roles interchangeably from time to time. Therefore, information available on the Web have started raising at a tremendous rate, reaching nowadays a huge and still growing number of contents, which spread over several media types (e.g., text, images, audio/video, etc.).

In order to have access to such a huge collection of data in a feasible and effective way, Web users have been provided with specific applications, i.e., *Web search engines.*

On one side, Web search engines aim at building a *back-end* infrastructure, where Web contents may be collected and managed both effectively and efficiently. Such infrastructure is in turn exploited by the *front-end*, i.e., an interface which users rely on for searching and retrieving contents they are interested in.

Although the huge number of features which now the most popular Web search engines come with, in essence they still belong to the category of Web documents retrieval tools. The typical interaction between a user and a search engine is based on the well-known "*query-look-refine*" paradigm, which is indeed proper of the traditional *Information Retrieval* (IR) domain. First, user formulates her information need by phrasing a natural language query, i.e., a list of keywords, then she submits it to the search engine throughout its user-friendly interface. Therefore, search engine exploits its back-end infrastructure for retrieving a list of Web contents that are considered relevant to the user's intent. Finally, user looks at one or more *Search Engine Results Pages* (SERPs), i.e., lists of "*ten blue links*", thereby either she clicks on some of the retrieved results or she re-phrases a brand new query in order to better specify her needs.

This paradigm of interaction is effective whenever the need behind a user query is to find information on topics of interest contained in Web pages. However, there is a common belief that people are increasingly asking Web search engines also for accomplishing their daily *tasks* (e.g., "*planning holidays*", "*obtaining a visa*", "*organizing a birthday party*", etc.), in an easier way. This claim is strongly supported

[1]`http://www.internetworldstats.com/stats.htm`

by the most authoritative people in the Web search domain. To this end, in 2008 Peter Norvig from *Google*[2] and Prabhakar Raghavan from *Yahoo!*[3] agreed that:

> "*People intrinsically don't want to search. People don't come to work every day saying 'I need to search'... They want to run their lives!*" [4]

At the same time, Ricardo Baeza-Yates from *Yahoo!* said that:

> "*People want to get tasks done!*" [5]

Furthermore, this trend is actually confirmed by a growing "addiction to Web search": no matter what an information need is, user is anyway brought to ask for it to a Web search engine, which will hopefully give the answer she expects.

Anyway, whenever the user's need behind a certain query is a task to be accomplished, the traditional "*query-look-refine*" paradigm should be improved for "driving" the user towards the execution of her desired task. Hence, we believe next-generation Web search engines should provide new features and capabilities to support users in their everyday activities, thus to enhance the overall user search experience.

Of course, this opens up novel and exciting research challenges, ranging from the ability to recognize the tasks behind user queries, to the design of new recommendation strategies as well as new user interfaces for showing relevant results. Some of these challenges represent the core of the whole research roadmap devised during the very first stage of Ph.D. activities. To this end, a sketch of the research presented in this dissertation was also initially proposed to the scientific community. In particular, we came up with a research proposal submitted to two Doctoral Consortia, namely the 3rd ACM Recommender Systems Doctoral Consortium (RecSys '09) and the 26th IEEE International Conference on Data Engineering Ph.D. Workshop (ICDE '10) [132, 133].

In this way, we were able to evaluate the validity and the strength of our research ideas, which were indeed positively judged by "offline" feedbacks coming from many expert computer scientists. In fact, the submitted proposal was successfully accepted to both events, thereby presented at the corresponding Conferences. There, other "online" comments and suggestions from many valuable researchers helped us to better clarify each step of our research roadmap and they were decisive for leading us to the results we achieved and that here we describe in the rest of this dissertation.

[2]`http://www.google.com`
[3]`http://www.yahoo.com`
[4]`http://blogoscoped.com/archive/2008-11-06-n63.html`
[5]`http://www.cwr.cl/la-web2008/slides/Wisdom-of-Crowds-long.pdf`

1.1 Contribution

The main contribution of this work concerns two distinct yet related topics, namely *(i)* the discovery of *search tasks* that users look for when interacting with Web search engines and *(ii)* the development of a novel *task-oriented* recommender system that goes beyond traditional *query suggestion.*

The first aspect, namely the discovery of user search tasks, is completely covered by Chapter 4 and it is based on the work proposed in three research papers, i.e., "*Detecting Task-based Query Sessions Using Collaborative Knowledge*" [81], "*Identifying Task-based Sessions in Search Engine Query Logs*" [83], and "*Discovering User Tasks in Long-Term Web Search Engine Logs*" [82].

This Chapter describes how we tackle the problem of identifying "*logical*" search sessions from the stream of user queries stored in *query logs* of Web search engines. Each logical search session may be considered as a set of queries issued by a certain user for accomplishing a specific task, i.e., *task-oriented session.* To this end, we first built, by means of a manual labeling process, a *ground-truth* where the queries of a real query log have been grouped in tasks. Our analysis of this ground-truth shows that users tend to interleave several search tasks within relatively small units of time, since about 75% of the submitted queries involve a *multi-tasking* activity.

Moreover, we define the *Task-oriented Session Discovery Problem* (TSDP) as the problem of best approximating the above ground-truth. The TSDP deals with two issues: *(i)* a robust measure of the *task relatedness* between any two queries, i.e., *task-based query similarity*, and *(ii)* an effective method for actually discovering task-oriented sessions by using the above measure of task relatedness.

Concerning *(i)*, we propose and compare both *unsupervised* and *supervised* approaches for devising several task-based query similarity functions. These functions also exploit the collaborative knowledge collected by *Wiktionary*[6] and *Wikipedia*[7] for detecting query pairs that are not similar from a lexical content point of view, but actually *semantically* related.

Finally, we tackle *(ii)* by introducing a set of *query clustering* methods that exploit the above similarity functions for detecting user tasks. All the proposed solutions were evaluated on the ground-truth and two of them have shown to perform better than state-of-the-art approaches.

Chapter 5 presents the second challenge addressed in this dissertation, which is based on the the work proposed in the research paper "*Beyond Query Suggestion: Recommending Tasks to Search Engine Users*" [84] as an evolutionary step of the first research topic discussed above. Basically, it concerns the investigation of a novel recommendation strategy that goes beyond traditional query suggestion mechanisms provided by modern Web search engines. In fact, current query sug-

[6]`http://www.wiktionary.org`
[7]`http://www.wikipedia.org`

gestion mechanisms are able to recommend to users lists of alternative queries that better specify their actual information needs.

So far, suggestions generated for a certain query are all related the *same* search task expressed by that original query. For instance, a user issuing the query "`new york hotel`" typically is provided with suggestions such as "`cheap new york hotels`", "`times square hotel`", "`waldorf astoria`", etc. Although these recommended queries are all related to the original query, they still refer to the same search task, i.e., "*finding a hotel room in New York*".

However, it might be the case that the "bigger" search need, i.e., *mission*, behind such first user query is to "*plan a travel to New York*". Therefore, if the Web search engine would be able to make this prediction, i.e., to conjecture the global mission that user is willing to perform, we could generate suggestions that are not *only* strictly related to her current task but that also refer to *other* tasks being part of the mission as a whole. In our example, this means recommending queries like "`mta subway`", or "`broadway shows`", or "`jfk airport shuttle`".

In this Chapter, we propose an innovative recommender application, namely *task-oriented* recommender system, which generates *task suggestions* on the basis of a model of user search tasks learned from the historical activities recorded in query logs. Experimental results show that our solution is able to suggest queries belonging to tasks that are different from the ones that users are currently performing but related to those which users are going to look for in the next future. Moreover, even when generated recommendations are not strictly part of future users' searches, we show that they could anyway represent *surprising* hints.

1.2 Organization

The dissertation is organized as follows. Chapter 2 presents an overview of Web search engines both from functional and architectural perspectives. Chapter 3 contains a survey of the main state-of-the-art contributions on the analysis of query logs collected by Web search engines as a method for extracting useful and valuable knowledge. In particular, we focus on discussing the statistical properties of various examples of real-world query logs. Moreover, we describe how query log analysis may be exploited in order to improve both the effectiveness and efficiency of Web search engines as well as to enhance user search experience. Chapter 4 is completely dedicated to the problem of discovering user search tasks from the stream of queries stored in query logs of Web search engines. In Chapter 5, we investigate how search tasks discovered by using the methodology described in Chapter 4 may be exploited for proposing a new recommendation model, which goes beyond traditional query suggestion schemes. Finally, Conclusions summarizes the work presented in this dissertation and points out some future challenges.

2 Web Search Engines

"Una volta un tale che doveva fare una ricerca andava in biblioteca, trovava dieci titoli sull'argomento e li leggeva; oggi schiaccia un bottone del suo computer, riceve una bibliografia di diecimila titoli, e rinuncia."
(Once, a person looking for something was used to go to the library, there she found ten subjects and simply read them; now, the same person clicks on a button of a PC, gets back ten thousands references then she gives up.)

Umberto Eco
La Bustina di Minerva (2000)

The smallest *unit of information* addressable on the Web is a *resource* identified by a proper "name", i.e., a URL[1], which typically points to a *Web page* or to any other kind of Web entity stored at a specific location, i.e., a *Web site*. Each Web resource is delivered through a specific protocol, i.e., HTTP[2], from a *Web server* where the resource is physically hosted to a *Web client*(i.e., usually a Web browser).

The content of a Web resource can be either *static* or *dynamic*, namely generated "*on-the-fly*" by the Web server depending on the client request. Anyway, whether the resource is static or dynamic it may contain unstructured data (e.g., ASCII plain text), as well as semi-structured or structured data (e.g., HTML or XML files).

There are three important factors that affect data and contents on the Web:

- ***size:*** there is a huge and still growing number of Web contents, which now can be directly "pushed" by almost every user at any time;
- ***heterogeneity:*** contents vary significantly among existing Web resources (e.g., text, images, audio/video, etc.);
- ***dynamism:*** new Web resources and applications rapidly appear while others disappear as well, so that available Web contents change very quickly from time to time.

[1]Uniform Resource Locator (`http://www.faqs.org/rfcs/rfc1738.html`)
[2]HyperText Transfer Protocol (`http://www.faqs.org/rfcs/rfc2616.html`)

As the total number of Web contents increases, a feasible way for retrieving and accessing them becomes necessary. In the last fifteen years, this resulted in the development of new software systems and applications, which represent the default "Web portal" for the users. Typically, people try to satisfy their *information needs*, namely their intents, through a specific category of such systems, called *Web search engines*. Therefore, Web search engines have become a sort of universally accepted *interface* to the whole information contained on the Web.

Although the huge number of features which now the most popular Web search engines come with, in essence they are still part of a broader class of popular and well-established software systems, namely *Information Retrieval* (IR) systems. Such systems were born in the early 1960s in order to respond to two main needs, namely *(i)* to allow people searching through remote digital libraries and *(ii)* to allow end users searching through the data stored in their own local digital repositories.

An IR system is a software whose main purpose is to return a list of documents in response to a user query. Somehow, this makes IR systems similar to *DataBase* (DB) systems. However, the most important difference between DB and IR systems is that DB systems return objects that *exactly* match the user query, which in turn is formulated using a standardized and "*structured*" dialect, i.e., SQL[3]. Besides, IR systems allow users to phrase "*unstructured*" natural language queries that makes it almost impossible to return perfect matches. As an example, the keyword "*kobe*" could either refer to Kobe Bryant (one of the best and most famous NBA basketball player), to a particular cuts of beef, or even to a Japanese city. Therefore, every single query may mean several things to several users.

Web search engines inherit from IR systems the well-known "*query-look-refine*" scheme that governs the interaction between users and those kinds of systems [26]. According to this paradigm, a user first formulates her information need by phrasing a natural language query, namely a list of keywords that she believes to be the most suitable for representing her intents, thus for retrieving relevant results. Then, user submits her query to the Web search engine throughout its user-friendly interface. Therefore, the Web search engine exploits its back-end infrastructure for retrieving a list of Web contents, which are assumed to be relevant to the user's intent. Finally, user scans and looks at one or more *Search Engine Results Pages* (SERPs), i.e., lists of "*ten blue links*" to Web pages, thereby either she clicks on some of the retrieved results or she re-phrases a brand new query in order to better specify her needs.

Many Web search engines adopt classical IR algorithms and techniques in order to implement their functionalities. Such approaches have proven to be very effective in the traditional IR domain, where information to collect and among which users look for is relatively *small*, *static*, and *homogeneous*. Besides, the size, dynamism, and heterogeneity of the Web make IR techniques as they are almost unsuitable for the Web context. Thereby, traditional IR algorithms have to be extended, and

[3]Structured Query Language (`http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45498`)

sometimes brand new techniques must be thought from scratch. In that way, the *scalability*, *manageability*, *efficiency*, and *effectiveness* of Web retrieval systems are improved, moving from classical IR search engines to modern Web search engines.

The first systems similar to modern Web search engines started to operate around 1994. To this extent, *World Wide Web Worm* (WWWW) [90] created by Oliver McBryan at the University of Colorado and the *AliWeb* search engine [72] created by Martijn Koster in 1994, are the two most famous examples. Since then, many examples of such systems have appeared on the Web, e.g., *AltaVista*, *Excite*, *Lycos*, *Yahoo!*, *Google*, *Ask*, *MSN*, etc.

Nowadays, *Web search* is considered one of the most useful and used application as well as one of the most exciting computer science research topic. To this end, Baeza-Yates *et al.* [7] overview and highlight the challenges in the design of modern Web search engines, stating that:

> "*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.*"

Thereby, two key performance indicators for this kind of application are: *(i)* the quality of returned results and *(ii)* the speed with which results are returned.

2.1 The Big Picture

A Web search engine is one of the most complicated software to develop. Consisting of tens of interdependent modules, it represents a big challenge in today's computer engineering world.

Many research works and books sketch the architecture of Web search engines. To this extent, Barroso *et al.* [20] present the architecture of Google as it was in 2003. Other Web search engines are supposed to have similar architectures.

According to [30, 86], a Web search engine is responsible for three main activities: *(i) crawling*, *(ii) indexing*, and *(iii) query processing.*

Every modern Web search engine is usually designed according to a modular architecture in order to accomplish all those activities above [5]. Typically, the architecture of a large-scale search engine is composed of several *modules* that work together and cooperate in a structured way, as depicted in Figure 2.1.

In the following Sections, we describe the three activities that a Web search engine is assumed to realize as well as how the modules implementing such activities interact to each other. Before going into further detail, we set up some useful notations that are used all along the rest of this Chapter.

We start by indicating with $W = \{w_1, w_2, \ldots, w_{|W|}\}$ the set of all the available Web resources, i.e., Web pages. Moreover, we define U to be the set of known Web URLs, i.e., $U = \{u_1, u_2, \ldots, u_{|U|}\}$. Since each $w_i \in W$ is referenced by *at least* one

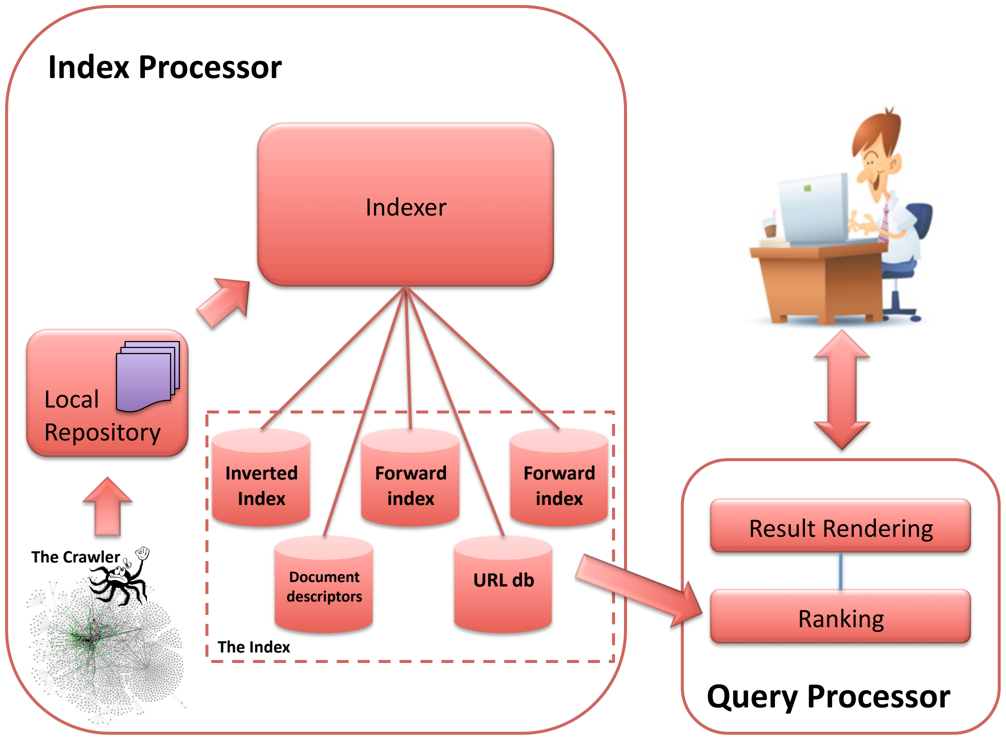


Figure 2.1: The typical architecture of a Web search engine [120].

URL, it usually holds that $|W| \leq |U|$. In fact, it is not uncommon to have two syntactically distinct URLs referring exactly the same Web page (i.e., *spider trap*).

Therefore, in order to map each URL to a specific Web page, there should exist an *onto* function $f : U \longmapsto W$, such that $f(u_j) = w_i$. However, for the sake of simplicity, here we instead assume that there exists a *one-to-one correspondence* g between the sets U and W, i.e., $g : U \longmapsto W$, such that $g(u_i) = w_i$. Nevertheless, although sometimes URLs and Web pages can be used both interchangeably, still they have to be considered as separate objects.

Moreover, $U(i) \subseteq U$ represents the set of URLs contained in a Web page $w_i \in W$, namely the set of *outgoing links* of w_i. Similarly, $\mathcal{O}(i) \subseteq W$ and $\mathcal{I}(i) \subseteq W$ are the sets of Web pages which $w_i \in W$ points to and are pointed by, respectively, that is:

$$\mathcal{O}(i) = \{g(u_j) \mid u_j \in U(i) \wedge g(u_j) \neq w_i\},$$

$$\mathcal{I}(i) = \{g(u_h) \mid u_i \in U(h) \wedge g(u_h) \neq w_i\}.$$

Finally, we define $V = \{t_1, t_2, \dots, t_{|V|}\}$ to be the *vocabulary* of terms contained in the collection of Web pages W.

2.2 Crawling

The *crawling* activity is performed by the *Crawler Module* (CM), which represents the basis on top of which any other operation is performed. The CM is composed of small programs, called *crawlers* (or also *wanderers*, *robots*, *spiders*, etc.), that “browse” the Web on the search engine’s behalf, in the same way in which human users follow links to reach several Web pages.

Typically, a crawler retrieves Web pages starting from an initial set of *seed* URLs, i.e., $U_{seed} \subseteq U$. Then, it downloads and copies the Web pages associated with each URL of the seeds to a local repository, i.e., the *Web Page Repository* (WPR). Furthermore, the crawler fetches all the URLs contained in each just stored Web page and adds them to a list of unvisited URLs, called *crawl frontier*, later on indicated with Φ.

More formally, the crawler first places the seeds in the crawl frontier, where all the URLs to be retrieved are kept and eventually *prioritized*, i.e., $\Phi = U_{seed}$. Then, it starts the so-called *crawling loop* phase, which consists of the following steps:

1. ***picking*** a URL $u_j \in \Phi$ in some order, following a policy specified by a dedicated *Crawler Control Module* (CCM);
2. ***fetching*** the corresponding Web page, i.e., $w_i = g(u_i)$;
3. ***parsing*** w_i to extract the set of URLs in it contained, i.e., $U(i)$;
4. ***adding*** the new set of unvisited URLs to the crawl frontier, i.e., $\Phi = \Phi \cup U(i)$.

The *crawling loop* phase is repeated for a specific number of times, until the crawler decides to stop. However, in modern Web search engines, crawlers continuously execute this phase, thus updating incrementally the content of their local repository.

According to [97], in the following Sections we sketch the main activities, which a crawler is responsible for, i.e., the steps of the *crawling algorithm*. In addition, we present a detailed description of the main components, i.e., *modules*, a crawler is usually made of, and *data structures* it uses for implementing its mechanisms.

2.2.1 The Crawling Algorithm

The core of the whole crawling process is represented by the crawling algorithm, which describes the behavior and the choices of the crawling phase with respect to the following four main issues:

- ***What Web pages should a crawler download?***

  In most cases, crawlers cannot together download *all* the pages available on the Web. So, a crawler algorithm must be designed to carefully select which Web pages have to be fetched, thus to visit “important” Web pages first by properly prioritizing the URLs contained in the frontier Φ. This *selection policy* is managed and specified by the *Crawler Control Module* (CCM) and it may vary among different Web search engines. As an example, some crawlers might be biased to download as many Web pages as possible, stressing their goal on *exhaustivity*, while other crawlers might specialize on Web pages belonging to certain domain, stressing their goal on *focusing*. The first case represents a *traditional* crawler (*breadth-first crawler*), while the second represents a so-called *focused* or *topic* crawler (*best-first crawler* or its variations). Anyway, the fraction of the Web that is visited and kept up-to-date should be as more *meaningful* as possible.

- ***How should a crawler refresh Web pages?***

  Once a crawler has downloaded a significant number of Web pages, it has to start to revisiting them in order to detect possible changes and refresh the downloaded collection. Because Web pages changing at very different rates, the crawler needs to carefully decide what Web pages to revisit and what to skip, thereby affecting significantly the *freshness* of the downloaded collection.

- ***How should the load on visited Web sites be minimized?***

  Clearly, when the crawler retrieves and collects Web pages it consumes resources belonging also to other organizations. As an example, when the crawler requests to download a (static) Web page hosted on a remote machine, that machine needs to get the requested document from its local filesystem, thus consuming disk and CPU resources. Moreover, after the remote machine has retrieved the document from its filesystem, such Web page still needs to be transferred through the network, which again is another resource shared among different entities. The crawler should be designed also to minimize its impact on these kinds of resources.

- ***How should the crawling process be parallelized?***

  Web's size is very huge and so crawlers often run on multiple machines and download pages in parallel. This kind of parallelization is necessary in order to download a large number of pages in a reasonable amount of time, increasing the *throughput*. Clearly, parallel crawlers must be coordinated properly, so that different crawlers do not visit the same Web site multiple times.

2.2.2 The Crawl Frontier

The *crawl frontier* Φ represents the "*to-do list*" of a crawler and it contains URLs of unvisited Web pages. Although it may be necessary to store the frontier on disk for large-scale crawlers, it can be considered as an in-main memory data structure by setting a maximum frontier size.

The frontier can be implemented as a simple *FIFO queue* to which corresponds a *breadth-first crawler* that can be used to blindly browse the Web. The URL u_j to be next fetched and visited is the head of the queue, while any new URLs are added to the tail of the queue.

Due to the limited size of Φ, it must be avoided to add duplicate URLs into the frontier. A linear search to find out if a newly extracted URL is already in the frontier is costly. Therefore, one solution is to allocate some amount of available memory to maintain a separate *hash table* with URL as key in order to store each URL currently in the frontier for fast lookup operations. Obviously, this hash table has to be kept synchronized and consistent with the current frontier. Another solution is to implement the frontier Φ itself as a hash table, again with the URL as key. However, each time the crawler needs a URL to visit, it would need to search and pick the URL with the earliest timestamp, that is the time at which the URL was added to the frontier.

If the frontier is implemented as a *priority queue*, the corresponding crawler is a so-called *preferential crawler*, also known as *best-first crawler*. The priority queue can be a dynamic array that is always kept sorted by the estimated score assigned to unvisited URLs. At each step, the best URL u_i is picked from the head of the queue. Once the corresponding Web page $w_i = g(u_i)$ is fetched, all the URLs which w_i contains, i.e., $U(i)$, are extracted and scored basing on some *heuristic*. Those URLs are then added to the frontier in such a way that the order of the priority queue is maintained.

The whole crawling process stops whenever the crawler finds the frontier empty when attempting to fetch the next URL. Anyway, with high values of the maximum frontier size this situation is quite uncommon.

Another classical problem derives from so-called *spider trap*. A spider trap leads the crawler to a large number of syntactically different URLs which all refer to the same Web page. One simple way for alleviating such problem is limiting the number of Web pages that the crawler accesses from a given domain. The code associated with the frontier can make sure that each consecutive sequence of k URLs picked by the crawler contains *only* one URL from a fully-qualified host name. As a consequence, the crawler is polite by not accessing the same Web site too often [43] and the crawled Web pages tend to be more heterogeneous.

2.2.3 Web Page Fetching

The *fetching* step of the whole crawling process consists in requesting and downloading a Web page associated with a specific URL.

In order to fetch a Web page, a HTTP client who sends a HTTP request for that Web page and reads the received response is needed. Obviously, the HTTP client must set timeouts properly to make sure it does not waste too much time for connecting and/or downloading a Web page from a remote Web server.

Typically, the client downloads only the first 10-20 kilobytes of the Web page, then it parses the `HTTP response` header for status codes and redirections and possibly the `last-modified` header parameter to determine the age of the document.

It is worth to mention a particular issue of the fetching step, that is the *Robot Exclusion Protocol.* This protocol provides a mechanism for Web server administrators to communicate their file access policy, more specifically to identify files that may not be accessed by a crawler. This is done by keeping a file named `robots.txt` under the root directory of the Web server. This file provides access policy for different *user-agents* (robots or crawlers). For example, a `user-agent` value of "*" denotes a default policy for any crawler that does not match other `user-agent` values in the file. Moreover, a number of `disallow` entries may be provided for a user-agent. Any URL that starts with the value of a `disallow` field must not be retrieved by a crawler matching the corresponding `user-agent`.

When a crawler wants to download a Web page from a Web server, it must first fetch the appropriate `robots.txt` file and make sure that the URL to be fetched is not disallowed. It is efficient to cache the access policies of a number of recently visited Web servers, avoiding accessing `robots.txt` file each time a URL must be fetched.

2.2.4 Web Page Parsing

Once a Web page has been downloaded, it is necessary to *parse* its content in order to extract information that will feed and possibly drive the future choices in the crawling process. Parsing may simply consist of hyperlink/URL extraction or it may involve the more complex process of *tidying up* the HTML content for analyzing the HTML *tag tree* [96].

2.2.4.1 URL Extraction and Normalization

HTML parsers are easily available for several programming languages. They basically provide the functionality to identify HTML tags, associating attribute-value pairs in a given HTML document.

In order to extract hyperlink URLs from a certain Web page, those parsers can be used to find anchor tags and grab the values of associated `href` attributes, converting any relative URLs to absolute ones using the base URL of the Web page from which they were retrieved.

As previously said, *spider traps*, that is syntactically different URLs corresponding to the same Web page, represent a serious problem affecting the performance of the overall crawling process. In order to avoid fetching the same Web page following syntactically different URLs, a normalization (or canonicalization) procedure is needed. Some of the typical steps in URL normalization procedure involve:

– converting the protocol and the hostname to lowercase;

– removing the "anchor" or "reference" part of the URL ("`#`");

– performing URL encoding for some commonly used character such as "∼";

– adding trailing slash "`/`" basing on some heuristics;

– using heuristics to recognize default Web page (e.g., removing from URLs file names such as `index.html` or `index.htm`);

– removing from the URL path "`..`" and its parent directory;

– leaving the port numbers in the URL unless it is the default HTTP port number 80 or, as an alternative, leave the port numbers in the URL and add port number 80 when no port number is specified.

Anyway, the only thing which is really important in URL normalization is to keep and preserve consistency when applying normalization rules.

2.2.4.2 Stoplisting and Stemming

When parsing the content of a Web page it is often helpful to remove commonly used words or *stopwords*[4] (e.g., "it", "can", etc.). The procedure of removing stopwords from a text is called *stoplisting.* In addition to stoplisting, it is also useful to "stem" the terms found in the Web page; the *stemming* procedure normalizes terms by conflating a number of morphologically similar terms to a single root form or *stem.* As an example of stemming procedure, the english terms "connect", "connected", and "connection" are all reduced to the stem "connect". One of the most popular english stemming procedure is provided by the *Porter stemming* algorithm, which was introduced by Martin Porter in 1980 [102].

2.2.5 Web Page Storing

Once a Web page is fetched and downloaded it has to be stored/indexed in order to be used by the master application, such as a Web search engine.

The *Web Page Repository* (WPR) is a scalable storage system for managing such large collections of Web pages and performs two main functions: *(i)* it provides an interface for crawlers of the CM to store Web pages they download and *(ii)* it supplies an efficient *Application Programming Interface* (API), which both the *Indexer Module* (IM) and the *Collection Analysis Module* (CAM) may use to retrieve Web pages (Section 2.3).

In its simplest form, the WPR may store the crawled Web pages as separate files and in that case, each Web page must map to a unique file name. One way

[4]for an example list of stopwords refer to `http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words`

to obtain that scheme is to map each URL corresponding to a Web page to a compact string using some form of *hash function* with low probability of collision. The resulting hash value is then used as the file name. To this end, the MD5[5] is a widely used one-way hash function that provides a 128 bit *fingerprint* for each URL, thus addressing 2^{128} possible file names. This fingerprint is "non-reversible", meaning that it is computationally infeasible, i.e., hard, to determine a file name, which the fingerprint originated from.

2.3 Indexing

The *Indexer Module* (IM) and the *Collection Analysis Module* (CAM) build a variety of indexes on the set of Web pages crawled by the CM. The IM builds two basic indexes: a *Text* (or *Content*) *Index* and a *Link* (or *Structure*) *Index*. Such indexes are useful data structures which Web search engines rely on for selecting Web pages relevant to specific user queries among the overall crawled collection of documents stored on the *Web Page Repository* (WPR).

2.3.1 Text-based Indexing

The primary method for identifying relevant Web pages to a certain user query still derives from the classical *text-based retrieval*. This method has been widely adopted in traditional IR contexts and it is based on searching for Web pages containing the *keywords* formulated in the query. Indexes to support such text-based retrieval can be implemented using any of the access methods typically used to search over text document collections. Examples include *suffix arrays*, *inverted indexes* or *inverted files*, and *signature files*. In Web domain, *inverted indexes* have become the index structure of choice.

The rest of this Section describes the *Text Index* and, in particular, it addresses the problem of quickly and efficiently building inverted indexes over Web-scale document collections.

2.3.1.1 Index Structure

The name *inverted index* is actually quite redundant because an index always maps back from terms to the portion of a document where they occur.

The basic idea behind an inverted index is to keep a *dictionary* D of terms (sometimes also referred to as *vocabulary* or *lexicon*). Let us use *dictionary* D to indicate the data structure and *vocabulary* $V = \{t_1, t_2, \ldots, t_{|V|}\}$ for representing the set of terms of the whole collection of Web pages $W = \{w_1, w_2, \ldots, w_{|W|}\}$.

Each term $t_i \in V$ is associated with a list, called *posting list* (or *inverted list*), that records which Web page $w_j \in W$ the term t_i occurs in. Each element of such

[5]`http://tools.ietf.org/html/rfc1321`

list is called *posting* and, in its simplest form, it only stores the unique identifier of the Web page containing the specific term t_i. However, the generic posting for the term t_i often stores other information, such as the frequency of t_i in w_j or the position(s) where t_i appears in w_j. All the *posting lists* taken together are referred to as *postings*.

Thus, an inverted (text) index over the collection of Web pages W consists of two parts: the dictionary D, which stores all the terms of the vocabulary V, and the corresponding postings.

Finally, the dictionary D has to be lexicographically-sorted, and the posting lists associated with each term must also be sorted by Web page identifier.

2.3.1.2 Index Partitioning

In real-world search engines, Web-scale inverted index requires a highly scalable and distributed architecture to improve the overall system efficiency (i.e., throughput, query response time, etc.) when processing a huge amount of incoming user queries [20].

Typically, the index is distributed among a set of **k** *query servers* coordinated by a *broker*, as showed in Figure 2.2.

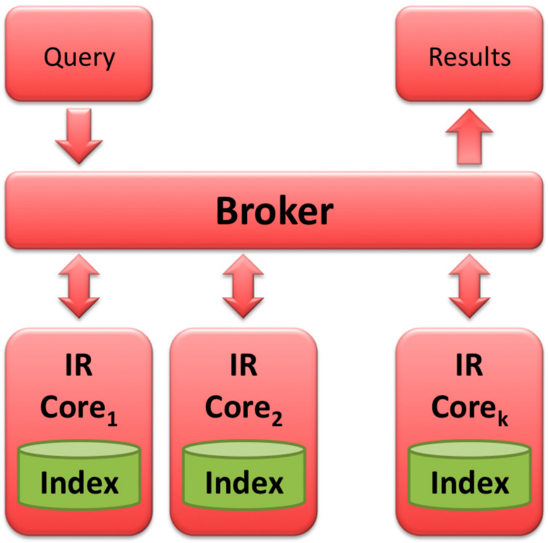


Figure 2.2: The typical structure of a distributed Web search engine [120].

The broker, accepts a user query and distributes it to the set of query servers. Thus, the index servers retrieve relevant Web pages, compute scores, rank results and return them back to the broker which renders the result page and sends it to the user.

Basically, there are two strategies for partitioning the inverted index across the set of query servers, i.e., *local inverted file* and *global inverted file*. In the *local* *inverted file* organization, each query server is responsible for a disjoint subset of Web pages of the whole collection, thus resulting in a partition over the Web pages (i.e., *document partitioning*).

In the *global inverted file* organization, index terms are partitioned, so that each query server stores inverted lists only for a subset of the terms in the collection, thus resulting in a partition over the index terms (i.e., *term partitioning*).

Finally, the broker is also the place where user activities (e.g., submitted queries, clicked results, etc.) are stored in files called *query logs*.

2.3.2 Link-based Indexing

Differently from traditional text documents, Web pages have also hypertext links for connecting them to each other. This information, namely the presence on a Web page w_i of a link pointing to another Web page w_j, could be useful for determining the quality and relevance of retrieved results.

To this end, the crawled portion of the Web is modeled as a directed graph $G = (W, E)$, where W is the set of vertexes (or nodes) and E is the set of edges. Each vertex in the Web graph corresponds to a Web page w_i and a directed edge between two vertexes (w_i, w_j) represents a hypertext link on page w_i that points to w_j. A *Link Index* on that structure is a *scalable* and *efficient* representation of the Web graph G.

The most common structural information that can be derived from such link index and used in turn by search algorithms is *neighborhood information*. Given a certain Web page w_i, it is easy to retrieve the set $\mathcal{O}(i)$ of Web pages which w_i points to (*outgoing pages*), or the set $\mathcal{I}(i)$ of Web pages pointing to w_i (*incoming pages*).

2.4 Query Processing

The typical interaction between a user and a Web search engine starts with the formulation of a natural language *query*, which represents a certain user's *information need*. It is worth noting that the information need is different from the query: the former is the topic about which the user desires to know more about, while the latter is what the user conveys to the system in an attempt to communicate her request.

A user query q consists of a list of *terms*, each one representing a specific *keyword*:

$$q = \{t_1, t_2, \dots, t_{|q|}\}.$$

Usually, q is composed of a little number of terms [62, 118, 122, 65], i.e., $|q|$ is about 2.5 on average.

Once a user has submitted a query q, the Web search engine starts its searching process in the background. First, the *Query Engine Module* (QEM) accesses to the back-end infrastructure (e.g., *Text* and *Link Indexes*, *Web Page Repository*, etc.). Eventually, it retrieves a single, uncategorized list of k result Web pages, i.e., $R_k(q) = \langle w_1, w_2, \ldots, w_k \rangle$, which are *ranked* according to their *relevance* to the guessed user's need, that is Web pages appearing in the top positions of $R_k(q)$ are also the most likely to be *relevant* to the user query q. Usually, $k = 10$ and the list of retrieved Web pages are rendered and displayed on user's screen as a single page containing the k URLs pointing to the corresponding results, i.e., *Search Engine Results Page* (SERP) or "*ten blue links*".

Anyway, before user is presented with the final ranked list of results $R_k(q)$, the following procedure has to take place. Indeed, $R_k(q)$ contains a very small portion of all the crawled Web pages W, i.e., $k \ll |W|$. In order for the search engine to select the k ultimate results that best match the guessed user's need, it has to look at each Web page in the collection. To this end, a first rough list $R(q)$ composed of all the candidate Web pages containing the keywords specified in q has to be retrieved. However, $|R(q)|$ is usually very large because of the huge number of available Web pages.

Therefore, $R(q)$ has to be *ranked* by an appropriate component, i.e., the *Ranking Module* (RM), which sorts $R(q)$ so that Web pages near the top of the list are the most likely ones that represent what the user is actually looking for. Once $R(q)$ has been sorted, a sublist $R_k(q)$ with the first k elements of $R(q)$ is finally returned to the user.

Sorting results by relevance requires establishing how much each Web page is relevant to the query issued by the user. More formally, given W the domain of all the available Web pages, and Q the domain of all possible queries, each Web page $w_i \in W$ must be associated with a *relevance score*, which is evaluated through a proper function *rel*, as follows:

$$rel : W \times Q \longmapsto \mathbb{R}^+ \cup \ \{0\}.$$

Each relevance score $rel(w_i, q)$ is in turn used for ranking the ultimate result list $R_k(q)$, so that:

- $|R_k(q)| = k$;
- $rel(w_i, q) \geq rel(w_{i+1}, q) \ \forall i \in \{1, 2, \ldots k-1\}$;
- $w_i \in R_k(q) \Rightarrow rel(w_i, q) \geq rel(w_j, q) \ \ \forall w_i \in R_k(q), \forall w_j \in R(q) \setminus R_k(q)$.

Roughly, there are two approaches for computing the relevance score of a Web page with respect to a issued query: *Text-based* and *Link-based*. The first one is a popular technique in traditional IR and it is based only on the lexical content of the Web page and the query, whereas the second one is typical of the Web domain and it exploits the hyperlinks connecting Web pages to each other.

2.4.1 Text-based Ranking

In order to sort the retrieved result list $R_k(q)$, early Web search engines only adopted a *text-based ranking scheme*, which measures the relevance score of a Web page $w \in W$ with respect to a query q as the *textual similarity* between w and q. Typically, the textual similarity between w and q is measured by using traditional IR approaches. According to such approaches, both w and q are viewed as *text documents* and each document is treated as a "*bag of words*", i.e., a set of distinct terms.

In traditional IR, given $V = \{t_1, t_2, \ldots, t_{|V|}\}$ a vocabulary of terms over a collection of text documents $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{D}|}\}$, the generic document $d_j \in \mathcal{D}$ can be represented by a n-dimensional *weighted term vector*, i.e., $\mathbf{d_j}$, where $n = |V|$. Moreover, the i-th component of such vector quantifies the level of importance of the term $t_i \in V$ in the document d.

Several weighting schemes have been proposed for evaluating each component of a term vector associated with a given document d. To this end, a first attempt relies on the well-known IR *boolean model* [112].

Using this model, $\mathbf{d_j}$ can be represented as a *binary vector*, i.e., $\mathbf{d_j} \in \{0,1\}^n$, and its i-th component is obtained as follows:

$$\mathbf{d_j}[i] = \begin{cases} 1 & \text{if the term } t_i \text{ appears in } d_j, \\ 0 & \text{otherwise.} \end{cases}$$

The boolean model only records term presence or absence inside the document, anyway it should be arguable to give more "weight" to documents where the term t_i appears several times as opposed to ones that contain it only once.

Therefore, a better approach to build the vector $\mathbf{d_j}$ could be obtained with the best known and most widely used IR *vector space model* [112]. Following this model, each document is represented as a weighted vector, in which each component weight is properly computed. The value $\mathbf{d_j}[i]$ for term t_i in document d_j is no longer in $\{0,1\}$ as in the boolean model, but it can be any real number, that is $\forall i \in \{1, 2, \ldots, n\}$, $\mathbf{d_j}[i] \in \mathbb{R}$, namely $\mathbf{d_j} \in \mathbb{R}^n$.

Again, the vector $\mathbf{d_j}$ depends on the documents d_j from which it is built. If the term t_i does not appear in a document d_j, the corresponding component $\mathbf{d_j}[i]$ is 0. On the other hand, if t_i appears in d_j, the corresponding component in the document vector $\mathbf{d_j}[i]$ represents the *significance* of the term.

One common way to compute the significance of a term, namely the value $\mathbf{d_j}[i]$, is given by the *term frequency* scheme (tf). In this scheme, the value $\mathbf{d_j}[i]$ coincides with the number of times t_i occurs in d_j and it is indicated with $tf(t_i, d_j)$.

The shortcoming of the tf scheme is that it does not consider the situation where a term appears in many documents of the collection. In order to address also that situation, it is needed a mechanism for *smoothing* the effect of terms that occur too often in the collection to be meaningful for determining their significance. A first idea could be to scale down the weights of terms with high *collection frequency* (cf), which is defined to be the total number of occurrences of a term t_i in the whole collection of documents $\mathcal{D}$:

$$cf(t_i, \mathcal{D}) = \sum_{d_k \in \mathcal{D}} tf(t_i, d_k).$$

Thus, each vector component can be scaled down as follows:

$$\mathbf{d_j}[i] = \frac{tf(t_i, d_j)}{cf(t_i, \mathcal{D})} = \frac{tf(t_i, d_j)}{\sum_{d_k \in \mathcal{D}} tf(t_i, d_k)}.$$

However, instead of cf it is more commonplace to use the *document frequency* (*df*), defined to be as the number of documents in the collection $\mathcal{D}$ that contain a specific term t_i and indicated with $df(t_i)$. The *df* scheme is used to scale down the weight of a term t_i, through the *inverse document frequency* (*idf*). Given $|\mathcal{D}|$ the total number of documents in the collection $\mathcal{D}$, the *idf* is computed as follows:

$$idf(t_i) = \log \frac{|\mathcal{D}|}{df(t_i)}.$$

Therefore, each vector component can be scaled down following the *tf-idf* weighting scheme:

$$\mathbf{d_j}[i] = tf\text{-}idf(t_i, d_j) = tf(t_i, d_j) \times idf(t_i) = tf(t_i, d_j) \times \log \frac{|\mathcal{D}|}{df(t_i)}.$$

In other words, the *tf-idf* weighting scheme assigns to a term t_i appearing in a document d_j a weight that is:

- ***highest*** when t_i occurs many times within a small number of documents, thus lending *high discriminating power* to those documents;
- ***lower*** when t_i occurs fewer times in d_j, or occurs in many documents, thus offering a less pronounced relevance signal;
- ***lowest*** when t_i virtually occurs in all documents of the collection.

Now, let us turn back to the representations of the Web page $w \in W$ and the query q according to the vector space model and following the *tf-idf* weighting scheme, as described before. Each document associated with w and q is represented by a weighted term vector, i.e., $\mathbf{w}$ and $\mathbf{q}$. Moreover, let us indicate with $tf(t_i, w)$ the number of occurrences of the term t_i in the Web page w and let analogously $tf(t_i, q)$ counts the number of times the term t_i appears in the query q. Also, $idf(t_i)$ is the inverse document frequency for the term t_i in the Web page collection W. Given $W = \{w_1, w_2, \ldots, w_{|W|}\}$, the i-th component of $\mathbf{w}$ and $\mathbf{q}$ is computed as follows:

$$\mathbf{w}[i] = tf\text{-}idf(t_i, w) = tf(t_i, w) \times idf(t_i) = tf(t_i, w) \times \log \frac{|W|}{df(t_i)},$$

$$\mathbf{q}[i] = tf\text{-}idf(t_i, q) = tf(t_i, q) \times idf(t_i) = tf(t_i, q) \times \log \frac{|W|}{df(t_i)}.$$

Therefore, the textual similarity between w and q, i.e., the relevance score $rel(w, q)$, is represented by the *dot product* between the two corresponding vectors $\mathbf{w}$ and $\mathbf{q}$, that is:

$$\mathbf{w} \cdot \mathbf{q} = \sum_{i=1}^{n} \mathbf{w}[i] \cdot \mathbf{q}[i] = \mathbf{w}[1] \cdot \mathbf{q}[1] + \mathbf{w}[2] \cdot \mathbf{q}[2] + \ldots + \mathbf{w}[n] \cdot \mathbf{q}[n]. \tag{2.1}$$

Using the Euclidian geometry interpretation, the dot product between two vectors $\mathbf{w}$ and $\mathbf{q}$ is strictly related to their *length* (or *magnitude*), i.e., $\|\mathbf{w}\|$, $\|\mathbf{q}\|$, and the *angle* θ between them, namely:

$$\mathbf{w} \cdot \mathbf{q} = \|\mathbf{w}\| \cdot \|\mathbf{q}\| \cdot \cos\theta. \tag{2.2}$$

Consequently, the measure of textual similarity between two documents codified as vectors assumes maximum value when *cos* $\theta = 1$, that is when $\theta = 2k\pi$ $(k \in \mathbb{N})$. Intuitively, that situation indicates that the two vectors have the *same* direction.

On the opposite side, the textual similarity assumes its minimum value when *cos* $\theta = -1$, that is when $\theta = \pi + 2k\pi$ $(k \in \mathbb{N})$, namely when the two vectors have *opposite* directions.

Finally, when *cos* $\theta = 0$, that is $\theta = \frac{\pi}{2} + k\pi$ $(k \in \mathbb{N})$ then the whole product is 0. Intuitively, that situation indicates that the two vectors have *orthogonal* directions.

In real-world Web search engines, the state-of-the-art tf-idf-like ranking function is represented by $BM25$, which was first introduced by Robertson *et al.* and that it is also known as "*Okapi* $BM25$" [110].

2.4.2 Link-based Ranking

Unfortunately, traditional text-based IR techniques may not be effective in ranking query results when dealing with Web-scale domain. First, Web is very large, with great variation in the amount, quality, and type of information contained in Web pages. Thus, many Web pages that contain the search terms might be of poor quality or not relevant at all. Second, many Web pages are not sufficiently *self-descriptive*, so the IR techniques examining only the contents of a Web page might not effectively work.

Moreover, Web pages are frequently altered by adding misleading terms so that they are ranked higher by a Web search engine (i.e., *spamming*). Therefore, approaches that are based only on the content of Web pages can easily be manipulated.

Fortunately, the *hyperlink structure* of the Web hides important implicit information and can help in filtering or ranking Web pages, thus in providing a valuable relevance scoring function. In particular, the existence of a link from a Web page

w_i to another w_j can be viewed as a sort of *recommendation* of Web page w_j by the author(s) of w_i.

To this end, two interesting link-based approaches are *PageRank* [30] and *HITS* [70], which in the following are described in more detail. Both these approaches rely on the notion of *Web graph*, as it was already introduced in Section 2.3.2. This is a graph $G = (W, E)$, where W is the set of Web pages and a directed edge between two vertexes (w_i, w_j) represents a hypertext link on page w_i that points to w_j.

2.4.2.1 PageRank

The *PageRank* algorithm proposed by Brin and Page [30] defines a *global ranking scheme*, which tries to capture the notion of "*importance*" of a Web page. The key parameter affecting the importance of a certain Web page w_i is reflected in the number of other Web pages w_j that point to w_i. Thus, the rank of w_i could be defined as the number of Web pages in the Web that point to w_i and could be used to rank results of a search query. However, citation ranking itself does not work well especially against spamming, since it is quite easy to artificially create a lot of Web pages to point to a desired target Web page.

For that reason, PageRank extends the classic citation ranking idea by taking into consideration the importance of the Web pages that point to a given Web page. Therefore, a Web page receives more importance if the Web pages pointing to it are important as well. In contrast, classical citation ranking does not make this distinction. This means that the definition of PageRank is *recursive*, namely the importance of a Web page both *depends on* and *affects* the importance of other Web pages.

Simple PageRank. Let us first introduce a simple definition of PageRank, before describing a practical variant of it.

Given $W = \{w_1, w_2, \dots, w_{|W|}\}$ the set of Web pages in the collection, let $\mathcal{O}(i)$ be the set of Web pages which w_i points to and let $\mathcal{I}(i)$ be the set of Web pages that point to w_i (Section 2.1).

Furthermore, let us assume that Web pages form a strongly connected graph, that is for each Web page w_i it always exists a *path* on the graph G, such that w_i can reach any other Web page w_j.

Therefore, the simple PageRank for a Web page w_i, i.e., $PR(w_i)$, can be defined as follows:

$$PR(w_i) = \sum_{w_j \in \mathcal{I}(i)} PR(w_j)/|\mathcal{O}(j)|. \tag{2.3}$$

The division by $|\mathcal{O}(j)|$ captures the intuition that Web pages which point to w_i evenly distribute their rank boost to all of the Web pages they point to.

Practical PageRank. Simple PageRank is well-defined only if the Web graph G is strongly connected. However, the Web is far from being strongly connected. In particular, there are two related problems arising on the real Web, namely *rank sinks* and *rank leaks*.

Any strongly connected cluster of Web pages within the Web graph from which no links point outwards forms a *rank sink*. An individual page that does not have any outward links constitutes a *rank leak*. In the case of *rank sink*, nodes which are not part of the sink receive a zero rank, meaning that it is not possible to distinguish the importance of such nodes. Besides, any rank reaching a *rank leak* is lost forever.

A possible solution is to remove all the leak nodes with out-degree = 0. Second, a decay factor δ, such that $0 < \delta < 1$, is introduced in the original PageRank definition. In this modified version, only a fraction δ of the rank of a Web page is distributed among the nodes that it points to, while the remaining rank is distributed equally among all the other Web pages in the collection. Thus:

$$PR(w_i) = \delta \times \sum_{w_j \in \mathcal{I}(i)} PR(w_j)/|\mathcal{O}(j)| + (1-\delta)/|W|, \tag{2.4}$$

where $|W|$ is the total number of nodes in the graph, that is the total number of Web pages in the collection.

Using PageRank for keyword searching. The Google search engine uses both classical IR techniques, i.e., *text-based*, and PageRank, i.e., *link-based* approaches, to answer users requests via keyword queries. Given a query $q = \{t_1, t_2, \ldots, t_{|q|}\}$ the Google search engine computes a text-based relevance score for all the Web pages which contains the query terms of q. This score is then combined with the PageRank of these Web pages in order to determine the final rank for this query results.

2.4.2.2 HITS

The *Hypertext Induced Topic Search* (i.e., *HITS*) is another important link-based ranking algorithm. In contrast to the PageRank algorithm, which assigns a global rank to every Web page, the HITS algorithm is a *query dependent* ranking technique. Moreover, instead of producing a single ranking score, HITS generates two scores: the *authority score* and the *hub score*.

Web pages classified as *authority* are those most likely to be relevant to a particular query, whereas *hub* Web pages are not necessarily authorities themselves but they are required to point to several authority Web pages.

There are basically two reasons why hub Web pages are interesting. First, they are used in the HITS algorithm to compute the authority Web pages. Second, hub Web pages represent themselves a useful set of documents to be returned to the user in response to a query [37].

There exists a mutually reinforcing relationship between the hubs and the authorities: an authority Web page is pointed to by many hubs, whereas hubs are Web pages pointing to many authorities.

The HITS algorithm. The basic idea of the HITS algorithm is to identify a small subgraph G' of the Web graph G, such that $G' \subset G$, and apply link analysis on this subgraph G' to locate the authority and hub Web pages for a given query. The subgraph that is chosen depends on the user query and, since both the subgraph selection and its analysis are done at query-time, it is important to perform them as quick as possible.

Link Analysis. Let the set of Web pages in the *focused subgraph* G' be denoted as W'. The link analysis algorithm produces an *authority score*, i.e., $a(i)$, and a *hub score*, i.e., $h(i)$, for each Web page $w'_i \in W'$. At the beginning, both $a(i)$ and $h(i)$ are initialized to arbitrary values. The algorithm is iterative and performs two kinds of operations at each step, called I and O, respectively. In the I operation, the authority score of each Web page is updated to the sum of the hub scores of all the Web pages pointing to it, that is:

$$a(i) = \sum_{w_j \in \mathcal{I}(i)} h(j). \tag{2.5}$$

In the O step, the hub score of each Web page is updated to the sum of the authority scores of all the Web pages that it points to, that is:

$$h(i) = \sum_{w_j \in \mathcal{O}(i)} a(j). \tag{2.6}$$

The above definitions capture the intuition that a good authority Web page is pointed to by many good hub Web pages and, conversely, that a good hub Web page points to many good authority Web pages. Obviously, a Web page may be, and often is, *both* a hub and an authority.

3 Query Log Mining

"La science avec des faits comme une maison avec des pierres; mais une accumulation de faits n'est pas plus une science qu'un tas de pierres n'est une maison."
(Science is built up of facts, as a house is built of stones; but an accumulation of facts is no more a science than a heap of stones is a house.)

Jules-Henri Poincaré
La Science et l'Hypothèse (1901)

In Chapter 2, we described the main functionalities and sketched the general architecture of a real-world Web search engine. Moreover, we caught how the overall quality of such a complex software system is affected by its capability in retrieving results that are relevant to user intents expressed by search queries.

Unfortunately, a key problem of modern Web search engines is that the actual needs behind user queries are very hard to recognize and understand. Indeed, keyword-based queries are not always an effective descriptor of user search intents. A first issue occurs because the ability of each single user to phrase effective queries highly depends on her subjective skills. It might be the case that two users, having exactly the same search intent, formulate two completely different queries, on the basis of their familiarity with the specific terminology of the knowledge domain. Moreover, even well-formulated and effective queries may suffer of ambiguity issues that arise in many terms of a language, thus resulting in retrieved Web pages which are not relevant to what users are really searching for. This phenomenon is even more remarkable if we consider that users typically submit very short queries, i.e., 2.5 terms on average [62, 118, 122, 65], which are more likely to be ambiguous.

However, differently from smaller scale IR systems, Web search engines can rely on a huge amount of historical usage data stored in their *query logs*. Such logs, indeed, represent a valuable source of knowledge for enhancing both the *effectiveness* and *efficiency* of Web search systems.

To this end, *Query Log Mining* is concerned with all the techniques aiming to discover interesting *patterns* from query logs with the purpose of enhancing the overall provided service. Somehow, query log mining may be considered as a branch of the more general *Web Analytics* scientific discipline [59], which, according to the Web Analytics Association, is defined as:

> "*The measurement, collection, analysis and reporting of Internet data for the purposes of understanding and optimizing Web usage.*" [6]

Moreover, query log mining represents a special application of *Web Usage Mining* [127], which generally refers to the discovery of user access patterns from Web usage logs.

3.1 What is a Query Log?

A query log keeps track of information about the interaction between users and the Web search engine. It records the issued queries and also a lot of additional information, such as an anonymized identifier of the user submitting the query, the pages viewed and clicked in the result set, the ranking of each result, the exact time at which a particular action took place, etc.

In its most general form, a query log $\mathcal{QL}$ stores the search activities of a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$ during a given observation period. Let $q_i \in \mathcal{QL}$ be a generic query issued by user u_i and $q_{i,j} \in \mathcal{QL}$ be the j-th query issued by user u_i. Furthermore, let $\tau(q_{i,j})$ be the time at which the query $q_{i,j}$ is issued. Finally, let $R_{i,j}$ and $C_{i,j}$ ($C_{i,j} \subseteq R_{i,j}$) be the set of retrieved and clicked results in response to $q_{i,j}$, respectively.

Moreover, let $\mathcal{S}_i$ be the sequence of *all* the queries $q_i \in \mathcal{QL}$ issued by user $u_i \in \mathcal{U}$, chronologically ordered during the period of observation recorded in the query log: $\mathcal{S}_i = \langle q_{i,1}, q_{i,2}, \ldots, q_{i,|\mathcal{S}_i|}\rangle$. Thus, $\mathcal{R}_i = \langle R_{i,1}, R_{i,2}, \ldots, R_{i,|\mathcal{S}_i|}\rangle$ is the sequence of result sets returned and $\mathcal{C}_i = \langle C_{i,1}, C_{i,2}, \ldots, C_{i,|\mathcal{S}_i|}\rangle$ is the sequence of result sets, which user clicked on. Therefore,

$$\mathcal{QL} = \bigcup_{i=1}^{N} \langle \mathcal{S}_i, \mathcal{R}_i, \mathcal{C}_i \rangle.$$

Each sequence $\mathcal{S}_i$ is roughly defined as a "*query session*". However, several kinds of query sessions may be specified. To this end, later on Section 3.6.1 we refer to a specific kind of query session, i.e., "*(logical) search session*", namely sets/sequences of queries that are all related to the same search goal. Moreover, click-through data, i.e., $\mathcal{C}_i$, represent a sort of implicit relevance feedback information. In particular, every single user action (also, for instance, the action of not clicking on a query result) may be exploited to derive aggregate statistics, which are very useful for the optimization of the search engine effectiveness.

Unfortunately, not all the data stored in a query log are equally relevant to the final analysis. Usually, data *pre-processing* and *preparation* steps are needed for producing a suitable data set from which knowledge could be better extracted.

These steps typically include: data cleaning, query session identification, merging query logs from several applications and removing requests issued by *robots*, etc. The final aim of this initial phase is to remove "noisy" items, so that the resulting inferred knowledge and statistics reflect accurately the actual user behaviors while interacting with the Web search engine.

Although query logs are undoubtedly sources of inspiration for devising novel research topics, real-world Web search engines have been often reluctant to release their historical usage data to the public. This is mostly because today query logs have also a great impact on the decision-making processes and strategic plans adopted by commercial Web search engines. Moreover, several issues arise whenever such sensible users' data are released without planning any effective privacy-preserving mechanism (Section 3.5).

Since 1997, only the following query logs have been made publicly available: Excite (1997), AltaVista (1998, 1999), AOL (2003, 2004, and 2006), and MSN (2006). Table 3.1 resumes the most important features of the query logs that have been analyzed in the latest years.

Query Log Name	Public	Period	# Queries	# Sessions	# Users
Excite (1997)	Y	Sep 1997	1,025,908	211,063	~410,360
Excite Small (1997)	Y	Sep 1997	51,473	–	~18,113
Altavista	N	Aug 2, 1998 Sep 13, 1998	993,208,159	285,474,117	–
Excite (1999)	Y	Dec 1999	1,025,910	325,711	~540,000
Excite (2001)	Y	May 2001	1,025,910	262,025	~446,000
Altavista (public)	Y	Sep 2001	7,175,648	–	–
Tiscali	N	Apr 2002	3,278,211	–	–
TodoBR	Y	Jan 2003 Oct 2003	22,589,568	–	–
TodoCL	N	May 2003 Nov 2003	–	–	–
AOL (big)	N	Dec 26, 2003 Jan 01, 2004	~100,000,000	–	~50,000,000
Yahoo!	N	Nov 2005 Nov 2006	–	–	–
AOL (small)	Y	Mar 1, 2006 May 31, 2006	~20,000,000	–	~657,000
Microsoft RFP 2006	Y	Spring 2006 (one month)	~15,000,000	–	–

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

The most famous publicly-available query log is undoubtedly from AOL, which was released on August 2006. This query log is a very large and long-term collection consisting of about 20 million of Web queries issued by more than $657,000$ users to the AOL search portal over a period of three months, i.e., from 1st March, 2006 to 31st May, 2006. After the controversial discussion related to users' privacy issues followed to its initial public delivery, AOL has withdrawn the query log from its servers and is not offering it for download anymore. Nevertheless, this data set is still hosted on many other mirror servers and available for download.[1]

```
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
```

Figure 3.1: An example of the 2006 AOL query log [98].

Figure 3.1 shows a fragment of the 2006 AOL query log. Each row of this query log consist of records collecting five fields: *(i)* the anonymous identifier of the user issuing the query, *(ii)* the issued query, *(iii)* the timestamp at which the query was issued to the search engine, *(iv)* the position of the clicked result in the retrieved result list, and *(v)* the domain name of the machine hosting the clicked document.

Furthermore, many queries stored in query logs reveal sometimes curious and funny user search behaviors. Just to give an example, from the 2006 AOL query log, user #2386968 asked "*why is my husband so talkative with my female friends*", thus revealing a jealous wife who is worried about her perhaps "lady-killer" husband. Another bizarre example is the request submitted by user #427326 who phrased the query "*where is my computer?*". Maybe, this user was looking for the path of the "My computer" directory on a MS Windows file system.

3.2 A Characterization of Web Search Queries

In this Section, we aim at presenting a characterization of query logs by showing how queries are distributed over time. In particular, we analyze topic-popularity,

[1] `http://sifaka.cs.uiuc.edu/xshen/aol_querylog.html`

term-popularity, differences with respect to the past, variations of topics during day hours, etc.

Some important efforts have been spent in the past to study how people interact with small-scale IR systems (commonly used for digital libraries or legal document retrieval), by analyzing the historical search data of their users [58, 46, 117, 125, 53]. However, the nature of query logs coming from large-scale Web search engines is different from that coming from small-scale IR systems. As an example, *search operators*, like quotes, "+", "-", etc., are rarely used by Web users.

The characteristics of query logs coming from some of the most popular Web search engines have been deeply studied [22, 23, 60, 61, 62, 64, 71, 89, 94, 95, 98, 122, 123, 142]. Typical statistics that can be drawn from query logs are: query popularity, term popularity, average query length, distance between repetitions of queries or terms.

The very first contribution in analyzing query logs comes from Silverstein *et al.* [118]. Here, the authors propose an exhaustive analysis by examining a large query log of the AltaVista search engine containing about a billion queries submitted by 285 million users during a period of 42 days. The study shows some interesting results, including the analysis of the query sessions for each user and the correlation among the terms of the queries. Similarly to other works, authors highlight that the majority of the users (i.e., about 85%) visit the first page of results only. They also show that 77% of the user sessions end up just after the first query.

Jansen *et al.* [62] analyze a log consisting of 51,473 queries submitted by 18,113 Excite users. The query log is properly anonymized. As the information are completely decontextualized, no user profiling can be carried out on this query log.

Lempel and Moran [78], as well as Fagni *et al.* [45], study the content of a publicly available AltaVista log. The log consist of 7,175,648 queries issued to AltaVista during the summer of 2001. No information referring the number of logged users is released. This second AltaVista log is three-years following the one used for the studies proposed by Jansen *et al.* and Silverstein *et al.* Furthermore, the log is smaller than the first one, nevertheless it represents a good picture of Web search engine users.

Moreover, previous works on query log analysis revealed that Web search queries are usually quite short. For instance, the average length of a query in the 1998 Excite log is 2.35 terms. Also, less than 4% of the queries contains more than 6 terms. In the case of the AltaVista log, the average query length is slightly greater, i.e., 2.55. These statistics are significantly different from the ones resulting from the analysis of classical IR systems, where the length of a query ranges from 7 to 15 terms. A possible explanation of this phenomenon is that the Web is a medium used by different people from different part of the world looking for disparate information, while traditional IR systems are instead manually used by professionals and librarians searching for very focused information, who are more able to precisely formulate their needs.

Both the query popularity and term popularity distributions are characterized by a *power law* [107]. This means that the number of occurrences $f(x)$ of a query (or a term) is proportional to $x^{-\alpha}$, where x is the popularity rank and $\alpha > 1$ is a real parameter measuring how popularity decreases against the rank. In a formula, $f(x) \propto Kx^{-\alpha}$, where K is a real positive constant corresponding to the query with the highest popularity.

Markatos [88] is the first who shows that query popularity distribution is characterized by a power law with an exponent $\alpha = 2.4$. He analyzes the Excite query log and plots the occurrence of the first 1,000 most popular queries (Figure 3.2 (a)).

Eventually, later studies confirm the previous power law trend in two other query logs: AltaVista [78] (Figure 3.2 (b)) and Yahoo! [8].

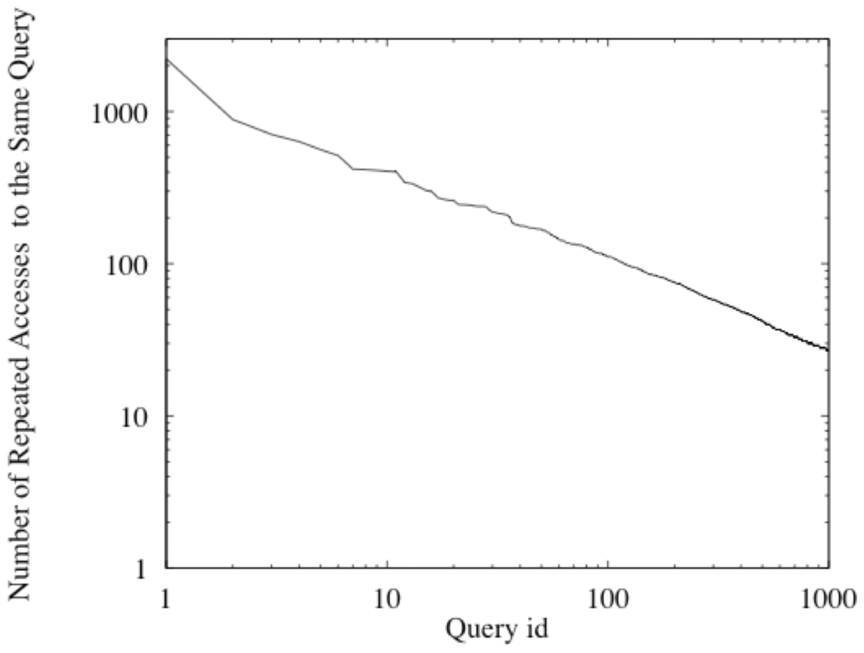
Tables in Figure 3.3 detail the top-20 queries for the Excite [88] and AltaVista [78] logs, respectively. Many queries in both logs refer to sex and sexually explicit topics and many others can be somehow related to the same topics as well. Furthermore, there are some unexpected outcomes in query logs. For instance, the most frequent query in the Excite query log is the empty query. This request accounts for 5% of the total amount of issued queries. Authors in [62] give a possible explanation of this phenomenon. Basically, they state it could be caused either by *(i)* possible errors in typing queries in the search box or *(ii)* Excite reaction to user actions. As an example, Excite result pages have a link pointing to a "More Like This" function that, if clicked, returns pages related to the ones selected. Excite counts that behavior as an empty query, thus incrementing the empty query count.

Moreover, topics covered by queries contained in search engines logs are the most disparate. Figure 3.4 highlights the 250 most frequent terms in the AOL query log by means of a *word cloud.* The dimension of each term in the word cloud is directly related to its frequency in the query log. The bigger a term in the word cloud, the more frequent it is in the log. As an example,"*google*" and "*free*" are the two most frequent terms in the log. Other very frequent words are: "*yahoo*", "*new*", "*county*", "*pictures*", "*http*".

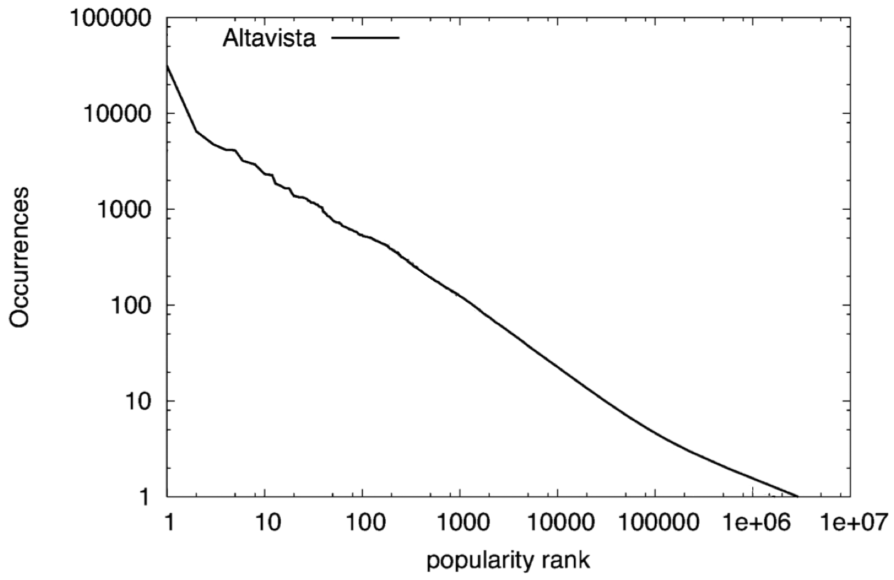
A proof of this multitude of topics may also be given by Figure 3.3. A very first result in categorizing queries is provided by [122]. Here, the authors show the percentage of queries submitted for each topic to the Excite search engine in 1997. Categorizing queries into topics is not a trivial task. Recent papers showing techniques for assigning labels to each query [54, 114, 138, 24, 25, 35] adopt a set of multiple classifiers, subsequently refining the classification phase.

Classification of the Excite queries made by Spink *et al.* in [122] shows that there is no way for "pornography" to be a major topic of Web queries, even though the top ranked query terms may indicate this. Only one in about six Web queries have been classified as about sex ("XXX"). Commerce, including travels, employment, and a number of economic topics are also high in the list. Furthermore, about 10% of the queries are about health and sciences.

Authors of [24, 22] outline similar results on a different query log. The log is made up of billions of Web queries constituting the total query traffic for six months



(a)



(b)

Figure 3.2: The distribution of query popularity (log-log scale) of two query logs: Excite [88] (a) and AltaVista [78] (b).

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

(a)

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

(b)

Figure 3.3: The top-20 queries of two query logs: Excite [88] (a) and AltaVista [78] (b).

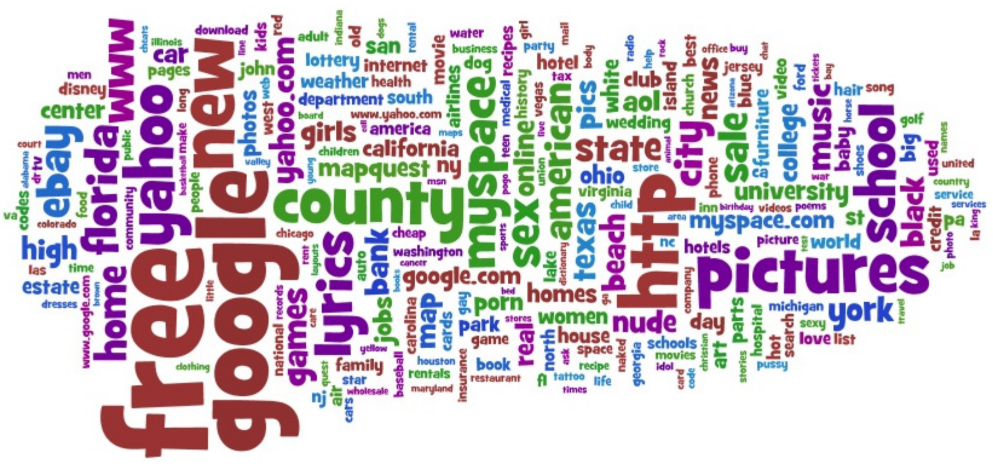


Figure 3.4: A word cloud of the 250 most frequent words in the AOL query log [98], generated using wordle.net [120].

of AOL. Categories are different as well as results (in terms of category percentages breakdown). The difference is caused by the distinct period of time in which the analysis was conducted: while the Excite log refers to queries issued in 1997, the AOL log is younger as it consists of queries issued in 2003. Furthermore, porn queries fell down considerably.

Query terms are distributed according to a power law as well (i.e., in particular, a double Pareto log-normal distribution). In fact, the curve of term distribution falls down sharply, thus denoting that the most frequent terms are much more frequent that the rest of the terms. Figure 3.6 depicts the curve progress of the term popularity distribution in the case of two query logs: Excite [88] and AltaVista [78].

An interesting statistics obtained from query logs is focused on understanding how terms co-occur. In [126], a follow-up of the work presented in [62], Spink *et al.* show the first fifty most frequently co-occurrent terms. Figure 3.2 depicts how terms co-occur in queries without reflecting topic popularity. The majority of term pairs concern "non-XXX" topics, whereas in the same analysis they found that "XXX" queries are highly represented. This highlights that, for some topics, people use more terms to search for precise information, while for other topics the same information need may be satisfied by shorter queries.

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

Table 3.2: List of the fifty most co-occurring terms (term$_1$–term$_2$, frequency) in the Excite query log [126].

Another interesting analysis that can be conducted on search engine logs concerns the repetition of some queries. Since many queries are seen only a few times, one could expect that in the majority of the cases the distance between subsequent submissions of the same query would be very large. Figure 3.7 shows the distance, measured in number of queries, between the submissions of two same queries. Differently from what is expected, the majority of queries have distances that are less than 1000 queries. A possible explanation is the *bursty* nature of query logs, i.e., a large number of people start looking for a topic at the same time. The bursty nature of queries is a feature that is indeed extensively used in many techniques for enhancing both the effectiveness and efficiency of Web search engines.

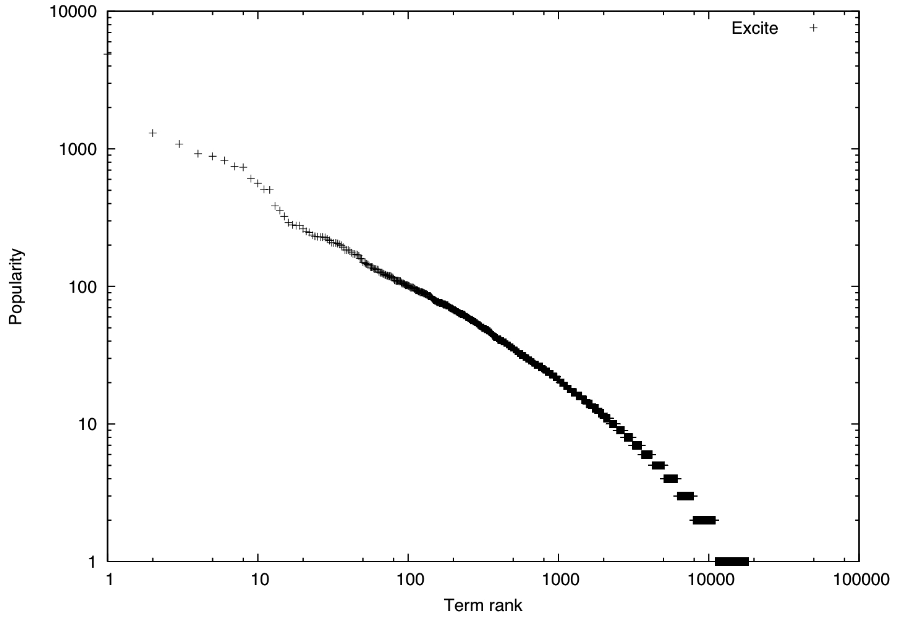
Topic	Percentage
Entertainment or recreation	19.9%
Sex and pornography	16.8%
Commerce, travel, employment, or economy	13.3%
Computers or Internet	12.5%
Health or sciences	9.5%
People, places, or things	6.7%
Society, culture, ethnicity, or religion	5.7%
Education or humanities	5.6%
Performing or fine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

(a)

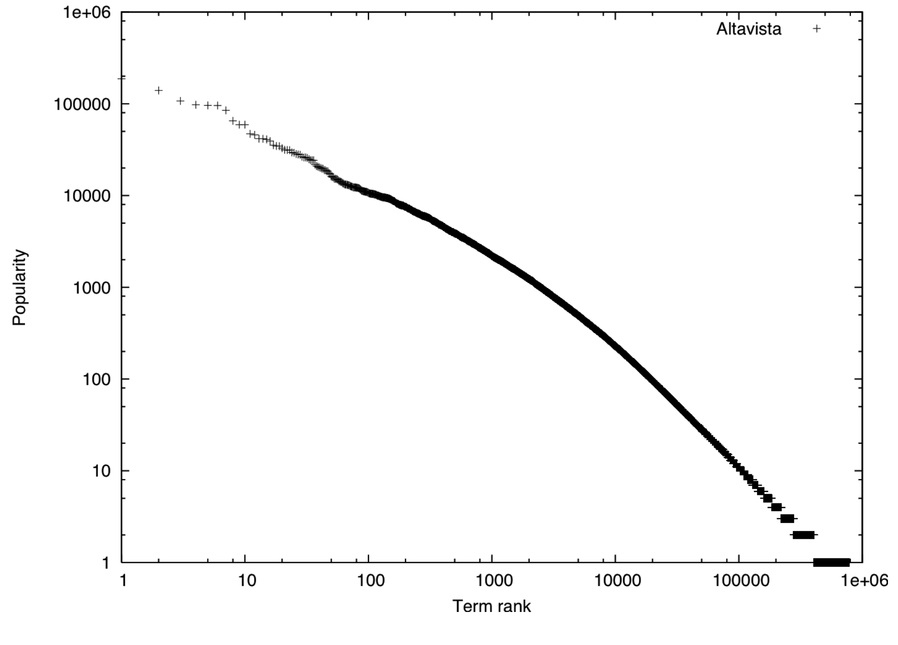
Topic	Percentage
Entertainment	13%
Shopping	13%
Porn	10%
Research & learn	9%
Computing	9%
Health	5%
Home	5%
Travel	5%
Games	5%
Personal & Finance	3%
Sports	3%
US Sites	3%
Holidays	1%
Other	16%

(b)

Figure 3.5: The distribution of query samples across general topic categories of two query logs: Excite [88] (a) and AOL [22] (b).



(a)



(b)

Figure 3.6: The distribution of term popularity (log-log scale) of two query logs: Excite [88] and AltaVista [78].

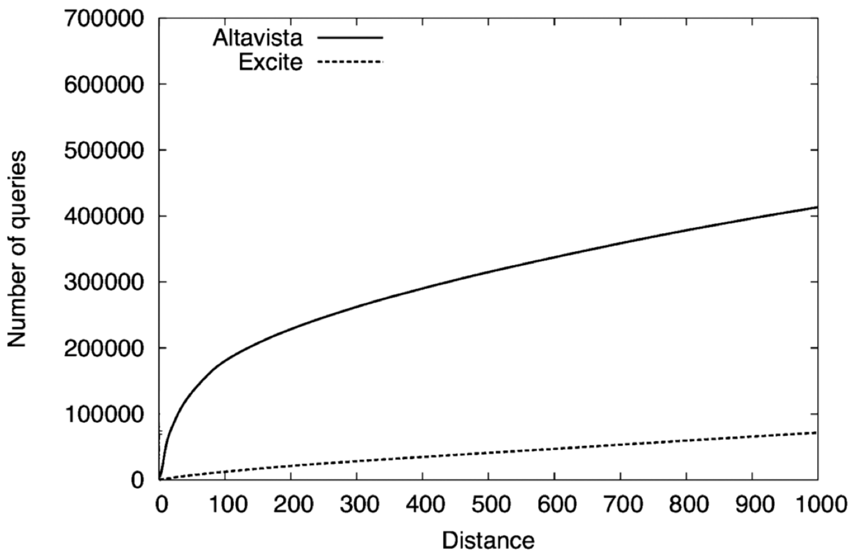


Figure 3.7: Distances (in number of queries) between subsequent submissions of the same query for the AltaVista and Excite logs.

3.3 Time Analysis of Query Logs

Queries may refer to several topics [98], depending also on the historical period [122]. Query logs can be analyzed at different levels of time granularity. On a daily granularity level, some of the topics have shown to be more popular during certain hours of the day [22, 23].

Frequency of queries vary considerably during the day. Ozmutlu *et al.* [95] analyze query frequency against arrival time for the Excite query log in a time period ranging from 9a.m. to 5p.m. Querying activity is higher during the first hours of the day than in the afternoon. There is a sharp decrease in the number of queries submitted, going down from 679 at 9a.m. to 224 at 4p.m. In particular, the number of queries at 4p.m. is about 30% of the queries that are usually submitted at 9a.m. Note that these numbers are small if compared to the activity of today's Web search engines. Comparing these numbers with a similar statistic performed in 2006 [98], results are completely different. Figure 3.8 illustrates how frequencies are distributed within the hours of the day. At 9a.m. queries submitted are almost half of those submitted at 5p.m.

Spink *et al.* [122] show how time periods affect querying behavior of users. Table 3.3 describes how the querying behavior is not changed from a statistical point of view, in a period of four years. The mean number of terms per query is only slightly raised in 2001, while the number of terms per query and the mean queries per user are basically unchanged in four years. Even if this study dates back to 2001, it is very likely that the results it presents are still valid today.

Obviously, the more spread a new technology is the more users become skilled on using it. From Table 3.4 it is clear that users querying for "People, Places or

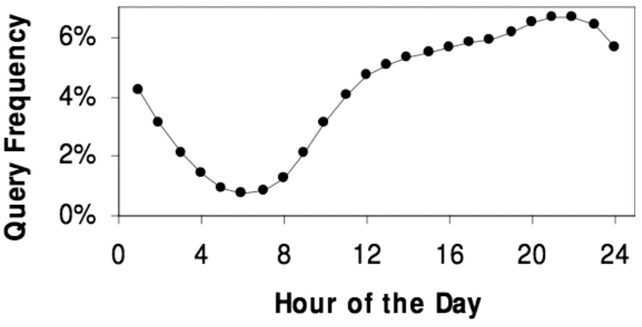


Figure 3.8: Frequencies of queries submitted to the AOL search engine during the day [98].

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

Table 3.3: Comparative statistics for Excite Web queries [122].

Things" were about 50% in 2002. Moreover, on the side of users there is an evident rise of interest for this category: in 1997 queries referring to "People, Places or Things" accounted for less than 7%. The 25% of users in 2002 query for "Commerce, Travel, Employment or Economy" and "Computers, Internet or Technology". This percentage shows an "up-and-down" trend, varying from a minimum of 25% to a maximum of 35%. Furthermore, "Sex and Pornography" shows a falling trend: from 16.8% in 1997 to 3.3% in 2002.

Beitzel *et al.* [22] measure the relative popularity of different categories over the hours in a day. Figure 3.9 shows the percentage of total query volume broken-down to a selected group of category. Different topical categories are more or less popular at different times of the day. As an example, while "Personal Finance" popularity raises during the first hours of the morning between 7a.m. and 10a.m., "Porn" is a category whose popularity raises during late-night until 6a.m.

Figure 3.10 depicts a different analysis on the same categories obtained by applying the *Kullback-Leibler divergence* (KL-divergence) [73]. The reason of applying this measure is to compare categories with a different relative level of popularity in a better way. The comparison is, in fact, affected by popularity shift. To overcome this issue, Beitzel *et al.* compute the KL-divergence between the likelihood of receiving a query on any topic at a particular time and the likelihood of receiving a query in a particular category. The KL-divergence measures a sort of "*most surprising*" category for a particular time of the day. Instead of measuring the popularity as the

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

Table 3.4: Comparison of categories breakdown (in %) for Excite (1997–2001) and Altavista (2002) Web queries [61].

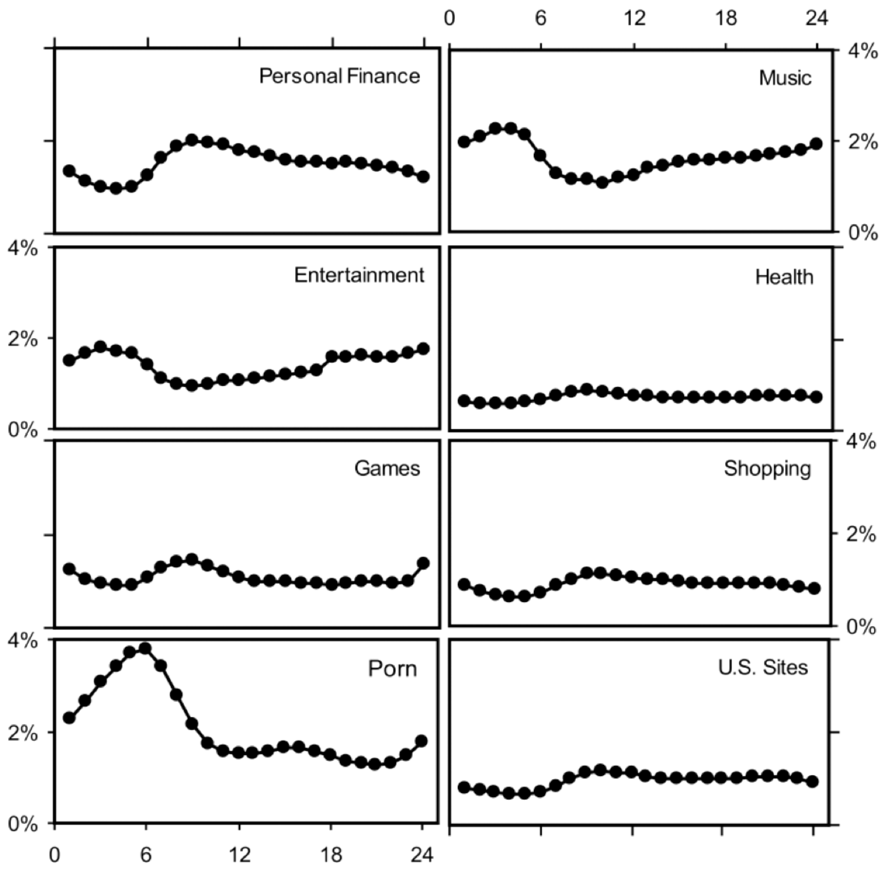


Figure 3.9: Percentage of the query stream covered by selected categories over hours in a day [22].

most numerous topic, the KL-divergence measures how popular is a query in terms of how much it is not being expected.

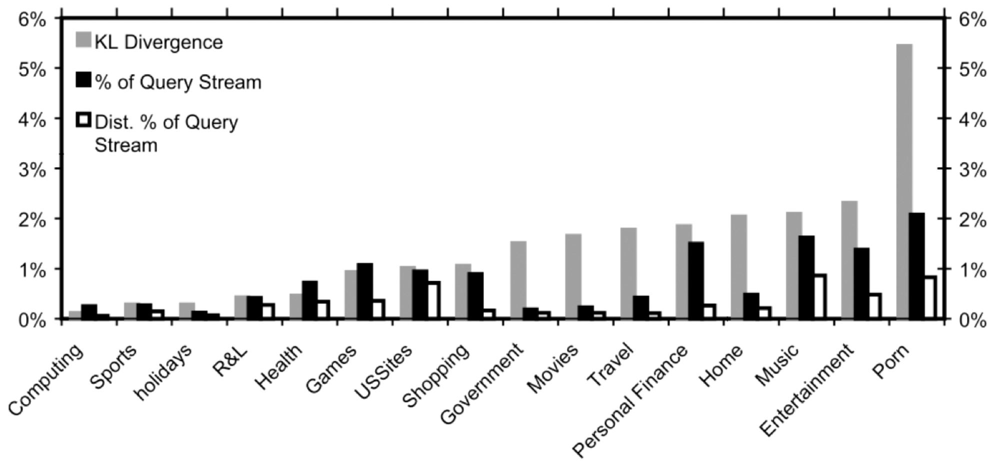


Figure 3.10: Average percentage of the query stream coverage and KL-divergence for each category over hours in a day [22].

A more recent work presents similar results on a MSN Web search engine query log [146]. Authors outline some results that are not detailed with respect to topics, as presented in the previous paper, yet they do not disagree with the overall results presented in [22]. This analysis reveals that the top three categories in terms of popularity are "pornography", "entertainment", and "music". Beitzel *et al.* [22] reach the same conclusion by discussing a more through analysis on weekly and monthly perspective.

3.4 Time-series Analysis of Query Logs

Queries issued by users can be considered as signals in the domain of time. In each time unit it is possible to record the occurrences of the query. The result is a sort of signal to which standard temporal series techniques could be applied [136, 137, 49, 36, 148]. These works introduce techniques for discovering query features, like periodicity or "burstiness".

Adar *et al.* [2] use time series to predict either *(i)* why users issue queries and *(ii)* how users react and cause new spreading. In particular, they propose a novel way to compare signals coming from different sources of information. *Dynamic Time Warping* (DTW) is a way to compare two time series also capturing behavioral properties by mapping inflection points and behavior, such as the rise in one curve to the rise in the second, peak to peak, run to run. Figure 3.11 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.

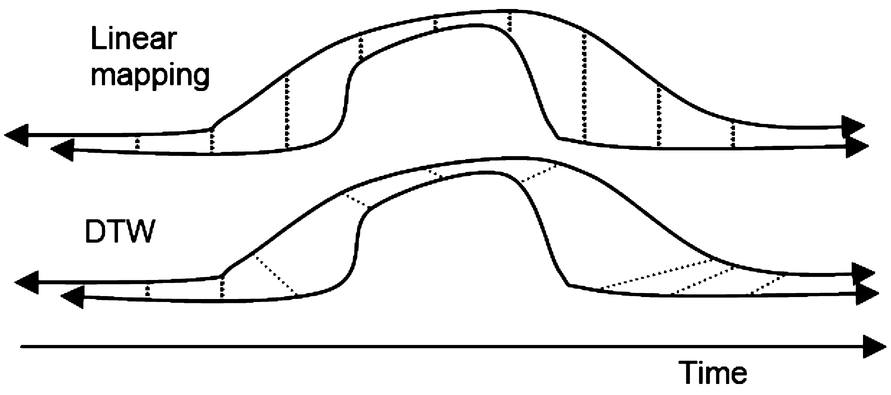


Figure 3.11: Comparing two time series using DTW vs. linear mapping [2].

DTW is obtained by applying a dynamical programming algorithm minimizing the distance in terms of Euclidean distance between two time series points. The algorithm produces a two dimensional array, i.e., DTW, containing how the two time series map to each other. The *best* warping path is simply obtained by crawling the array from the extreme corner backwardly along the minimum gradient direction.

The goal of this work is to discover how time-series are correlated in order to be able to use events in one information source to predict those in another. The data sets used are two query logs (i.e., MSN and AOL), a Web blog, and a news channel. By using human-based tests, authors devise the following 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, if combined, reduce the noise that might have been generated around a topic (e.g., a set of blog posts discussing a news article).

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

Zhang *et al.* [145] study time-series analysis to evaluate predictive scenarios using search engine transactional logs. Authors deal with developing models for the analysis of user behaviors over time and investigate if time series analysis is a valid method for predicting relationships between searchers actions. Time-series analysis is a method often used to understand the underlying characteristics of temporal data in order to make forecasts. The study uses a Web search engine transactional log and time-series analysis to investigate user actions. Authors conduct their analysis in two main phases. The initial phase employs a basic analysis and finds that 10% of searchers clicked on sponsored links. Furthermore, the analysis reveals that, from 10p.m. to 12a.m., searchers almost exclusively click on organic links with almost no clicks on sponsored links. The second and more extensive phase deals with using a one-step prediction time-series analysis method along with a transfer function method. Authors show that the period rarely affects navigational and transactional queries, while rates for transactional queries vary during different periods. Results also highlight that the average length of a searcher session consists of about 2.9 interactions and that this average is consistent across time periods. Moreover, results reveal that searchers who submit the shortest queries (i.e., in number of terms) click on highest ranked results.

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

Vlachos *et al.* [136] present several methods for mining knowledge from the MSN query logs. They analyze time-series built on each query of the log. They demonstrate how to efficiently and accurately discover the important periods in a time-series. Authors also propose a simple but effective method for identification of bursts (either long- or short-term). Later, Vlachos *et al.* [137] motivate the need for more flexible structural similarity measures between time-series sequences, which are based on the extraction of important periodic features. Authors present non-parametric methods for accurate periodicity detection and introduce new periodic distance measures for time-series sequences. Furthermore, they combine these new measures with an effective metric tree index structure for efficiently answering *k-nearest-neighbor* queries. The goal of these tools and techniques is to assist in detecting, monitoring and visualizing structural periodic changes. Authors claim that these methods can be directly applicable in the manufacturing industry for preventive maintenance and in the medical sciences for accurate classification and anomaly detection.

Moreover, Vlachos *et al.* [135] investigate lower and upper distance bounds on time-series logs when working directly in a compressed form. Optimal distance estimation means tighter bounds, leading to better candidate selection/elimination and ultimately faster search performance. Their derivation of the optimal distance bounds is based on the careful analysis of the problem using optimization principles. The experimental evaluation shows a clear performance advantage of the proposed method, compared to previous compression/search techniques. This method results in a 10–30% improvement on distance estimations, which in turn leads to 25–80% improvement on the search performance.

Yizhou *et al.* [143] study the problem of mining causal relation of queries in Web search engine logs. They firstly detect events in query logs by applying a statistical frequency threshold. Then, the causal relation of queries is mined by the geometric features of the events. The experimental results demonstrate that this approach can accurately detect the events in temporal query logs and effectively identify the causal relation of queries.

Finally, Chien *et al.* [38] try to find semantically related search engine queries based on their temporal correlation. They figure out that two queries are related if their popularity behave similarly over time. They also define a new measure of the temporal correlation of two queries based on the correlation coefficient of their frequency functions. Eventually, authors come up with a method for efficiently finding the highest correlated queries for a given input query.

3.5 Privacy Issues in Query Logs

Privacy in query logs becomes a hot topic in 2006 when AOL compiled a statistical sampling of more than 20 million queries entered by approximately 657,000 of their users. Such huge quantity of data was released for research purposes. Usernames were anonymized replacing them by unique numbers. Anyway, these numbers provide a sort of *thread* by which queries of a given user could be identified. The identification process is easy if user entered some pieces of information, which allows her identity to be discerned.

Many commercial search engines overcome the problem by simply not publishing their logs. However, lot of efforts have been spent in order to find good techniques for "sanitizing" query logs. As an example, Bar-Ilan in [16] state that "interesting results can be obtained from query logs without jeopardizing the privacy of users".

A seminal solution to the *anonymity preservation* challenge has been proposed by Sweeney in [130]. Sweeney introduces the concept of *k-anonymity*, which ensures that each information request contains at least k (anonymized) individuals with the same values, so that it is not possible to identify one individual in particular.

Jones *et al.* [68] provide a detailed description of a data analysis process that leads to information disclosure in a query log. They show how the combination of simple classifiers can be used to map a series of user queries into a gender, age, and location, showing that this approach remains very accurate even after personally identifying information have been removed from the log. Authors emphasize that a user can be identified by a real-life acquaintance. This type of person has background knowledge on the user (e.g., location, age, gender, or even access the user's browser) and can use it to disclose the activities of the user in the log.

Adar [1] elaborates on vulnerabilities in the AOL log and shows that traditional privacy preservation methods cannot be transferred directly to query logs. Adar also points out that k-anonymity is too costly for query log "*anonymization*", because this type of data set changes very rapidly. Two user anonymization methods are proposed, whose goal is to balance the achieved privacy and the retained *utility*, i.e. the usability of the anonymized log for statistical analysis. The term "utility" refers to the *data utility* of the anonymized log for the purposes of non-adversarial information acquisition. Verykios *et at.* [134] count utility as one of the important features for the evaluation of privacy preserving algorithms (next to the performance of the algorithm) the level of uncertainty with which the sensitive information can be predicted and the resistance to different data mining techniques.

Xiong and Agichtein [141] describe some important applications of query log analysis and discuss requirements on the degree of granularity of query logs. Then, authors analyze the sensitive information in query logs and classify them from the privacy perspective. Two orthogonal dimensions are described for anonymizing query logs and a spectrum of approaches along those dimensions is presented. Furthermore, the authors discuss whether existing privacy guidelines such as HIPAA[2] can apply to query logs directly.

Cooper [39] assesses seven state-of-the-art privacy-preserving techniques against three main criteria: *(i)* how well the technique protects privacy, *(ii)* how well the technique preserves the utility of the query logs, and *(iii)* how well the technique might be implemented as a user control. The author highlights that achieving the right balance between protecting privacy and promoting the utility of the query logs is thus difficult but necessary to ensure that Web users still rely on search engines without fear of adverse privacy consequences.

Poblete *et al.* [101] look into the privacy related implications of query log analysis and, in particular, they analyze a new concern, namely the *privacy of businesses.* This includes both institutions, such as companies, and people in the public eye, such as political leaders. Authors provide a definition of *confidential* information and analyze attacks that lead to confidentiality breaches, including methods to prevent information disclosure.

Kumar *et al.* [74] study the privacy preservation properties of a specific technique for query log anonymization: *token-based hashing.* In this approach, each query is tokenized and then a secure hash function is applied to each token. Authors show that statistical techniques may be applied to partially compromise the anonymization. Thereby, authors analyze the specific risks that arise from these partial compromises, focused on revelation of identity from unambiguous names, addresses, and the revelation of facts associated with an identity that are deemed to be highly sensitive. The goal of the study is twofold: from a side it observes that token-based hashing is unsuitable for anonymization and on the other hand it presents a concrete analysis of specific techniques that may be effective in breaching privacy, against which other anonymization schemes should be measured.

[2] `http://www.hhs.gov/ocr/privacy/hipaa/administrative/privacyrule/index.html`

3.6 Applications of Query Log Mining

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

In the rest of this Chapter, we report the major state-of-the-art contributions that are related to the two main subjects of this dissertation. In particular, we discuss about *search session discovery* and *query suggestion*, which are thereby addressed in Chapter 4 and Chapter 5, respectively. Finally, we refer to [120] those readers who are interested in topics that are not fully covered by this Chapter.

3.6.1 Search Session Discovery

The vast majority of the works concerning with query log mining aim to understand the real intent behind queries issued by users. To this end, Broder [33] claims that the "need behind the query" in Web context is not clearly *informational* like in classical information retrieval domain. Hence, he proposes a *taxonomy* of Web searches by classifying the queries according to their intent into three classes: *(i) navigational*, whose goal is to reach a specific Web site (e.g., "`nba.com`"), *(ii) informational*, which aims to acquire some information assumed to be present in one or more Web documents (e.g., "`2011 best nba player`"), and finally *(iii) transactional*, whose intent is to perform some Web-mediated activity (e.g., "`nba tickets`").

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

Moreover, Rose and Levinson [111] propose their own user search goals classification by extending the taxonomy devised by Broder in [33] and adding more hierarchical levels.

However, queries taken on their own are often not enough to determine actual user search goal. In order to overcome this issue, a very important piece of information we can extract from a query log is represented by "*search sessions*", i.e. specific

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

Table 3.5: Query classification on the basis of user survey [33].

sets/sequences of queries submitted by a user while interacting with a Web search engine for satisfying a specific information need. Indeed, users typically submit a series of queries as part of a single information seeking activity.

Unfortunately, effectively discovering user search sessions from the queries issued in a row and stored in query logs is not straightforward. The main problem is that users often interact with the Web search engine in a *multi-tasking* way, namely they interleave queries related to different search needs within a small amount of time.

Previous works on search session discovery may be classified into: *(i) time-based*, *(ii) content-based*, and *(iii) ad-hoc*, which usually combines both *(i)* and *(ii)*.

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

Silverstein *et al.* [118] present a broad analysis of a very large query log data set collected by the AltaVista search engine and firstly define a concept of *session* as follows: two consecutive queries are part of the same session if they are issued at most within a 5-minutes time window. According to this definition, they find that the average number of queries per session in the data they analyzed is 2.02. He and Göker [55] use different timeouts to split user sessions of Excite query log, ranging from 1 to 50 minutes.

Radlinski and Joachims [105] observe that users often perform a sequence of queries with a similar information need and they refer to those sequences of reformulated queries as *query chains*. Their work presents a method for automatically detecting query chains in query and click-through logs using 30 minutes threshold for determining if two consecutive queries belong to the same search session.

Jansen and Spink [61] make a comparison of nine Web search engines transaction logs from the perspectives of *session length*, *query length*, *query complexity*, and *content viewed*. Here, they provide another definition of session, i.e. *search episode*, describing it as the period of time occurring from the first to the last recorded timestamp on the Web server from a particular user in a single day, so that session length might vary from less than a minute to a few hours. Moreover, using the same concept of search episode, Spink *et al.* [124] investigate also *multi-tasking* behaviors, while users interacting with a Web search engine. Multi-tasking during Web searches involves the *seek-and-switch* process among several topics within a single user session. Again, a user session is defined to be the entire series of queries submitted by a user during one interaction with the system, so that session length might vary from less than a minute to a few hours. The results of this analysis performed on an AltaVista query log show that multi-tasking is a growing element in Web searching.

Finally, Richardson [109] shows the value of long-term query logs with respect to short-term, i.e., within-session, query information. He claims that long-term query logs can be used to better understand the world where we live, showing that query effects are long-lasting. Basically, in his work Richardson does not look at term co-occurrences just within a search session (which he agree to be a 30-minutes time window) but rather across entire query histories.

Content-based. Content-based approaches suggest to exploit the lexical content of the queries themselves for determining a possible topic shift in the stream of issued queries and thus a *session boundary* [75, 56, 93].

To this extent, several *search patterns* have been proposed by means of lexical comparison, using different string similarity scores (e.g., *Levenstein*, *Jaccard*, etc.). However, approaches relying only on content features suffer of the so-called *vocabulary-mismatch problem*, namely the existence of topically-related queries without any shared terms (e.g., the queries "*nba*" and "*kobe bryant*" are completely different from a lexical content perspective but they are undoubtedly related).

In order to overcome this issue, Shen *et al.* [115] compare *expanded representation* of queries, instead of the actual queries themselves. Each individual expanded query was obtained by concatenating the titles and the Web snippets for the top-50 results provided by a search engine for the specific query. Thus, the relatedness between query pairs is computed using the *cosine similarity* between the corresponding expanded queries.

Ad-hoc. Usually, these approaches propose to combine the two previous techniques. Jansen *et al.* [63] assume that a new search pattern always identifies the start of a new session. Moreover, He *et al.* [56] show that statistical information collected from query logs could be used for finding out the probability that a search pattern actually implies a session boundary. In particular, they extend their previous work [55] to consider both temporal and lexical information.

Similarly, Ozmutlu *et al.* [93] describe a mechanism for identifying topic changes in user search behavior by combining time and query content features. They test the validity of their approach using a genetic algorithm for learning the parameters of the topic identification task. The algorithm takes into account topic shift and continuation probabilities of the data set leveraging on query patterns (i.e., lexical content) and time intervals. Seco and Cardoso [113] propose that a candidate query belongs to a new session if it does not have any common terms with the queries of the current session or the time interval between the candidate query and the last query in the current session is greater than 60 minutes. Gayo-Avello [51] presents an exhaustive survey on session boundary detection methods and proposes a new technique, which works on the basis of a geometric interpretation of both the time gap and content similarity between consecutive query pairs.

Boldi *et al.* [28] introduce the *Query Flow Graph* (QFG) as a graph model for representing data collected in Web search engine query logs. Intuitively, in the QFG a directed edge from query q_i to query q_j means that the two queries are likely to be part of the same search goal. Any path over the QFG may be seen as a searching behavior, whose likelihood is given by the strength of the edges along the path. Figure 3.12 shows an example of the user search behavior represented by means of a QFG. Authors propose a methodology that builds a QFG by mining time and textual information as well as aggregating queries from different users. Using this approach, authors build a real-world QFG from a large-scale query log and demonstrate its utility in two concrete applications: *(i)* discovering search sessions and *(ii)* query suggestion. Concerning *(i)*, they exploit this model for segmenting the query stream into sets of related information-seeking queries, leveraging on an instance of the *Asymmetric Traveling Salesman Problem* (ATSP). The application of QFG to the query suggestion problem is explained later on Section 3.6.2.

Finally, Jones and Klinkner [67] argue that within a user's query stream it is possible to recognize particular hierarchical units, i.e., *search missions*, which are in turn subdivided into disjoint *search goals*. A search goal is defined as an atomic information need, resulting in one or more queries, while a search mission is a set of topically-related information needs, resulting in one or more goals. Given a manually generated ground-truth, Jones and Klinkner [67] investigate how to *learn* a suitable binary classifier, which is aimed to precisely detect whether two queries belong to the same task or not. Among various results, they realize that timeouts, whatever their lengths, are of limited utility in predicting whether two queries belong to the same goal and thus unsuitable to identify session boundaries.

3.6.2 Query Suggestion

Often users of Web search engines refine their previously submitted queries by adding or modifying keywords until they satisfy their search goal. In order to support users during the search process, modern Web search engines have started to add mechanisms and features aimed to help people in better formulating their information needs, thus to quickly satisfy their goals. In this regard, *query suggestion* plays a significative role by recommending lists of alternative yet related queries that are likely to better specify user search needs.

Differently from *query expansion* [41], query suggestion aims at producing hints for users, thus giving them the possibility to select the best similar query to refine their search, instead of having the query automatically expanded with a lot of different terms.

The activity of producing suggestions from queries submitted in the past may be considered as a way of "exploiting queries submitted by experts to help non-expert users" [10], namely as a way of profiting by the so-called "*wisdom of crowds*" [129]. 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. This means that suitable lists of recommended queries are obtained by analyzing past user search activities recorded on query logs.

A *naïve* approach to devise a list of candidate queries to recommend for a given query simply looks at those queries sharing many common terms. Using this approach, the two queries "*new york knicks*" and "*new york yankees*" would be considered to some extent *similar*, as they both share the terms "new" and "york". The example above shows that the naïve approach might result in misleading suggestions for users.

Lot of efforts have been spent on query recommendation thus producing several contributions in the field. Some works propose to select queries to be suggested from those that frequently appear in query sessions [48]. Other works use clustering to devise similar queries on the basis of clustering membership [10, 11, 12], or they exploit click-through data information to devise query similarity [147, 40].

Query sessions can be an important source of information for devising potentially related queries to be suggested to a user. An important intuition is that if a lot of users in the past asked for a certain query q_j *after* they issued the query q_i, then q_j is likely to be a *valid*, i.e., interesting, suggestion for q_i.

Fonseca *et al.* [48] exploit the idea above by mining association rules from query logs. Authors run an association rules mining algorithm on a query log. However, extracting association rules from query logs could be computationally expensive due to the typical dimensions of the data sets. Thereby, the approach presented by Fonseca *et al.* [48] mines only rules of the form $q_i \Rightarrow q_j$, thus reducing the total effort needed for the overall computation. Basically, for each query q_i, all the associations whose support is above a given threshold σ are sorted by confidence level and saved in the form of rules $q_i \Rightarrow q_1$, $q_i \Rightarrow q_2$, $\ldots$, $q_i \Rightarrow q_m$. Authors test the proposed technique on a real-world query log coming from a Brazilian search engine consisting of 2.312.586 queries. Experiments use a support threshold $\sigma = 3$. The produced recommendations are evaluated by means of a survey involving five assessors. Results are encouraging as the technique scores 90.5% of precision (with five queries suggested) measured as the number of suggestions retained relevant for the five assessors. By augmenting the number of suggestions provided to users, precisions drops to 89.5% when ten queries are suggested and down to 81.4% when 20 queries are suggested.

Zaïane and Strilets [144] use a *Query Memory* model to store past queries and retrieve them according to the one submitted. The method computes associations on-the-fly at query resolution time. A *Query Memory* is a set of queries represented by six different features, namely *(i)* a bag of words representation of the query (*BagTerms*), *(ii)* the frequency with which the query has been submitted (*Count*), *(iii)* the timestamp recording the last time the query was submitted (*LDate*), *(iv)* the timestamp recording the first time the query was submitted (*FDate*), *(v)* the query result list stored as records made up of: URL, Title and Snippet of each result page (*QResult*), and *(vi)* the timestamp recording the date at which results were obtained. In order to produce query recommendations, seven different methods may be used:

- ***Naïve query-based method:*** returns queries having in common at least one term;
- ***Naïve simplified URL-based method:*** returns queries having in common at least one URL in the result lists;
- ***Naïve URL-based method:*** returns queries having in common a large portion of the URLs in the result list;
- ***Query-Title-based method:*** returns queries where terms in their result titles are contained in the submitted query;
- ***Query-Content-based method:*** the same as the above only considering snippet terms instead of title terms;
- ***Common query title method:*** returns queries whose results share title terms;
- ***Common query text method:*** the same of the previous one only considering snippet terms instead of the title terms.

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

Baeza-Yates *et al.* [10] use a clustering approach to query recommendation. The query recommendation technique works by following a two-level approach. An *offline* process clusters past queries using text from queries and clicked URLs. Moreover, an *online* process performs the following two steps, that is *(i)* given an input (i.e., a query), the most representative cluster is found and *(ii)* each query in the cluster is ranked according to two criteria: the *similarity* and the *attractiveness* of query answer, i.e., how much the answers of the query have attracted the attention of users. In their work, authors refer to this as *support*. Moreover, it should not be confused with the more famous support of association rules [3]. The offline query clustering algorithm operates over queries that are previously enriched by a selection of terms extracted from the clicked documents. Clusters are computed by applying a K-MEANS algorithm [85]. The similarity between queries is computed according to a vector space approach. Therefore, each query q is represented by a vector $\mathbf{q}$ whose i-th component $\mathbf{q}[i]$ is computed as:

$$\mathbf{q}[i] = \sum_{u \in URLs} \frac{Clicks(q, u) \times tf(t_i, u)}{\text{argmax}_t \, tf(t, u)},$$

where $Clicks(q, u)$ is the percentage of clicks the URL u receives when answered in response to the query q and $tf(t, u)$ is the number of occurrences of the term t in the document pointed to URL u. All the clicked URLs are considered in computing the sum. The distance between two queries is computed by the cosine similarity of their relative vectors. The approach is evaluated on the TodoCL search engine[3] query log containing 6,042 unique queries with the associated click-through information. Also, 22,190 clicks are present in the log referring to 18,527 different URLs. The evaluation is performed on ten queries by means of a user study. Results show that presenting query suggestions ranked by support (i.e., the frequency of the query in the query log) yields to more precise and high quality suggestions.

Jones *et al.* [69] propose a model for suggesting queries based on the concept of query rewriting. Basically, a query is rewritten into a new one by means of query or phrase substitution. The rewriting process is based on the *Log-Likelihood Ratio* (LLR) measure [87] to assess interdependencies between terms of queries. Given two terms, the higher is the LLR, the higher is the likelihood of the two words of being dependent to each other. Thus, query pairs with high LLR are identified as *substitutable*. Authors also propose a systemization of possible suggestions into four classes, ranging from the most precise to the less accurate class: *precise rewriting*, *approximate rewriting*, *possible rewriting*, and *clear mismatch*. Precise rewriting means that the suggested query has exactly the same semantic of the one to be replaced. Approximate rewriting is the class containing narrowed or broadened queries of the initial one. Possible rewriting is a still less precise query suggestion methodology: queries are in some categorical relationship. The last class, i.e., clear mismatch, is the less precise and contains query pairs where no relationships can be found.

Authors use such four classes for defining the two general tasks of query suggestion: *specific rewriting* (i.e., precise and approximate rewriting) and *broad rewriting* (i.e., precise, approximate, and possible rewriting). Given these two tasks, the problem of generating query recommendations can be considered as a classification problem. Authors thus adopt and test different classifiers. Training is performed on a set of manually annotated *candidate* substitutions extracted from queries of a query log. The assessment of the method is performed by using an automatic

[3]`http://www.todocl.cl/`

evaluation. For the query log used in [69], the precision of the method has been measured to be 76% with a recall figure of 92%.

Boldi *et al.* [28] introduce the Query Flow Graph (QFG), a graph representation of the interesting knowledge about latent querying behavior. Using this approach, authors build a real-world QFG from a large-scale query log and demonstrate its utility for generating valuable query recommendations.

In a later study, Boldi *et al.* [29] propose, and experimentally study, query recommendations based on short random walks on the query flow graph. The conducted experiments show that these methods can match in precision, and often improve, recommendations based on query-click graphs, without using click-through data. Experiments also show that it is important to consider transition-type labels on edges for having good quality recommendations.

Moreover, Baraglia *et al.* [18, 19] present a study of the effects of time on recommendations generated by using the QFG introduced by [28]. Indeed, user search interests change over time and the knowledge extracted from query logs may suffer of aging effects, as new interesting search topics appear. In order to overcome this issue, authors propose a novel incremental algorithm for keeping up-to-date the QFG recommendation model, without having to rebuild it from scratch every time freshest query log data happen. Furthermore, Broccolo *et al.* [31] propose two novel query recommendation algorithms that *incrementally* update the model on top of which suggestions are generated, taking care of each new processed query.

Baeza-Yates and Tiberi [14] use click-through data as a way to provide recommendations. The method is based on the concept of *cover graph*. A cover graph is a bipartite graph of queries and URLs, where a query and a URL are connected if the URL was returned as a result for the query and a user clicked on it. To catch the relations between queries, a graph is built out of a vectorial representation for queries. In such a vector space, queries are points in a high-dimensional space where each dimension corresponds to a unique URL u that was, at some point, clicked by some user. Each component of the vector is weighted according to the number of times the corresponding URL has been clicked when returned for that query.

For instance, let us suppose to have five different URLs, namely, $u_1, u_2, \ldots, u_5$, and suppose also that for query q users have clicked three times on URL u_2 and four times on URL u_4, thereby the resulting 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 vectors representing the linked queries. More formally, the weight $\omega(q, q')$ of an edge $e = (q, q')$ is computed according to Equation (3.1). In the formula, D is the number of dimensions of the space, i.e., the number of distinct clicked URLs, whereas $\mathbf{q}$ and $\mathbf{q}'$ are the vectors associated with q and q', respectively.

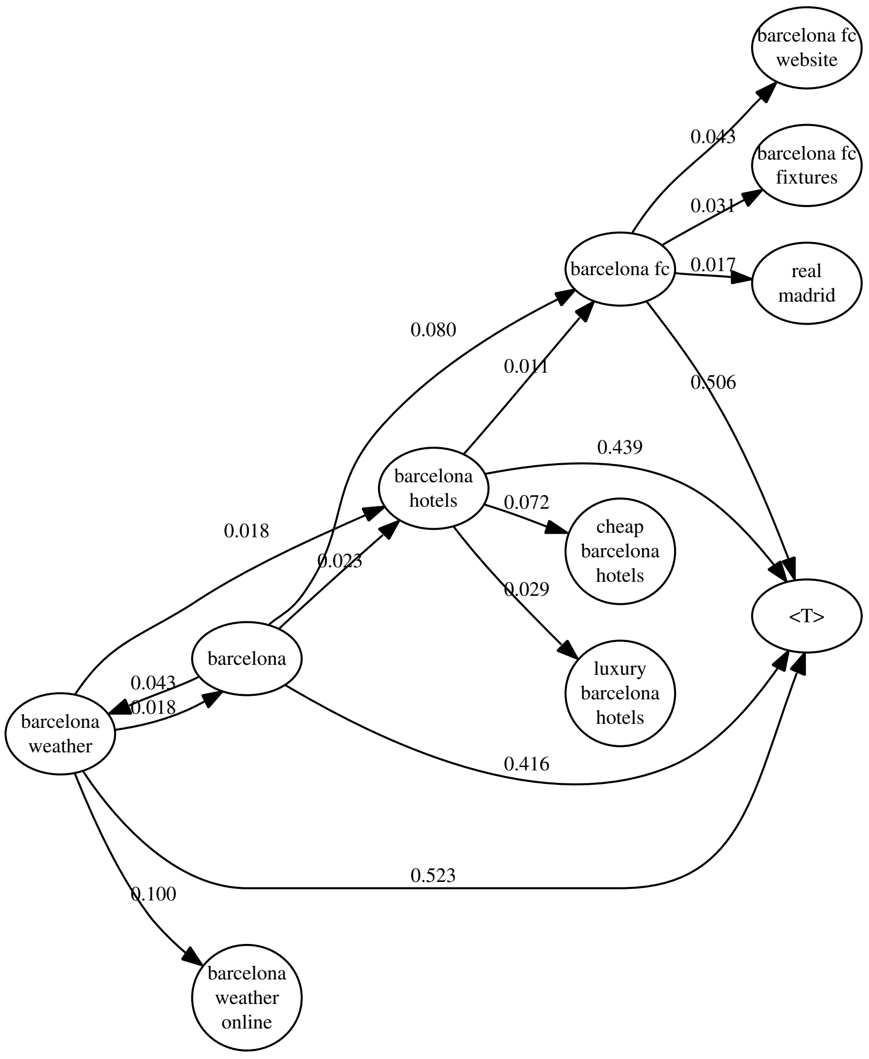


Figure 3.12: An example of user search behavior represented by means of Query Flow Graph [28].

$$\omega\left(q, q'\right) = \frac{\mathbf{q} \cdot \mathbf{q}'}{\|\mathbf{q}\| \cdot \|\mathbf{q}'\|} = \frac{\sum\limits_{i \leq D} \mathbf{q}[i] \cdot \mathbf{q}'[i]}{\sqrt{\sum\limits_{i \leq D} \mathbf{q}[i]^2} \sqrt{\sum\limits_{i \leq D} \mathbf{q}'[i]^2}}. \tag{3.1}$$

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

Baraglia *et al.* [17] propose a model for query recommendation, which address a newly introduced problem, i.e., the *Search Shortcut Problem* (SSP), which basically consists in recommending *successful* queries, namely those queries that in the past allowed other users to satisfy similar information needs. This new model presents several advantages with respect to traditional query suggestion approaches. Firstly, it allows a straightforward evaluation of algorithms from available query log data. Moreover, it simplifies the application of several recommendation techniques from other domains. In particular, authors apply collaborative filtering techniques to this problem. The proposed query recommendation technique is evaluated on two large query logs (i.e., AOL and MSN). Different techniques for analyzing and extracting information from query logs, as well as new metrics and techniques for measuring the effectiveness of recommendations, are proposed and evaluated. The presented approach highlights notably accurate results, which demonstrate that collaborative filtering techniques can be useful in recommending queries.

So far, most of the described studies only focus on *popular* queries. Since rare queries have much less information (e.g., clicks) than popular queries in the query logs, it is much more difficult to efficiently suggest relevant queries to a rare query.

Yang *et al.* [121] propose an optimal rare query suggestion framework by leveraging on implicit feedbacks from users in the query logs. The model resembles the principle of *pseudo-relevance feedback*, which assumes that top-returned results by search engines are relevant. However, authors argue that the clicked URLs and skipped URLs contain different levels of information and thus should be treated separately. Hence, the query suggestion framework optimally combines both the click and skip information from users and uses a *random walk* [99, 100] model to optimize the query correlation. The proposed model specifically optimizes two parameters: *(i)* the restarting (jumping) rate of random walk and *(ii)* the combination ratio of click and skip information. Unlike the *Rocchio* algorithm [86], the proposed learning process does not involve the content of the URLs but simply leverages on the click and skip counts in the query-URL bipartite graph. Consequently, the model is able to scale up to the need of commercial search engines.

Broder *et al.* propose to rely on the results from search engines as an external knowledge base for building the word features for rare queries [35]. The authors train a classifier on a commercial taxonomy consisting of 6,000 nodes for categorization. Results show a significant boost in term of precision with respect to the baseline query expansion methods. Lately, Broder *et al.* propose an efficient and effective approach for matching ads against rare queries [34]. The approach builds an expanded query representation by leveraging offline processing done for related popular queries. Experimental results show that the proposed technique significantly improves the effectiveness of advertising on rare queries with only a negligible increase in computational cost.

Mei *et al.* propose a novel query suggestion algorithm based on ranking queries with the hitting time on a large-scale bipartite graph [91]. The rationale of this method is to capture semantic consistency between the suggested queries and the original query. Empirical results on a query log from a real-world search engine outline that hitting time is effective to generate semantically consistent query suggestions. Authors show that the proposed method and its variations are able to boost long-tail queries and personalized query suggestion.

Finally, Broccolo *et al.* [32] propose an efficient and effective query recommendation algorithm that can "*cover*" also rare, i.e., *long-tail*, queries by exploiting the Search Shortcut model introduced in [17].

4 Search Task Discovery

"Un même sens change selon les paroles qui l'expriment. Les sens reçoivent des paroles leur dignité, au lieu de la leur donne."
(The same meaning changes with the words which express it. Meanings receive their dignity from words instead of giving it to them.)

BLAISE PASCAL
Pensées (1669)

In this Chapter, we present an innovative approach to the first research challenge already introduced in Section 3.6.1, namely the application of query log mining techniques for effectively discovering actual user search sessions "hidden" inside the raw stream of queries stored in Web search engine query logs. The contents of this Chapter are based on three research papers, i.e., "*Detecting Task-based Query Sessions Using Collaborative Knowledge*" [81], "*Identifying Task-based Sessions in Search Engine Query Logs*" [83], and "*Discovering User Tasks in Long-Term Web Search Engine Logs*" [82]. The first paper was published in the Proceedings of the International Workshop on Intelligent Web Interaction (IWI 2010), which was held in conjunction with the IEEE/WIC/ACM International Conferences on Web Intelligence. The second one was published in the Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM 2011) and it was also one of the six candidates to the best paper award. Finally, the third paper represents the completion of the first one and it was submitted to the ACM Transactions on Information Systems (TOIS) journal.

People are increasingly asking *Web search engines* for satisfying the information needs related to accomplish their daily *tasks*, such as "*planning holidays*", "*obtaining a visa*", "*organizing a birthday party*", etc. Although the huge number of features which now the most popular Web search engines come with, in essence they still belong to the category of Web documents retrieval tools. Indeed, results provided by modern search engines in response to a user query are still *unstructured* lists of "*ten blue links*" to Web pages. However, when the user's need behind a certain query is a task to be accomplished, the traditional "*query-look-refine*" paradigm should be improved for "driving" the user towards the execution of her desired task. Therefore, we believe next-generation Web search engines should provide new features and capabilities to support users in their everyday activities. Of course, this opens up novel and exciting research challenges, in particular the ability to recognize the tasks behind user queries. Thus, we focus on identifying *task-oriented sessions* from past issued queries, i.e., sets of possibly non-contiguous queries phrased by users for carrying out various tasks. First, we built, by means of a manual labeling process, a *ground-truth* where the queries of a real query log have been grouped in tasks. Our analysis of this ground-truth shows that users tend to interleave several search tasks within relatively small units of time, since about 75% of the submitted queries involve a multi-tasking activity. Moreover, we define the *Task-oriented Session Discovery Problem* (TSDP) as the problem of best approximating the above ground-truth. The TSDP deals with two aspects: *(i)* a robust measure of the *task relatedness* between any two queries, i.e., *task-based query similarity*, and *(ii)* an effective method for actually discovering task-oriented sessions by using the above measure of task relatedness. Concerning *(i)*, we propose and compare both *unsupervised* and *supervised* approaches for devising several task-based query similarity functions. These functions also exploit the collaborative knowledge collected by *Wiktionary* and *Wikipedia* for detecting query pairs that are not similar from a lexical content point of view, but actually *semantically* related. Therefore, we tackle *(ii)* by introducing a set of *query clustering* methods that exploit the above similarity functions for detecting user tasks. All the proposed solutions have been evaluated on the ground-truth and two of them have been shown to perform better than state-of-the-art approaches.

4.1 Introduction

The *World Wide Web* (i.e., Web) was born as a *platform* to connect academic and research people, which exploit the Internet as the communication medium infrastructure. Rapidly, an increasing number of users, which were not directly involved in academia or research activities, have started to have access to the Web as well.

Nevertheless, in the first Web era there was still a clear separation of roles between *few* content providers, i.e., typically "authoritative" skilled workers and professionals, and *many* content consumers, i.e., common end users. During the last years, a new trend has gained momentum: new "*social-oriented*" applications that allow easy authoring and content creation have lead to an increased *democratization* and *collaborative involvement* in the Web. Somehow, this process caused the end of the first Web era, by bringing down the wall between content providers and consumers, which now can play both roles interchangeably from time to time. Therefore, information made available on the Web have started raising at a tremendous speed rate, reaching nowadays a huge and still growing number of contents, which spread over several media types (e.g., text, images, audio/video, etc.).

This great repository of data makes the Web the place of choice where people look at whenever they come up with *any* sort of information need. Indeed, there is a common belief that the Web is increasingly used not only for consulting documents but also for trying to simplify the accomplishment of various everyday activities, i.e., *tasks*. Moreover, most of the interactions between users and the Web are often mediated by *Web search engines*, which are amongst the most important and used Web-based tools. This trend is confirmed by a rising "addiction to Web search": no matter what an information need is, user is anyway brought to ask for it to a Web search engine, which will hopefully give the answer she expects.

Although the huge number of features which now the most popular Web search engines come with, in essence they still belongs to the category of Web-scale information retrieval tools. The results they provide in response to a user query are given according to the traditional "*ten blue links*" paradigm, i.e., links to Web pages that are considered *relevant* to the given user query. If results are not satisfactory, decision taken by looking at some of them, users may decide to "re-phrase" the query to try to refine the retrieved results.

However, when the need behind a certain query is a task to be accomplished, this "*query-look-refine*" paradigm could not satisfy user's expectation, which does not refer to simply consulting Web pages but, instead, aims at executing an activity. In other words, for certain (quite popular) tasks, Web search engines as we know can be considered obsolete tools.

We believe next-generation Web search engines should turn from mere Web documents retrieval tools to multifaceted systems, which fully support users while they are interacting with the Web. Of course, this opens up novel and exciting research challenges, in particular the ability to recognize implicit user tasks from the issued queries.

In this Chapter, we focus on discovering implicit user tasks from past issued queries, i.e., sets of possibly non contiguous queries phrased by the users of a Web search engine for carrying out various tasks. Since user search activities are recorded by most Web search engines into log files, i.e., *query logs*, interesting behaviors and patterns can be revealed by analyzing and mining Web search engine query logs [33, 111, 77, 61, 120].

In particular, a very important piece of information we can extract from a query log is represented by "*query sessions*", i.e. specific sets/sequences submitted by a user while interacting with a Web search engine. Sessions represent the basic unit of information for query suggestion [28], learning to rank [105], enhancing the interactions with the Web search engine [140], etc.

In the literature, there are many definitions of query sessions. Here, we are interested in identifying sessions composed of queries issued by users having in mind a particular *task* [124], i.e., *task-oriented sessions* or *Web-mediated tasks*. In our vision, a Web-mediated task can be thought as a *template* for representing any atomic activity, which is achievable by exploiting Web information: e.g., "*find a receipe*", "*book a flight*", "*read news*", etc.

So far, sessions are simply obtained by considering fixed-length time windows of successive queries entered by the same user. Unfortunately, this *time-based* detection methods fails in revealing such task-oriented sessions due to the *multi-tasking* users' behavior.

Multi-tasking refers to the way users interact with a Web search engine, by intertwining different tasks within a small time period. Therefore, the extraction of such task-oriented sessions firstly requires to detect whether pairs of user queries are task-related, in other words the goal is to compute their *task-based similarity*.

4.1.1 Contribution

This Chapter starts with an exhaustive analysis of our benchmark Web search engine query log, thereby showing that users perform multi-tasking search activities in the issued query streams. This makes it unsuitable to identify task-oriented sessions by only exploiting techniques that simply split the stream of queries. Thus, a more precise notion of a measure of the *task relatedness* between query pairs is needed. To this end, we first propose an *unsupervised* approach for measuring the task-based query similarity, which relies on a suitable selection of both *internal* and *external* query log features. Internal features are directly available from the original query log data, whereas external ones might be somehow derived by exploiting other data sources. This approach results in two *query similarity functions*. The first one, i.e., σ_1, is a simple *convex combination* of the selected query log features. The second one, i.e., σ_2, combines classical lexical content similarity measures, with the collaborative knowledge provided by *Wiktionary*[1] and *Wikipedia*[2] in a smart way. These external knowledge bases are used to enrich the meaning of each issued query, i.e., to "*wikify*" each query and thus to make more accurate decisions during the actual task-oriented session discovery.

Moreover, we propose a refined analysis showing the way in which task relatedness is determined. In fact, we also introduce and evaluate a *supervised* approach for establishing the task-based similarity between any two queries. Roughly, unlike the pure unsupervised approach, in which we exploit some query log features for computing a task-based query similarity function, here the task relatedness is *learned* by training a classifier on a manually built *ground-truth*, i.e., a set of task-oriented sessions manually detected over the queries submitted by several users. In particular, we exploit the binary classifier introduced by Jones *et al.* [67] and we use the prediction provided by such classifier in order to determine if two queries are task-related or not. Since this prediction comes with an *accuracy* value associated,

[1]`http://www.wiktionary.org`

[2]`http://www.wikipedia.org`

this is also used as a measure of how strong the task relatedness is between pairs of queries.

In more detail, we train a set of classifiers by taking into account all the features that Jones *et al.* [67] claim to be most suitable for predicting whether two queries belong to the same search goal. We extend the set of query log features on top of which the classifiers are trained by considering both Wikipedia, i.e., the "*wikification*" of the query, and the URL overlapping degree between the results associated with each query, i.e., the *Jaccard* similarity score between the top-20 results returned for each query.

Both unsupervised and supervised approaches for measuring the "task relatedness" between query pairs may be thus exploited by an effective method for discovering the actual task-oriented sessions. To this end, we propose a set of *clustering-oriented* approaches, i.e., *query clustering* methods, whose aim is to group together task-related queries, namely queries that are supposed to be part of the same task according to a specified task relatedness measure.

Therefore, we compare and evaluate the quality of all these methods by exploiting the manually generated ground-truth. In particular, we compare two techniques derived from well-known clustering algorithms, i.e., K-Means [85] and DB-Scan [44] with two other *graph-based* techniques for detecting Web-mediated tasks. The obtained results show that the latter two techniques perform better than the former. Indeed, they noticeably improve current state-of-the-art approaches.

Finally, experimental results show that combining supervised task relatedness learning to our best-performing clustering-oriented task discovery methods does not significantly enhance the overall effectiveness in discovering task-oriented sessions.

4.1.2 Organization

The rest of the Chapter is organized as follows. Section 4.2 describes related work on query log analysis and mostly focuses on search session boundaries detection. Section 4.3 provides the description and analysis of our benchmark data set, i.e., the 2006 AOL query log. In Section 4.4, we propose our theoretical model and the statement of the *Task-oriented Session Discovery Problem* (TSDP). Section 4.5 presents the construction of a ground-truth by manually grouping queries that are claimed to be task-related in a portion of our sample data set. Also, we propose some statistics on such corpus of manually identified tasks. Section 4.6 presents several approaches for measuring the task relatedness between query pairs, i.e., task-based query similarity functions, which in turn are exploited by the task-oriented session discovery methods proposed and compared in Section 4.7. Thus, Section 4.8 shows the experiments we conducted as well as the results we obtained. Finally, Section 4.9 presents our conclusions and points out any possible future work.

4.2 Related Work

Analysis of query logs collected by most Web search engines has increasingly gained interest across Web mining research community. Roughly, query logs record information about the *search activities* of users and so they are suitable data sources for understanding how people search the Web [120]. Moreover, Silvestri *et al.* [119] show a number of applications, i.e., caching, index partitioning, and document prioritization that can benefit from analysis performed on Web search engine logs.

Typical statistics that can be drawn from query logs either refer to *(i)* the analysis on the *query set* itself (e.g., query popularity, term popularity, average query length, distance between repetitions of queries or terms, etc.) or *(ii)* a higher level analysis of *search sessions*, i.e., sequences of queries issued by users for satisfying specific information needs.

The first study on a transaction log from a commercial Web search engine was conducted by Jansen *et al.* [62]. There, authors analyze a one-day log collected by the Excite search engine, which contains 51,473 queries issued by 18,113 users. Furthermore, Silverstein *et al.* [118] present an exhaustive analysis of a very large query log data set collected by the AltaVista search engine, which consists of about a billion queries submitted in a period of 42 days by approximately 285 million users. Authors show interesting results including the analysis of the query sessions for each user and the correlation among query terms.

However, most works concerning with mining of query logs aim to understand the real intent behind queries issued by users. Broder [33] claims that the "need behind the query" in Web context is not clearly *informational* like in classical information retrieval domain. Hence, he proposes a *taxonomy* of Web searches by classifying the queries according to their intent into three classes: *(i) navigational*, whose intent is to reach a specific Web site, *(ii) informational*, which aims to acquire some information from one or more Web documents, and finally *(iii) transactional*, whose intent is to perform some Web-mediated task. Moreover, Rose and Levinson [111] propose their own user search goals classification by extending the taxonomy devised in [33] adding more hierarchical levels. Lee *et al.* [77] describe *whether* and *how* search goal identification process behind a user query might be automatically performed on the basis of two features, that is *past user-click behavior* and *anchor-link distribution*.

Besides, many other works deal with the identification of users' search sessions boundaries. Previous works on session identification can be classified on the basis of their used method into: *(i) time-based*, *(ii) content-based*, and *(iii) ad-hoc* techniques, which usually combine both *(i)* and *(ii)*.

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

According to this vision, Silverstein *et al.* [118] firstly define a concept of "*session*" as follows: two consecutive queries are part of the same session if they are issued at most within a 5-minutes time window. Applying this definition to the data set they used, authors find 2.02 queries per session on the average. He and Göker [55] use different timeouts to split user sessions of the Excite query log, ranging from 1 to 50 minutes.

Radlinski and Joachims [105] observe that users often perform a sequence of queries with a similar information need and they refer to those sequences of reformulated queries as *query chains*. Their work presents a method for automatically detecting query chains in query and click-through logs using 30 minutes threshold for determining if two consecutive queries belong to the same search session.

Jansen and Spink [61] make a comparison of nine Web search engine transaction logs from the perspectives of session length, query length, query complexity, and content viewed. Here, they provide another definition of session, i.e. *search episode*, describing it as the period of time occurring from the first to the last recorded timestamp on the Web search engine server from a particular user in a single day, so that session length might vary from less than a minute to a few hours.

Moreover, using the same concept of search episode, Spink *et al.* [124] investigate also *multi-tasking* behaviors while users interacting with a Web search engine. Multi-tasking during Web searches involves the "seek-and-switch" process among several topics within a single user session. Again, a user session is defined to be the entire series of queries submitted by a user during one interaction with the Web search engine, so that session length might vary from less than a minute to a few hours. The results of this analysis performed on an AltaVista query log show that multi-tasking is a growing element in Web searching. In our work, we show the presence of multi-tasking also within shorter user activities.

Shi and Yang [116] describe the so-called *dynamic sliding window segmentation* method, which relies on three temporal constraints: α as the maximum time interval between two consecutive queries in the same session, β as the maximum inactivity time within the same session, and γ as the maximum length of a single session. They empirically estimate α, β, and γ to be 5 minutes, 24 hours, and 60 minutes, respectively.

Finally, Richardson [109] shows the value of long-term Web search engine query logs with respect to short-term, i.e., within-session, query information. He claims that long-term query logs can be used to better understand the world where we live, showing that query effects are long-lasting. Basically, in his work Richardson does not look at term co-occurrences just within a search session (which he agree to be a 30-minutes time window) but rather across entire query histories.

Content-based. Some works suggest to exploit the lexical content of the query itself for determining a possible topic shift in the stream of issued queries and thus a *session boundary* [75, 56, 93]. To this extent, several *search patterns* have been proposed by means of lexical comparison, using different string similarity scores (e.g., *Levenshtein*, *Jaccard*, etc.). However, approaches relying only on content features suffer of the so-called *vocabulary-mismatch problem*, namely the existence of topically-related queries without any shared terms (e.g., the queries "`nba`" and "`kobe bryant`" are completely different from a lexical content perspective but they are undoubtedly related). In order to overcome this issue, Shen *et al.* [115] compare "expanded representation" of queries, instead of the actual queries. Each individual expanded query is obtained by concatenating the titles and the Web-snippets for the top-50 results provided by a Web search engine for the specific query. Thus, the relatedness between query pairs can be computed using the *cosine similarity* between the corresponding expanded queries.

Ad-hoc. Jansen *et al.* [63] assume that a new search pattern always identifies the start of a new session. Moreover, He *et al.* [56] show that statistical information collected from query logs could be used for finding out the probability that a search pattern actually implies a session boundary. In particular, they extend a previous work [55] to consider both temporal and lexical information.

Similarly, Ozmutlu and Çavdur [93] describe a mechanism for identifying topic changes in user search behavior combining time and query content features. They test the validity of their approach using a genetic algorithm for learning the parameters of the topic identification task. The algorithm takes into account topic shift and continuation probabilities of the data set leveraging on query patterns (i.e., lexical content) and time intervals.

Seco and Cardoso [113] propose that a candidate query belongs to a new session if it does not have any common terms with the queries of the current session or the time interval between the candidate query and the last query in the current session is greater than 60 minutes.

Gayo-Avello [51] presents an exhaustive survey on session boundary detection methods and proposes a new technique, which works on the basis of a geometric interpretation of both the time gap and content similarity between consecutive query pairs.

Other approaches tackle the session boundary detection problem by leveraging on more complex models and by combining more features.

Boldi *et al.* [28] introduce the *Query Flow Graph* (QFG) as a model for representing data collected in Web search engine query logs. Intuitively, in the QFG a directed edge from query q_i to query q_j means that the two queries are likely to be part of the same search goal. Any path over the QFG may be seen as a searching behavior, whose likelihood is given by the strength of the edges along the path. The authors exploit this model for segmenting the query stream into sets of related information-seeking queries, reducing the problem to an instance of the *Asymmetric Traveling Salesman Problem* (ATSP).

Finally, Jones and Klinkner [67] address a problem that appears to be similar to ours. In particular, they argue that within a user's query stream it is possible to recognize specific hierarchical units, i.e., *search missions*, which are in turn divided into disjoint *search goals*. A search goal is defined as an atomic information need, resulting in one or more queries, while a search mission is a set of topically-related information needs, resulting in one or more goals.

Given a manually generated ground-truth, authors investigate how to *learn* a suitable binary classifier, which is aimed to precisely detect whether two queries belong to the same task or not. Among various results, they realize that timeouts, whatever their lengths, are of limited utility in predicting whether two queries belong to the same goal and thus unsuitable to identify session boundaries.

4.3 Query Log Analysis

The overall research challenge we want to address in this Chapter strongly relies on extracting useful patterns from Web search engine query log data. Indeed, meaningful analysis of query logs is often hard due to the lack of available data collections. To this extent, we used the 2006 AOL query log as our testing data set. This query log is a very large and long-term collection consisting of about 20 million of Web queries issued by more than 657,000 users over 3 months (from 1st March, 2006 to 31st May, 2006)[3].

4.3.1 Session Size Distribution

We have analyzed the entire AOL query log and we have extracted several statistics, such as the number of queries, the number of queries in each original *long-term session*, its average duration, etc.

The distribution of long-term sessions size on the entire collection, depicted in Figure 4.1, is characterized by a *Zipf's law* [151], i.e., one of a family of related discrete *power law* probability distributions [107].

Indeed, 67.5% of user sessions contains less than 30 queries, meaning that more than 2/3 of the users had issued about 10 queries per month on average. Besides, longer user sessions, i.e., sessions with more than 1,000 queries over 3 months, represent only $\approx 0.14\%$ of the whole data set.

To some extent, this result is compliant with the outcomes about the distribution of query frequencies and query term frequencies showed by Baeza-Yates *et al.* in [9]. Here, the authors observe that a small portion of the terms appearing in a large query log are used most often, while the remaining terms are individually used less

[3]`http://sifaka.cs.uiuc.edu/xshen/aol_querylog.html`

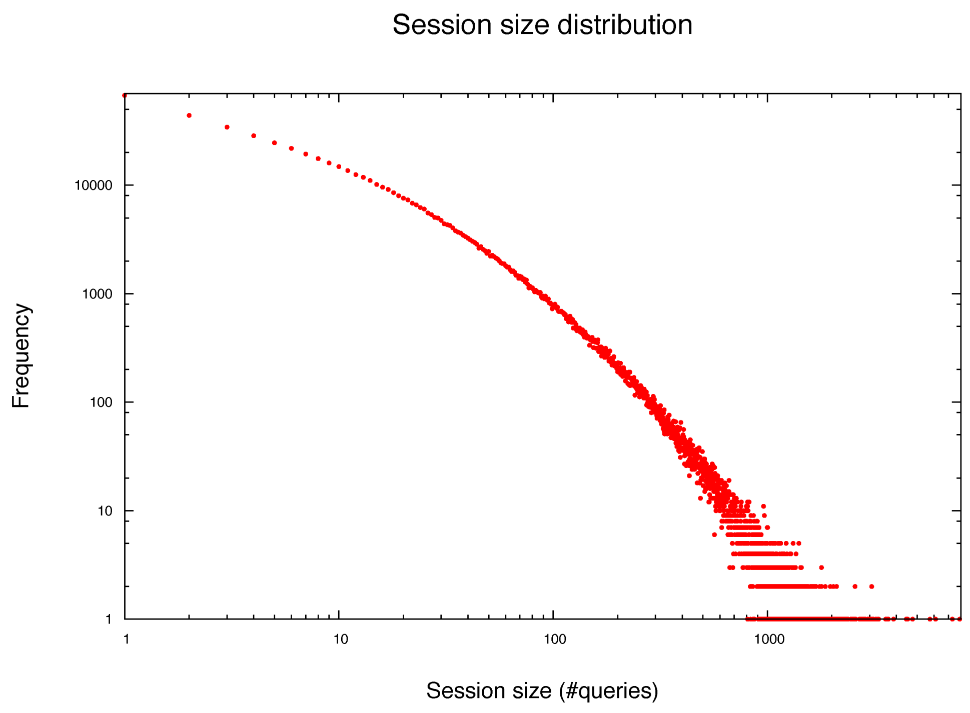


Figure 4.1: The distribution of long-term sessions size (log-log scale).

often. In the same way, a small portion of the user sessions contains many queries, while the remaining sessions are composed of few queries.

Finally, it is worth to remark that in our analysis we do not consider empty sessions, which globally account for $\approx 0.99\%$ of the total.

4.3.2 Query Time-Gap Distribution

Since users tend to issue *bursts* of queries for relatively short periods of time, which are usually followed by longer periods of inactivity, the *time gap* between queries plays a significative role in detecting session boundaries.

According to [118], we detect the session boundaries by considering the user's inactivity periods, i.e. the time gaps between consecutive queries in each long-term user session. To this end, we have to devise a suitable time threshold t_ϕ, which can be obtained by analyzing the distribution of time gaps between all the consecutive query pairs in our original and complete data set described before. We divide all the time gaps into several *buckets*, 60-seconds each. Therefore, we analyze the query inter-arrival times distribution, which, again, is revealed to be a power law (Figure 4.2).

This model tightly fits user behaviors during Web search activities, when consecutive queries issued within a short period of time are often not independent because they are also task-related.

More formally, given the following general form of a power law distribution $p(x)$:

$$p(x) = \frac{\alpha - 1}{x_{\min}} \left(\frac{x}{x_{\min}} \right)^{-\alpha},$$

where $\alpha > 1$ and $x_{\min}$ is the minimum value of x from which the law holds, we were interested in finding the value $\bar{x}$, such that two consecutive queries whose time gap is smaller than $\bar{x}$ are considered to belong to the same time-gap session.

Whenever the underlying distribution is unknown, it makes sense to assume a Gaussian distribution and use a threshold $\bar{x} = \mu + \sigma$ being equal to mean μ plus standard deviation σ, which results in "accepting" $\lambda = 84.1\%$ of the samples. This is equivalent to consider the cumulative distribution $P(\bar{x}) = Pr(X \leq \bar{x})$ and to determine $\bar{x}$, such that $P(\bar{x}) = \lambda$. Since we know the underlying distribution, and since $P(\bar{x}) = Pr(X \leq \bar{x}) = 1 - P'(\bar{x}) = 1 - Pr(X > \bar{x})$, we map the threshold λ into our context as follows:

$$P'(\bar{x}) = C \int_{\bar{x}}^{\infty} p(X)\, dX = \frac{\alpha - 1}{x_{\min}^{-\alpha+1}} \int_{\bar{x}}^{\infty} X^{-\alpha}\, dX = \left(\frac{\bar{x}}{x_{\min}} \right)^{-\alpha+1}.$$

Hence, for our purpose we have to solve the following equation in $\bar{x}$:

$$P(\bar{x}) = 1 - P'(\bar{x}) = 1 - \left(\frac{\bar{x}}{x_{\min}} \right)^{-\alpha+1} = \lambda = 0.841\ . \tag{4.1}$$

The value $x_{\min}$ represents the minimum query pair time gap and corresponds to the first interval, i.e., 60 seconds. Therefore, we estimated $\alpha = 1.564$ and finally we can solve Eq. 4.1 finding $\bar{x} \simeq 26$ minutes. This means to assume 84.1% of consecutive query pairs are issued within 26 minutes. Finally, we used this value $\bar{x}$, as the threshold t_{ϕ} for splitting each long-term user session of the the query log.

4.4 Task Discovery Problem

In this Section, we describe the theoretical model we adopt in order to formally define our research problem, i.e., finding user search sessions stored on a Web search engine query log whose final aim is to perform some tasks.

4.4.1 Theoretical Model

A Web search engine query log stores queries submitted by users, along with other information, such as userIDs, timestamps, etc. We denote with $\mathcal{QL}$ a Web search engine log of the queries submitted by a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$ during a given observation period. Moreover, let $q_i \in \mathcal{QL}$ be a generic query issued by user u_i and $q_{i,j} \in \mathcal{QL}$ be the j-th query issued by user u_i.

The methods that extract meaningful user sessions from $\mathcal{QL}$ have to analyze all the queries issued by each user u_i. Let $\mathcal{S}_i$ be the sequence of *all* the queries $q_i \in \mathcal{QL}$

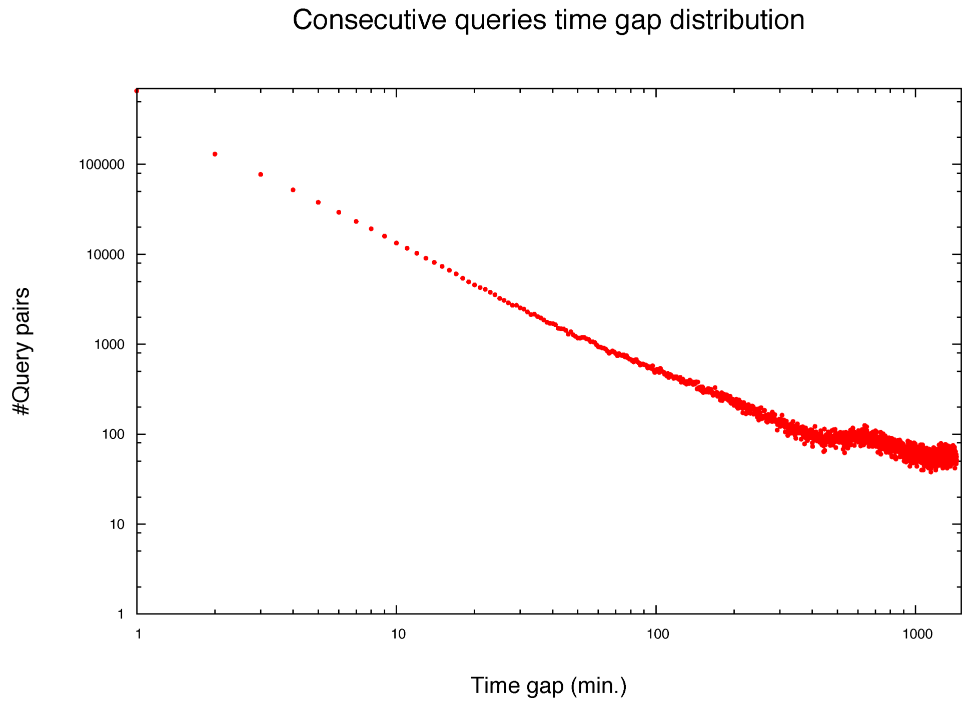


Figure 4.2: The distribution of time gaps between each consecutive query pair (log-log scale).

issued by user $u_i \in \mathcal{U}$, chronologically ordered during the period of observation recorded in the query log: $\mathcal{S}_i = \langle q_{i,1}, q_{i,2}, \ldots, q_{i,|\mathcal{S}_i|}\rangle$.
Therefore,

$$\mathcal{QL} = \bigcup_{i=1}^{N} \mathcal{S}_i.$$

The "bursty" nature of user query submission makes time interval between consecutive query pairs playing a significative role in detecting session boundaries. As it has been done by Silverstein *et al.* [118], we detect session boundaries by considering user's inactivity periods, i.e. the time gaps between consecutive queries in each $\mathcal{S}_i$.

Definition 4.4.1 (Time-Gap Session $\phi_{i,k}$) *Let $\tau(q_{i,j})$ bet the time at which the query $q_{i,j}$ is issued and t_ϕ be the maximum* time gap *threshold. The ordered set of consecutive queries $\phi_{i,k} = \langle q_{i,s_k}, \ldots, q_{i,e_k}\rangle \subseteq \mathcal{S}_i$, with $s_k < e_k$, is said to be a* time-gap session *if:* (i) $\tau(q_{i,j+1}) - \tau(q_{i,j}) \leq t_\phi$ *for every j, $s_k \leq j < e_k$; and* (ii) *there is no time-gap session being a superset of $\phi_{i,k}$.* □

This splitting technique makes no restrictions on the total elapsed time between the first and the last query of the sequence $\phi_{i,k}$. Usually, the inactivity threshold is fixed arbitrarily. In fact, in our tests we set $t_\phi = 26$ minutes. This parameter is obtained by analyzing the distribution of the time gaps in the query log used for the experiments as described later on Section 4.5.

We remark that here we are interested in studying to which extent in such time-gap sessions we can further recognize *task-oriented sessions*, i.e. sets of queries aimed at performing some Web-mediated tasks. Queries within the same task-oriented session do not necessarily occur consecutively in the time-gap session $\phi_{i,k}$. As a matter of fact, we will show that a generic user u_i usually interleaves many different information needs and related queries in each $\phi_{i,k}$.

Definition 4.4.2 (Task-oriented Session $\theta^j_{i,k}$) *Let $\phi_{i,k}$ be a time-gap session included in $\mathcal{S}_i$ and let $\theta^j_{i,k} \subseteq \phi_{i,k}$ be a* task-oriented session*, i.e., a set of (not necessarily consecutive) queries issued by user u_i for performing a given Web-mediated task. Such tasks form a disjoint partitioning of a time-gap session.* $\square$

We denote by $\Theta_{i,k} = \cup_j \theta^j_{i,k}$ all the task-oriented sessions in a given time-gap session $\phi_{i,k}$ and by $\Theta = \cup_{i,k}\Theta_{i,k}$ the set of all the task-oriented sessions in the query log $\mathcal{QL}$, i.e. the union-set of $\Theta_{i,k}$ for all users i and associated time-gap sessions k.

Therefore, the problem of finding Θ in a given query log can be formulated as the *Task-oriented Session Discovery Problem* (TSDP), whose goal is to find the best *query partitioning strategy* π that, when used to segment each time-gap session $\phi_{i,k}$ in $\mathcal{QL}$, approximates the *actual* user task-oriented sessions $\Theta_{i,k}$.

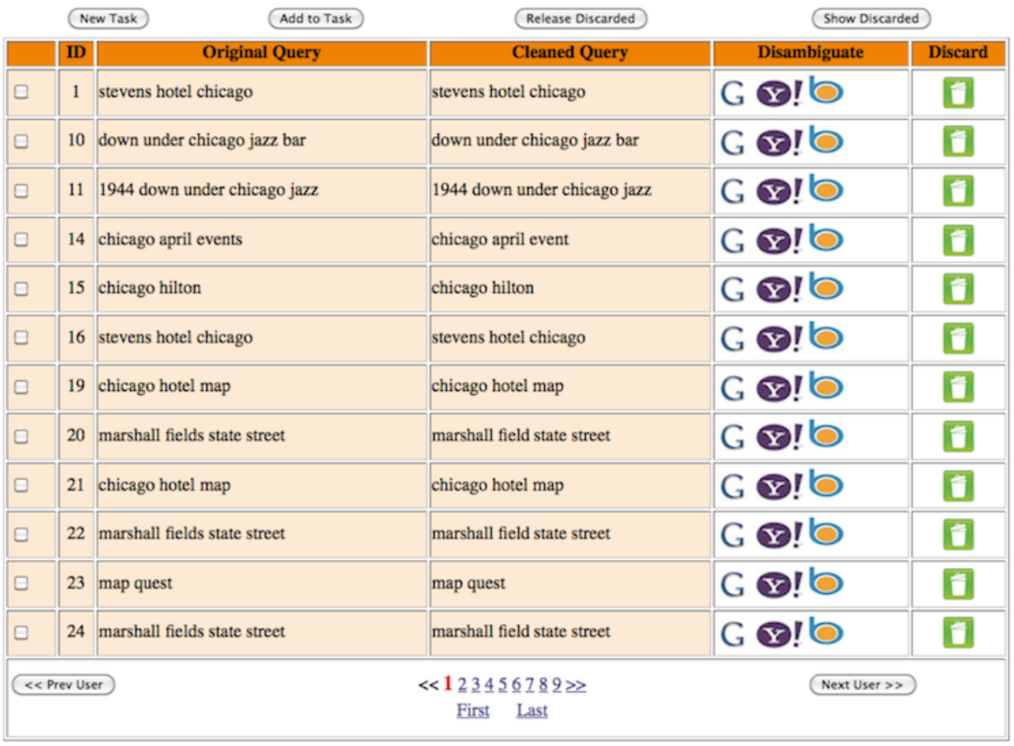
Task-oriented Session Discovery Problem (TSDP): Given a query log $\mathcal{QL}$, let $\mathcal{T}_{i,k} = \{t^1_{i,k}, t^2_{i,k}, \ldots\}$ be the task-oriented sessions discovered by the *query partitioning strategy* π, when applied to $\phi_{i,k}$, i.e., $\pi(\phi_{i,k}) = \mathcal{T}_{i,k}$. Also, let $\Theta = \cup_{i,k}\Theta_{i,k}$ and $\mathcal{T}_\pi = \cup_{i,k}\mathcal{T}_{i,k}$.
The TSDP requires to find the best partitioning $\bar{\pi}$, such that:

$$\bar{\pi} = \underset{\pi}{\operatorname{argmax}}\ \xi(\Theta, \mathcal{T}_\pi). \qquad (4.2)$$

Several quality measures can be used to evaluate the accuracy of a task-oriented session extraction, and consequently, several ξ functions can be devised. In Section 4.8, we instantiate ξ in terms of *F-measure*, *Rand index* and *Jaccard index*.

4.5 Ground-truth: Definition and Analysis

According to the Task-oriented Session Discovery Problem statement defined in Section 4.4.1, we need to find the query partitioning strategy $\mathcal{T}_{\bar{\pi}}$ that best approximates the *actual* task-oriented segmentation Θ. Such optimal task-oriented partitioning can be manually built from real Web search engine query log data. To this end, we develop a Web application that helps human assessors to manually identify the optimal task-oriented query sessions from the previously prepared AOL query log, thus producing a *ground-truth* that can be used for evaluating automatic task-oriented session discovery methods and which is also freely available for download[4]. Figure 4.3 shows a snapshot of the Web application we used to annotate and produce the ground-truth used in our experiments.



	ID	Original Query	Cleaned Query	Disambiguate	Discard
☐	1	stevens hotel chicago	stevens hotel chicago		
☐	10	down under chicago jazz bar	down under chicago jazz bar		
☐	11	1944 down under chicago jazz	1944 down under chicago jazz		
☐	14	chicago april events	chicago april event		
☐	15	chicago hilton	chicago hilton		
☐	16	stevens hotel chicago	stevens hotel chicago		
☐	19	chicago hotel map	chicago hotel map		
☐	20	marshall fields state street	marshall field state street		
☐	21	chicago hotel map	chicago hotel map		
☐	22	marshall fields state street	marshall field state street		
☐	23	map quest	map quest		
☐	24	marshall fields state street	marshall field state street		

Figure 4.3: A snapshot of the Web application used for generating the ground-truth.

Human annotators group together queries that they claim to be task-related within each time-gap session. Also, they have chance to discard meaningless queries from those sessions. For each manually identified task (i.e., set of task-related queries), evaluators have to add a *tag* and, optionally, a longer description. Such data source could possibly represent a semantic knowledge base of users search goals (i.e., *taxonomy of tasks*).

Furthermore, dealing with such massive data sets typically needs a *pre-processing* phase in order to "clean" the collection and to make it more suitable for performing analysis steps. In particular, pre-processing concerned the following aspects. First of all, we removed query log records containing both empty and "nonsense" query strings (e.g., query strings composed of only punctuation symbols). Also, we removed all the *stop-words* from each query string. Then, we run the *Porter stemming algorithm* [102] for removing the most common morphological and inflectional English endings from the terms of each query string. Finally, the data cleaning

[4]`http://miles.isti.cnr.it/~tolomei/?page_id=36`

phase involved removing the long-term user sessions containing too much queries, which were probably generated by *robots*, instead of human users.

Therefore, we considered as a sample the 500 user sessions with the highest number of queries, hereinafter the *top*-500 data set. This data set contains a total amount of 518, 790 queries, meaning that each user issued on average ≈ 1, 038 queries in 3 months, i.e., ≈ 12 queries per day. The maximum number of queries in a user session is 7, 892 and the minimum is 729. As a consequence, this means users submitted from a minimum of ≈ 8 to a maximum of ≈ 88 queries per day. As a last remark, human evaluators are people selected from our laboratory, but not directly involved in this work.

Manual annotation concerns a total of 2, 004 queries, from which 446 time-gap sessions are extracted automatically. A total of 139 time-gap session are discarded as meaningless by the annotators and, therefore, they are removed from the ground-truth. Eventually, 1, 424 queries are actually clustered from 307 time-gap sessions.

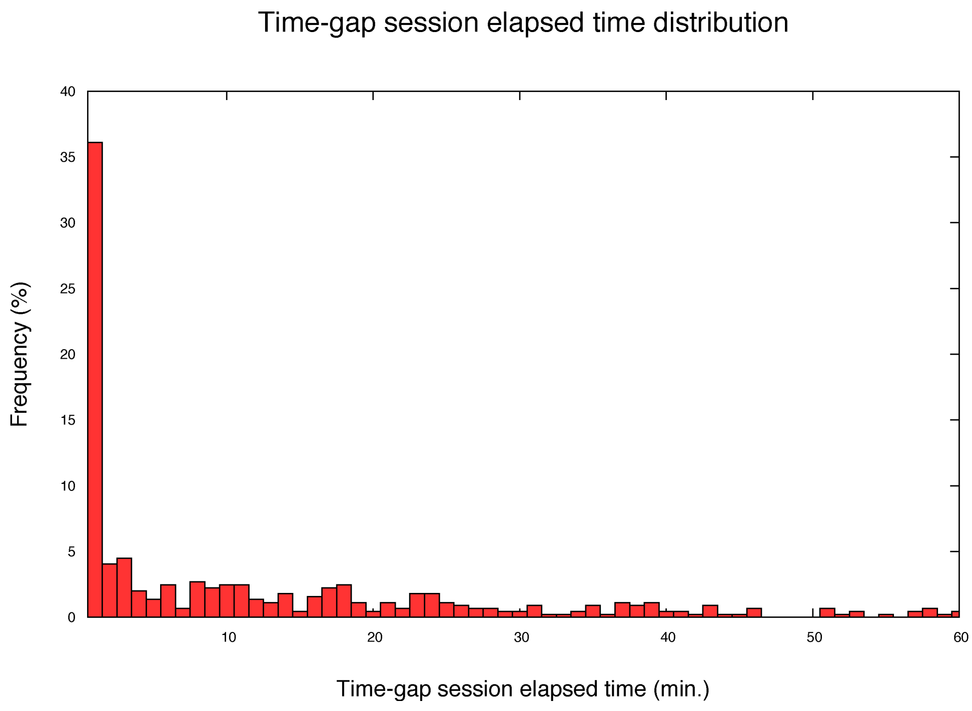


Figure 4.4: The distribution of time-gap session duration.

Figure 4.4 shows the distribution of time-gap session length, using a discretization factor of 60 seconds. While there are many sessions being shorter than 1 minute, usually containing only one or two queries, the duration of a time-gap session is 15 minutes on average. Indeed, we observe, with a non-negligible frequency, sessions lasting for 40 (or more) minutes. Also, in this cases, the session length suggests that the interaction of users with Web search engines is non trivial and it is likely to involve multi-tasking. The longest time-gap session lasts 9207 seconds, i.e. about 2 hours and a half and this happens only once in our data set.

In Figure 4.5 we report the time-gap session size distribution. On average, each time-gap session contains 4.49 queries, the sessions with at most 5 queries cover slightly more than half of the query log. The other half of the query log contains longer sessions with high probability of having multiple tasks being carried on in the same session.

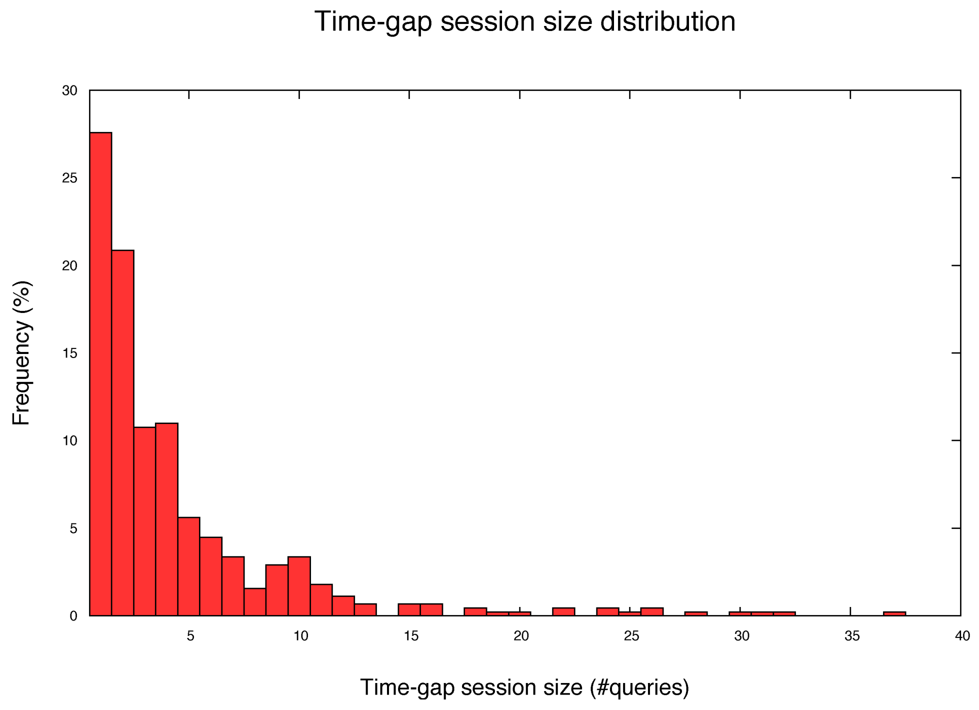


Figure 4.5: The distribution of time-gap session size.

The total number of human annotated task-oriented sessions is 554, with an average of 2.57 queries per task. The distribution of the task-oriented sessions size is illustrated in Figure 4.6. The number of tasks accomplished in a time-gap session is 1.80 (Figure 4.7).

Among all the 307 time-based sessions considered, 145 contain multiple tasks. We find that this 50% split between single-tasking and multi-tasking sessions is consistent across the various users. Interestingly enough, this shows that a good task detection algorithm have to be able to handle efficiently both single and multi-tasking sessions.

If we consider all the queries included in each task, then $1,046$ out of $1,424$ queries are included in multi-tasking sessions, meaning that about 74% of the user activity is multi-tasking.

Finally, we also evaluate the degree of multi-tasking by taking into account the number of overlapping task-oriented sessions. We say that a *jump* occurs whenever two queries in a manually labelled task-oriented session are not consecutive. For instance, let $\phi = \langle q_1, q_2, \ldots, q_9 \rangle$ be a time-gap session and let $\pi(\phi) = \{\theta_1, \theta_2, \theta_3\}$ be the result of the manual annotation procedure for ϕ, where $\theta_1 = \{q_1, q_2, q_3, q_4\}$,

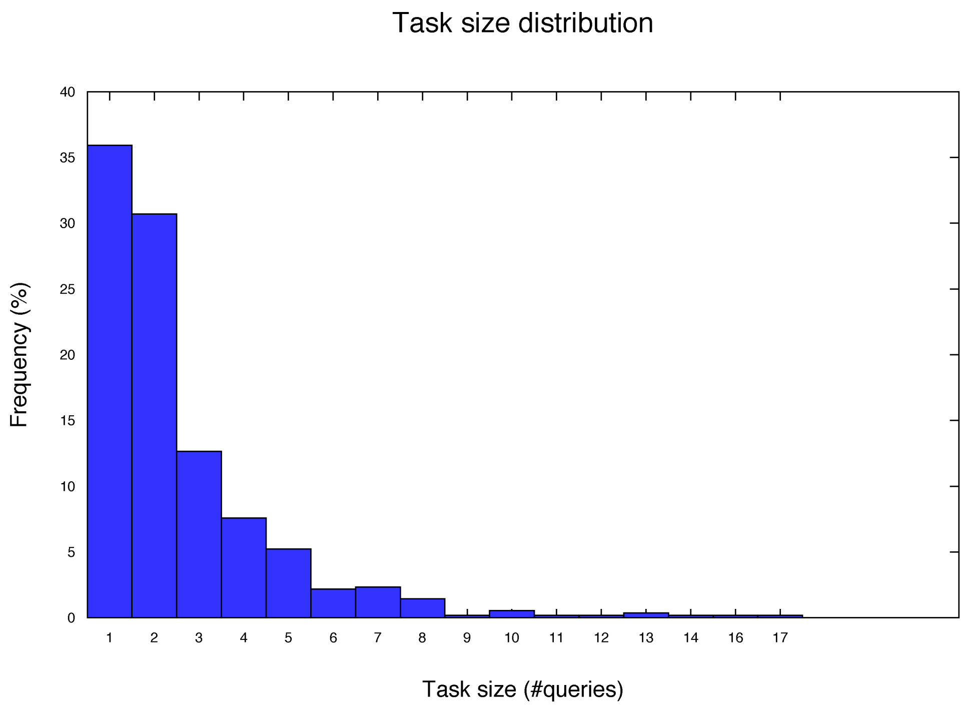


Figure 4.6: The distribution of task-oriented session size.

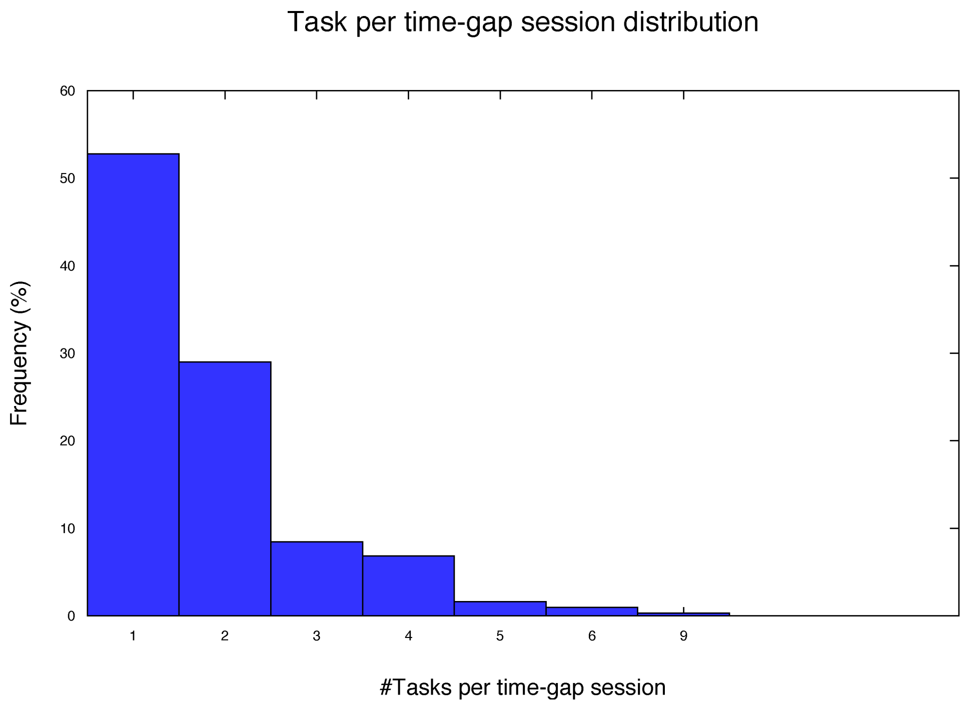


Figure 4.7: The distribution of tasks per time-gap session.

$\theta_2 = \{q_5, q_7\}$ and $\theta_3 = \{q_6, q_8, q_9\}$. In this case, the number of jumps observed in ϕ is 2, because there are two query pairs $(q_5, q_7) \in \theta_2$ and $(q_6, q_8) \in \theta_3$, which are not consecutive.

The number of jumps gives a measure of the *simultaneous* multi-tasking activity. We denote with $j(\phi)$ the simultaneous multi-tasking degree of ϕ as the ratio of task-oriented sessions in ϕ having at least one jump. In the previous example $j(\phi) \simeq 0.67$, since 2 out of 3 tasks contain at least one jump.

In Figure 4.8, we show the distribution of the multi-tasking degree over all the time-gap sessions. Note that the result for $j(\phi) = 0$ is omitted, because we already know that 50% of the sessions are single-tasking.

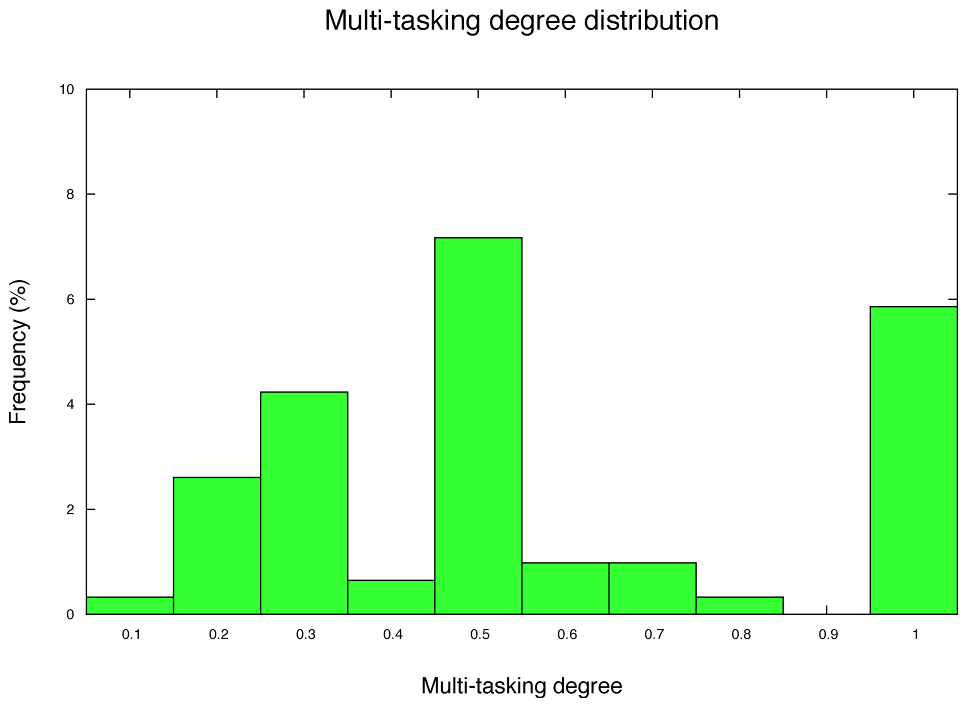


Figure 4.8: The distribution of multi-tasking degree.

4.6 Task-based Query Similarity

We define the measure of how likely two queries are part of the same search task as *task relatedness*. In other words, task relatedness measures how *similar* two queries are, with respect to a hypothetic common search task.

In this Section, we describe three different ways of computing the task relatedness between any two queries (q_i, q_j), which results in several *task-based query similarity functions*. The first approach, i.e., *time-based*, is the most simple and takes into account only the time gap between two adjacent queries. Therefore, we present an *unsupervised* approach for computing task-based query similarity functions. Such similarity functions exploit a set of features extracted from query log data. Finally, we introduce a *supervised* approach, which *learns* task relatedness by training a classifier on a set of features derived from the query log itself.

4.6.1 Time-based Approach

In the literature, the simplest approach for measuring task-based similarity between two queries is that proposed by [118], which *only* takes into account the query submission time. This measure is based on the assumption that if two *consecutive* queries are issued within a "*small enough*" time window then they are also likely to be related. In fact, having two consecutive queries whose time gap is greater than a certain value could reveal a topic shift in user search behavior.

The main issue with this approach is how to effectively define what a "small enough" time window means, i.e., to devise a time threshold $\bar{t}$ on the inter-query submission time, so that as long as two adjacent queries are issued within $\bar{t}$ they are also high probably task-related. On the basis of this threshold, it can then be devised a time-based *binary* similarity function σ_{time}, only defined for adjacent query pairs as follows:

$$\sigma_{time}(q_i, q_j) = \begin{cases} 1 & \text{if } |\tau(q_j) - \tau(q_i)| \leq \bar{t} \text{ and } j == i+1, \\ 0 & \text{if } |\tau(q_j) - \tau(q_i)| > \bar{t} \text{ and } j == i+1, \\ \perp & \text{otherwise.} \end{cases} \quad (4.3)$$

Of course, the overall effectiveness of this approach changes by varying the value chosen for the threshold $\bar{t}$. Several past works perform their experimental evaluation by empirically testing different values of $\bar{t}$, ranging from 5 to 60 minutes thresholds [118, 116, 109]. Instead, in this work we devise this threshold value directly from the query log data, as specified in Section 4.3.2.

4.6.2 Unsupervised Approach

Following this approach, we first have to select the set of query log features we want to take care of for determining the task-based query similarity. Thus, in the remaining of this Section we describe a selection of features derived from query log data, either *internally* or *externally* available, which we claim to be useful in establishing if two queries are task-related. Therefore, we also figure out two task-based query similarity functions, which combine the selected features in two different ways.

4.6.2.1 Feature Selection

Evaluating the similarity between two queries is a very complex issue and even harder is determining their similarity in terms of a search task, i.e., their task relatedness. Most of the previous approaches are based on the similarity between query lexical content [112]. The precision of those approaches results to be quite low due to the short length of queries [118] and the lack of the contextual information in which queries are issued [139]. Thus, some approaches try to expand those short queries by exploiting resulting URLs returned by Web search engines [52], or the returned Web-snippets [80], or the documents themselves [106]. Two queries might be considered similar if they return similar results, or similar documents. Unfortunately, it might be the case for unrelated queries to share some results.

Here, we propose two features and two similarity measures for assessing the task relatedness of two queries q_i and q_j, both in terms of their lexicographical content and their semantics.

***Content-based* ($\sigma_{content}$).** Two queries that share some common terms are likely related. Sometimes, such terms may be very similar, but not identical, due to misspelling, or different prefixes/suffixes. To capture content similarity between queries, we adopt a *Jaccard* index on tri-grams [66]. Let $T(q)$ be the tri-grams resulting from the terms of query q, we define the similarity $\sigma_{jaccard}$ as follows:

$$\sigma_{jaccard}(q_i, q_j) = \frac{|T(q_i) \cap T(q_j)|}{|T(q_i) \cup T(q_j)|}.$$

In addition, we exploit a normalized *Levenshtein* similarity $\sigma_{levenshtein}$, which Jones and Klinker [67] claimed to be the best edit-based feature for identifying goal boundaries. Finally, the overall content-based similarity is computed as follows:

$$\sigma_{content}(q_i, q_j) = \frac{\sigma_{jaccard}(q_i, q_j) + \sigma_{levenshtein}(q_i, q_j)}{2}.$$

***Semantic-based* ($\sigma_{semantic}$).** We are interested in finding a measure of the *semantic relatedness* between query pairs. Typically, humans can easily judge the semantic relatedness between two terms. This human ability is backed by their experience and knowledge, which makes it a hard task for machines. If a machine has to solve this task, it also needs some source of knowledge. Usually, this knowledge comes from: *(i)* large text collections, i.e., *corpora*, or from *(ii)* semantic resources. Thus, we figured out that we could expand each query with its "*wikification*". Basically, we exploit both *Wiktionary* and *Wikipedia* external data sources for increasing the meaningfulness of each query, trying to overcome its lack of semantic information.

Several semantic relatedness metrics dealing with semantic resources have been proposed in the past. Roughly, they can be classified into:

- ***path-based*:** represent knowledge as a graph of concepts and compute paths over that graph [104, 76];

- ***information content-based*:** take into account the information content of a concept [108];

- ***gloss-based*:** rely on term overlaps between definitions of concepts [79];
- ***vector-based*:** model each concept as a vector of anchor links [92] or terms [50].

Following the last approach, we assume that a Wiktionary entry, or a Wikipedia article, describes a certain concept and that the presence of a term in a given article is an evidence of the correlation between that term and that concept. Thus, we describe the *wikification* $\overrightarrow{C}(t)$ of a term t as its representation in a high dimensional concept space $\overrightarrow{C}(t) = (c_1, c_2, \ldots, c_W)$, where W is the number of articles in our collections and c_i scores the relevance of the term t for the i-th article. We measure this relevance by using the well known *tf-idf* score [112].

In order to "*wikify*" the whole string associated with a query q, we sum up the contribution from its terms, i.e.:

$$\overrightarrow{C}(q) = \sum_{t \in q} \overrightarrow{C}(t).$$

Then, we compute the relatedness $\sigma_{wikification}(q_i, q_j)$ between two queries q_i and q_j as the *cosine similarity* of their corresponding concept vectors:

$$\sigma_{wikification}(q_i, q_j) = \frac{\overrightarrow{C}(q_i) \cdot \overrightarrow{C}(q_j)}{\|\overrightarrow{C}(q_i)\| \|\overrightarrow{C}(q_j)\|}.$$

Of course, we use the same approach both for computing $\sigma_{wiktionary}$ and $\sigma_{wikipedia}$ similarity measures, taking into account Wiktionary and Wikipedia corpora, respectively. Finally, the overall semantic-based similarity is obtained as follows:

$$\sigma_{semantic}(q_i, q_j) = \max(\sigma_{wiktionary}(q_i, q_j), \sigma_{wikipedia}(q_i, q_j)).$$

4.6.2.2 Similarity Functions

An immediate way to put together *(i)* the lexical content ($\sigma_{content}$) and *(ii)* the semantic expansion ($\sigma_{semantic}$) similarity is via a *convex combination*:

$$\sigma_1 = \alpha \cdot \sigma_{content} + (1 - \alpha) \cdot \sigma_{semantic}\ . \tag{4.4}$$

In addition, we propose a novel *conditional* similarity function σ_2 based on the following heuristic: if the content-based similarity between two queries is equal to or greater than a certain threshold $\mathbf{t}$ then we can be confident that queries are also task-related; otherwise, we look at the semantic expansion of the queries and we compute the final similarity score as the maximum between content-based and semantic-based similarity values.

$$\sigma_2 = \begin{cases} \sigma_{content} & \text{if } \sigma_{content} \geq \mathbf{t}, \\ \max(\sigma_{content}, \mathbf{b} \cdot \sigma_{semantic}) & \text{otherwise.} \end{cases} \tag{4.5}$$

The rationale for introducing the *conditional* similarity function σ_2 is based on the following conjecture: *if two queries are close in term of lexical content, the semantic expansion could be unhelpful.* Vice-versa, nothing can be said when queries do not share any common content feature (e.g., again consider the two queries "*nba*" and "*kobe bryant*").

Both σ_1 and σ_2 rely on the estimation of some parameters, i.e., α for σ_1 and $\mathbf{t}$ as well as $\mathbf{b}$ for σ_2, which are learned directly from the ground-truth.

4.6.3 Supervised Approach

In Section 4.2, we pointed out that Jones and Klinkner [67] addressed a problem which is similar to the TSDP. They argue that user search activities can be divided into particular hierarchical units, i.e., *search missions*, which are in turn composed of disjoint *search goals*. According to the authors, a search goal is defined as an atomic information need, resulting in one or more queries, while a search mission is a set of topically-related information needs, resulting in one or more goals. Up to here, there is a clear similarity between a search goal and what we have defined to be a task-oriented session, according to Def. 4.4.2.

However, in [67] authors investigate a completely *supervised* approach in order to learn a suitable binary classifier for precisely detecting whether two queries belong to the same goal/mission or not. Basically, given a manually generated *ground-truth*, they exploit several characteristics (i.e., temporal, word- and character-edit, query log sequence, and Web search features) for predicting whether two queries belong to the same goal/mission and thus to identify goal/mission boundaries. Anyway, authors do not explore how such binary classifier could be exploited for segmenting users' query streams into goals and missions, thus for actually *discovering* search goals, i.e., task-oriented sessions and search missions.

Nevertheless, it is worth noting that this supervised approach could also be used for deriving another task-based query similarity measure, which in turn might be further exploited by task discovery methods. In particular, the task relatedness between two queries (q_i, q_j) could be derived from the prediction value with which q_i and q_j are classified in the same search goal, i.e., *same task*.

The rest of this Section is structured as follows. First, we start by describing the set of features $\mathcal{F}_{jk}$ that Jones and Klinkner [67] claim to provide the best accuracy results in predicting whether two queries belong to the same task or not. Moreover, as another major novel contribution of this work, we introduce a set of features $\mathcal{F}'$ that might be combined with $\mathcal{F}_{jk}$. Therefore, we train several binary classifiers using various combination of features on top of our manually generated ground-truth described in Section 4.5. Finally, we exploit the output of such classifiers for determining 12 new task-based query similarity functions, each of which will be in turn exploited by two task discovery methods, as specified in Section 4.7.2.1.

4.6.3.1 Feature Selection

According to [67], given any two queries q_i and q_j, the following set of features $\mathcal{F}_{jk}$ provides best accuracy results in predicting whether they are part of the same task:

– *edlevGT2*: this is a binary feature that evaluates to 1 if the normalized *Levenshtein* edit distance between q_i and q_j is greater than 2, 0 otherwise;

– *wordr*: this feature corresponds to the *Jaccard* distance between the sets of words which q_i and q_j are composed of;

– *char_suf*: this feature counts the number of common characters between q_i and q_j, starting from the right;

– *nsubst_q_j_X*: given $P(q_i \longrightarrow q_j)$ the probability of q_i being reformulated as q_j, this feature is computed as $count(X : \exists\ P(q_j \longrightarrow X))$;

– *time_diff*: this feature represents the inter-query time gap between q_i and q_j, expressed in seconds;

– *sequential*: this binary feature is positive if the queries q_i and q_j are sequentially issued;

– *prisma*: this feature refers to the *cosine distance* between vectors obtained from the top-50 search results for the terms of q_i and q_j, respectively;

– *entropy_q_i_X*: this feature concerns with the *entropy* of rewrite probabilities from query q_i and it is computed as $\sum_k P(q_k|q_i)\log_2(P(q_k|q_i))$.

In addition to the above set $\mathcal{F}_{jk}$, we propose another set of features $\mathcal{F}'$, as a novel contribution of this work. This is composed of a semantic feature we already used for computing the task relatedness measures of our unsupervised approaches, i.e., $\sigma_{wikipedia}$ (Section 4.6.2.1). Moreover, $\mathcal{F}'$ also contains another feature, i.e., $\sigma_{jaccard\_url}$, which measures the Jaccard similarity between the top-20 domain URLs returned as search results in response to q_i and q_j, respectively. The rationale of this last feature is to capture the similarity of two apparently different queries, which share many relevant links to Web documents, i.e., URLs, of the retrieved results provided by most popular Web search engines. Thus, given $url_{20}(q_i) = \{u_i^1, u_i^2, \dots, u_i^{20}\}$ the set of top-20 domain URL results returned by a Web search engine in response to the query q_i, this feature is computed as follows:

$$\sigma_{jaccard\_url}(q_i, q_j) = \frac{|url_{20}(q_i) \cap url_{20}(q_j)|}{|url_{20}(q_i) \cup url_{20}(q_j)|}.$$

4.6.3.2 Binary Classifiers

Following the approach presented in [67], we exploit both the sets of features previously described in Section 4.6.3.1 for training several binary classifiers. In particular, we devise four different combinations of features extracted from our manually generated ground-truth described in Section 4.5:

- $\mathcal{F}_1 \equiv \mathcal{F}_{jk}$;
- $\mathcal{F}_2 = \mathcal{F}_{jk} \cup \sigma_{wikipedia}$;
- $\mathcal{F}_3 = \mathcal{F}_{jk} \cup \sigma_{jaccard\_url}$;
- $\mathcal{F}_4 = \mathcal{F}_{jk} \cup \mathcal{F}'$.

Moreover, we decide to use three different classification models:

- $\mathcal{C}_{dt}$: a clone of the C4.5 Decision Tree learner [103];
- $\mathcal{C}_{nb}$: a Naïve Bayesian learner;
- $\mathcal{C}_{lr}$: a Logistic Regression learner.

Therefore, the classification step requires the training of the set of classifiers, i.e., $\mathcal{C} = \{\mathcal{C}_{dt}, \mathcal{C}_{nb}, \mathcal{C}_{lr}\}$, over the set of feature sets, i.e., $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4\}$. By combining each of the three classifier models with each feature set, we obtain 12 distinct classifiers, i.e., $\mathcal{C}_x^y$, where $x \in \{dt, nb, lr\}$ and $y \in \{1, 2, 3, 4\}$.

All these classifiers are trained on the manually built ground-truth described in Section 4.5. Roughly, for each query pair (q_i, q_j) of the ground-truth we extract a specific set of features $\mathcal{F}_y \in \mathcal{F}$ and we assign the *class attribute* "`same_task = yes`" if and only if q_i and q_j are part of the same task in the ground-truth, "`same_task = no`" otherwise. Furthermore, we run a classifier model $\mathcal{C}_x \in \mathcal{C}$ on such training data set for building a binary classifier, which could be able to predict if any two queries are part of the same task-oriented session or not. Moreover, each prediction made by a certain classifier for a specific query pair comes with an *accuracy* value, i.e., a measure of the "*strength*" of the provided prediction. Thus, such accuracy value could also be used as a task relatedness measure between each query pair. Of course, by training 12 different binary classifiers, we generally have 12 distinct accuracy values for each query pair (q_i, q_j) in predicting if q_i and q_j are task-related.

4.6.3.3 Similarity Functions

According to Section 4.6.3.2, the accuracy of prediction provided by a binary classifier for estimating if any two queries belong to the same search task could be interpreted as a *similarity score*, which in turn could be exploited by any clustering-oriented technique that will be presented in Section 4.7.2.1.

Here, we propose 12 new similarity functions, which exploit the output of the same binary classifiers. Table 4.1 contains the name of each similarity function obtained as combination of the classifier model and the set of exploited features.

		Feature Set			
		$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
Classification Model	$\mathcal{C}_{dt}$	σ^1_{dt}	σ^2_{dt}	σ^3_{dt}	σ^4_{dt}
	$\mathcal{C}_{nb}$	σ^1_{nb}	σ^2_{nb}	σ^3_{nb}	σ^4_{nb}
	$\mathcal{C}_{lr}$	σ^1_{lr}	σ^2_{lr}	σ^3_{lr}	σ^4_{lr}

Table 4.1: Supervised task-based query similarity functions.

Let $acc^y_x(q_i, q_j) \in [0, 1]$ be the accuracy value of the classifier $\mathcal{C}_x$ trained on the set of features $\mathcal{F}_y$ in predicting that q_i and q_j are part of the same search task. Thus, each similarity function σ^y_x, where $x \in \{dt, nb, lr\}$ and $y \in \{1, 2, 3, 4\}$ might be instantiated as follows:

$$\sigma^y_x(q_i, q_j) = acc^y_x(q_i, q_j). \tag{4.6}$$

However, in order to detect which similarity functions are supposed to provide better results, we first have to compare the performances of the classifiers which they are derived from.

Typically, the performance of a classifier is measured by counting the proportion of correctly predicted examples in an unseen data set, i.e., *test data set*, which is different from the *training data set*. This proportion is called *1-ErrorRate* (or also *accuracy*). However, a more elaborate method for measuring the performance of a classifier is *cross-validation*, where a number n of folds is specified. Here, the training data set is randomly reordered and then split into n folds of equal size. At each iteration, one fold is used for testing and the other $n - 1$ folds are used for training the classifier. The test results are collected and averaged over all folds and this provides the cross-validation estimate of the accuracy.

In the following, we describe the performances of each binary classifier obtained using a specific classification model and varying the set of training features. All the evaluations are measured according to 10-fold cross-validation.

We decide to express the performance of each binary classifier in terms of *TP Rate*, *FP Rate*, *Precision*, *Recall*, and *F-measure*. All these measures are opportunely averaged with respect to the two distinct prediction classes, i.e., "`same_task = yes`" and "`same_task = no`". In particular, *TP Rate* refers to the ratio of *true positive* examples, i.e., examples that are correctly classified for both prediction classes and it is equivalent to *Recall*. Similarly, *FP Rate* describe the ratio of *false positive* examples, i.e., examples that are misclassified for both prediction classes. Moreover, *Precision* is the proportion of the examples which truly have a certain class among all those which are classified with that class. Finally, *F-measure* is the weighted harmonic mean of *Precision* and *Recall*:

$$\textit{F-measure} = \frac{2 \times \textit{Precision} \times \textit{Recall}}{(\textit{Precision} + \textit{Recall})}.$$

Decision Tree Classifier ($\mathcal{C}_{dt}$)***.*** This classification model is based on a clone of the C4.5 Decision Tree learner [103]. Table 4.2 describes the performance of four binary classifiers, i.e., $\mathcal{C}^1_{dt}, \mathcal{C}^2_{dt}, \mathcal{C}^3_{dt}$, and $\mathcal{C}^4_{dt}$, obtained using this model in combination with the sets of features $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$, and $\mathcal{F}_4$, respectively.

		Accuracy Measures				
		TP Rate	*FP Rate*	*Precision*	*Recall*	*F-measure*
Classifiers	$\mathcal{C}^1_{dt}$	**0.987**	0.463	0.985	**0.987**	0.985
	$\mathcal{C}^2_{dt}$	**0.987**	0.457	0.985	**0.987**	0.985
	$\mathcal{C}^3_{dt}$	**0.987**	0.457	**0.986**	**0.987**	**0.986**
	$\mathcal{C}^4_{dt}$	**0.987**	**0.450**	**0.986**	**0.987**	**0.986**

Table 4.2: Performance evaluation of the set of classifiers derived from $\mathcal{C}_{dt}$.

All the binary classifiers behave similarly. However, the very best performance is obtained with $\mathcal{C}^4_{dt}$, which is trained using the set of features $\mathcal{F}_4$. This means that the similarity function of choice for this classification model is σ^4_{dt}.

Naïve Bayesian Classifier ($\mathcal{C}_{nb}$)***.*** This classification model is based on a Naïve Bayesian learner. Table 4.3 describes the performance of four binary classifiers, i.e., $\mathcal{C}^1_{nb}, \mathcal{C}^2_{nb}, \mathcal{C}^3_{nb}$, and $\mathcal{C}^4_{nb}$, obtained using this model in combination with the sets of features $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$, and $\mathcal{F}_4$, respectively.

		Accuracy Measures				
		TP Rate	*FP Rate*	*Precision*	*Recall*	*F-measure*
Classifiers	$\mathcal{C}^1_{nb}$	**0.982**	**0.488**	**0.981**	**0.982**	**0.981**
	$\mathcal{C}^2_{nb}$	0.981	0.516	0.980	0.981	0.980
	$\mathcal{C}^3_{nb}$	**0.982**	0.489	**0.981**	**0.982**	**0.981**
	$\mathcal{C}^4_{nb}$	0.981	0.515	0.980	0.981	0.980

Table 4.3: Performance evaluation of the set of classifiers derived from $\mathcal{C}_{nb}$.

As for the Decision Tree model, all the binary classifiers perform similarly. However, the very best accuracy is obtained both with $\mathcal{C}^1_{nb}$ and $\mathcal{C}^3_{nb}$, thus using the set of features $\mathcal{F}_1$ and $\mathcal{F}_3$, respectively. Here, the chosen similarity functions for this classification model might be either σ^1_{nb} or σ^3_{nb}.

Logistic Regression Classifier ($\mathcal{C}_{lr}$). This classification model is based on Logistic Regression. Table 4.4 describes the performance of four binary classifiers, i.e., $\mathcal{C}^1_{lr}, \mathcal{C}^2_{lr}, \mathcal{C}^3_{lr}$, and $\mathcal{C}^4_{lr}$, obtained using this model in combination with the sets of features $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$, and $\mathcal{F}_4$, respectively.

		Accuracy Measures				
		TP Rate	*FP Rate*	*Precision*	*Recall*	*F-measure*
Classifiers	$\mathcal{C}^1_{lr}$	0.983	0.617	**0.981**	0.983	**0.981**
	$\mathcal{C}^2_{lr}$	0.983	0.616	**0.981**	0.983	**0.981**
	$\mathcal{C}^3_{lr}$	0.983	0.612	**0.981**	0.983	**0.981**
	$\mathcal{C}^4_{lr}$	**0.984**	**0.609**	**0.981**	**0.984**	**0.981**

Table 4.4: Performance evaluation of the set of classifiers derived from $\mathcal{C}_{lr}$.

Apart from a little difference in the *FP Rate*, the Logistic Regression model leads to four same performing binary classifiers. Thus, any classifier could be chosen almost arbitrarily as well as the similarity functions, i.e., σ^*_{lr}, where $* \in \{1, 2, 3, 4\}$.

Globally, the best performing classifier is $\mathcal{C}^4_{dt}$. Indeed, it gains $\approx 0.50\%$ in terms of *F-measure* and it reduces the *FP Rate* of $\approx 8.44\%$ with respect to the best Naïve Bayesian classifiers, i.e., $\mathcal{C}^1_{nb}$ and $\mathcal{C}^3_{nb}$. Similarly, it gains $\approx 0.50\%$ in terms of *F-measure* and it reduces the *FP Rate* of $\approx 35.33\%$ with respect to the best Logistic Regression classifiers, i.e., $\mathcal{C}^4_{lr}$. This means that σ^4_{dt} might be considered the very best query similarity function for determining task relatedness.

Finally, in Section 4.8.2.3 we show the results obtained when the two best-performing task discovery methods exploit the three best-performing supervised similarity functions, i.e., σ^4_{dt}, σ^1_{nb} (or, equivalently, σ^3_{nb}), and σ^*_{lr}, where $* \in \{1, 2, 3, 4\}$.

4.7 Task Discovery Methods

In this Section, we address the *Task-oriented Session Discovery Problem* (TSDP) introduced in Section 4.4.1 by proposing and comparing several methods and techniques. The first family of task discovery methods we describe is globally referred to as *TimeSplitting-t*. Basically, *TimeSplitting-t* relies on the task relatedness measure presented in Section 4.6.1, i.e., σ_{time}, and it consists of splitting each session when the time between two query submissions is greater than a threshold $\bar{t}$. In the following, this set of methods will be considered as our *baseline*.

Therefore, we introduce *QueryClustering-m*, i.e., a family of *clustering-oriented* task discovery techniques based on several query clustering methods m.

The rationale of using query clustering leverages on the idea that queries ending up in the same cluster are also likely to be task-related, as long as the adopted clustering strategy is properly designed for dealing with query log data and for the problem we want to address. All the query clustering techniques presented in this work rely both on the *unsupervised* and *supervised* query similarity functions proposed in Section 4.6.

4.7.1 TimeSplitting-t

Time splitting techniques totally leverage on time-based approaches for measuring the task relatedness between two adjacent query pairs described in Section 4.6.1, i.e., σ_{time}. Roughly, they work as follows: for any given consecutive query pair (q_i, q_{i+1}) if $\sigma_{time}(q_i, q_{i+1})$ evaluates to 1 then q_i and q_{i+1} become part of the same task-oriented session, otherwise they are considered the *last* and the *first* query of two distinct task-oriented sessions, respectively.

According to Def. 4.4.1 of our theoretical model, time splitting techniques are used for detecting time-gap sessions, which represent a preliminary step before identifying task-oriented sessions. In particular, we use a time threshold $\bar{t} = t_{\phi} = 26$ minutes for identifying time-gap sessions of our query log (i.e., TS-26). This threshold was figured out from the testing data set as described in Section 4.3.2.

Also, the TSDP requires to find the best partitioning strategy over all the time-gap sessions available in the query log. A trivial partitioning strategy is the one that simply consider each time-gap session as a task-oriented session. In our case, this is equivalent to use only TS-26 for addressing the TSDP. However, other partitioning strategies might be devised by simply applying different time splitting techniques to each identified time-gap session. In this regard, there are several time thresholds that have been extensively proposed in literature [118, 55]. In this work, we use TS-5 and TS-15, i.e., 5 and 15 minutes thresholds, respectively.

The most prominent advantage of these techniques is their computational complexity, which is *linear* in the size of the input, i.e., number of queries. Indeed, since σ_{time} is only defined on *consecutive* query pairs and given that a sequence of n elements contains exactly $n - 1$ consecutive pairs, the overall time complexity of time-splitting techniques results to be linear in the number of input queries.

However, the main drawback of time splitting methods is that they are unable to properly deal with multi-tasking sessions, since identified sets of task-related queries are actually composed of temporarily ordered consecutive queries. Moreover, this makes splitting methods unsuitable since multi-tasking sessions represent a significative sample of the total available sessions, at least according to the analysis of our testing data set we provided in Section 4.5.

In Section 4.8.2, we compare the results provided by TS-5, TS-15, and TS-26. Also, we show that alone they are not suitable for identifying task-oriented sessions.

4.7.2 QueryClustering-m

We also propose a family of *clustering-oriented* task discovery methods, called *QueryClustering-m*. Basically, they differ from each other on the basis of the chosen clustering method *m*. To this extent, we start studying two algorithms derived from well-known clustering methods: QC-Means [85] and QC-Scan [44]. Moreover, we introduce two *graph-based* techniques: QC-wcc and its computationally-lighter variation QC-htc. All clustering algorithms have been applied to time-gap sessions, which in turn have been previously identified using TS-26 time splitting technique.

As for any other clustering problem, the effectiveness of these set of task discovery methods mostly depends on the *robustness* of the *similarity function* used by the algorithms, i.e., the measure of task-based query similarity as described in Section 4.6.

4.7.2.1 Algorithms

In the following, we describe four clustering algorithms, i.e., QC-Means, QC-Scan, QC-wcc, and QC-htc, which exploit the task-based query similarity functions proposed in Section 4.6. The first two are inspired to well-known clustering algorithms and they only make use of the task relatedness measures derived from the *unsupervised* approach, i.e., σ_1 and σ_2.

The last two algorithms follow a *graph-based* approach: QC-wcc identifies task-oriented sessions with the *connected components* of a query similarity graph, while QC-htc, which is a variation of QC-wcc, aims at reducing the computational cost of the clustering without impacting on the overall effectiveness. Both these two graph-based query clustering methods exploit the task relatedness scores either derived from *unsupervised* and *supervised* approaches.

Anyway, each algorithm takes as input the generic time-gap session $\phi_{i,k} = \langle q_{i,s_k}, q_{i,2}, \ldots, q_{i,e_k}\rangle$, where $|\phi_{i,k}| = |e_k - s_k| = n$, and it is associated with a specific partitioning strategy π_x. Thus, each algorithm provides as output $\pi_x(\phi_{i,k}) = \mathcal{T}_{i,k} = \{t^1_{i,k}, t^2_{i,k}, \ldots t^{|\mathcal{T}_{i,k}|}_{i,k}\}$, i.e., the set of task-oriented sessions of $\phi_{i,k}$ obtained by applying the partitioning strategy π_x, where $x \in \{$QC-Means, QC-Scan, QC-wcc, QC-htc$\}$.

QC-Means. It is a *centroid-based* algorithm and represents a variation of the well-known K-Means [85]. Here, we replace the usual K parameter, i.e. the number of clusters to be extracted, with a threshold ρ that defines the maximum radius of a centroid-based cluster. This allows us to better deal with the variance in length of user sessions as well as to avoid specifying the number of final clusters *apriori*.

At each step, a query $q_{i,j} \in \phi_{i,k}$ is either added to an existing cluster of queries $t^{\gamma}_{i,k}$ if its similarity with respect to the centroid query of $t^{\gamma}_{i,k}$ is at least $1-\rho$, otherwise $q_{i,j}$ itself becomes the centroid of a brand new cluster $t^{\gamma'}_{i,k}$. Query similarity score is computed using the *unsupervised* task-based query similarity functions described in Section 4.6.2.2, i.e., σ_1 and σ_2.

It is clear that the worst case happens when each cluster contains a single query, so that in the end the similarity between all query pairs has to be computed, making the the time complexity of QC-Means quadratic in the size of the input, i.e., $O(n^2)$.

QC-scan. It is the *density-based* DB-Scan algorithm [44], specifically tailored for extracting task-oriented sessions from Web search engine query logs. The rationale of evaluating also a variation of DB-Scan is that centroid-based approach may suffer the presence of noise in query logs. Moreover, the ϵ-neighborhood of each query is computed on the basis of the two *unsupervised* task-based query similarity functions σ_1 and σ_2.

Again, in the worst case QC-Scan requires to compute all query pairs similarity, making its time complexity quadratic in the size of the input, that is $O(n^2)$.

QC-wcc. It is a query clustering algorithm based on finding the *weighted connected components* of a graph. Given a time-gap session $\phi_{i,k}$, this algorithm builds a complete graph $G_{\phi_{i,k}} = (V, E, w)$, whose nodes V are the queries in $\phi_{i,k}$, i.e., $V = \{q_{i,j} \mid q_{i,j} \in \phi_{i,k}\}$, and whose E edges are weighted by the similarity of the corresponding nodes. The weighting function w is a query similarity function $w : E \longmapsto [0, 1]$ that can be easily computed in terms of the task-based query similarity functions proposed in Section 4.6. Thus, the graph $G_{\phi_{i,k}}$ describes the task-based similarity between any pair of queries in the given tim-gap based session.

The rationale of QC-wcc is to drop *weak edges*, i.e., with low similarity, since the corresponding queries are not related, and to build clusters on the basis of the *strong edges*, i.e., with high similarity, which identify the related query pairs. The algorithm performs two steps. In the first step, given the graph $G_{\phi_{i,k}}$ all the edges $e \in E$ whose weight is smaller than a given threshold, that is $w(e) < \eta$, are removed, thus obtaining a pruned graph $G'_{\phi_{i,k}}$. In the second step, the connected components of $G'_{\phi_{i,k}}$ are extracted. Such connected components identify the set of clusters of related queries $\mathcal{T}_{i,k} = \{t^1_{i,k}, t^2_{i,k}, \ldots t^{|\mathcal{T}_{i,k}|}_{i,k}\}$, which are returned by the algorithm.

Indeed, assuming a robust similarity function, the QC-wcc algorithm is able to handle the multi-tasking nature of users sessions. Groups of related queries are isolated by the pruning of weak edges and links with large similarity identify the generalization/specialization steps of the users, as well as restarts from a previous query when the current query chain is found to be unsuccessful.

The computational complexity of QC-wcc is dominated by the construction of the graph $G_{\phi_{i,k}}$. Indeed, the similarity between any pair of edges must be computed, resulting in a number of similarity computations quadratic in the number of nodes, i.e., $O(n^2)$, where $n = |V| = |\phi_{i,k}|$ is the number of queries in the session $\phi_{i,k}$. Instead, all the connected components are easily computed in linear time with respect to the number of nodes in $G_{\phi_{i,k}}$, i.e., $O(n)$, by using either *breadth-first* or *depth-first* search.

QC-HTC. It is a variation of the previous QC-WCC algorithm, which is based on *head-tail* components and does not need to compute the full similarity graph. Since queries are submitted one after the other by the user, the QC-HTC algorithms takes advantage of this sequentiality to reduce the number of similarity computations needed by QC-WCC. The algorithm works in two phases as follows.

The first step aims at creating an approximate fine-grained clustering of the given time-gap session $\phi_{i,k}$. Every single Web-mediated task generates a sequence of queries. Due to the multi-tasking behavior of users, multiple Web-mediated tasks are carried on at the same time, therefore the query log records such overlapping tasks and the corresponding query sequences. As a consequence, each Web-mediated task is observed as a set of *fragments*, i.e. smaller sets of consecutive queries, and fragments of different tasks are interleaved in the query log because of multi-tasking.

The algorithm exploits the sequentiality of user queries and tries to detect the above fragments, by partitioning the given time-gap session $\phi_{i,k}$ into *sequential clusters*, where a sequential cluster, denoted with $\widetilde{t}^{\gamma}_{i,k}$, must contain only queries that occur in a row within the query log, such that each query is similar enough to the chronologically following one.

The second step of the algorithm merges together those fragments when they are related, trying to overcome the interleaving of different tasks. Here, we introduce another assumption that reduces the computational cost of the algorithm. We assume that a cluster of queries can be well described by just the chronologically first and last of its queries, respectively denoted with $head(\widetilde{t}^{\gamma}_{i,k})$ and $tail(\widetilde{t}^{\gamma}_{i,k})$. Therefore, the similarity sim between two clusters $\widetilde{t}^{\gamma_1}_{i,k}$, $\widetilde{t}^{\gamma_2}_{i,k}$ is computed as:

$$sim(\widetilde{t}^{\gamma_1}_{i,k}, \widetilde{t}^{\gamma_2}_{i,k}) = \min_{\substack{q \in \{head(\widetilde{t}^{\gamma_1}_{i,k}), tail(\widetilde{t}^{\gamma_1}_{i,k})\} \\ q' \in \{head(\widetilde{t}^{\gamma_2}_{i,k}), tail(\widetilde{t}^{\gamma_2}_{i,k})\}}} w(e(q, q')).$$

where w weights the edge $e(q, q')$ linking the queries q and q' with respect to their task-based similarity, analogously to QC-WCC.

The final clustering is produced as follows. The first cluster $t^1_{i,k}$ is initialized with the oldest sequential cluster $\widetilde{t}^1_{i,k}$, which is removed from the set of sequential clusters. Then, $t^1_{i,k}$ is compared with any other chronologically-ordered sequential cluster $\widetilde{t}^{\gamma}_{i,k}$ by computing the similarity as above. Given a threshold η, if $sim(t^1_{i,k}, \widetilde{t}^{\gamma}_{i,k}) \geq \eta$, then $\widetilde{t}^{\gamma}_{i,k}$ is merged into $t^1_{i,k}$, the *head* and *tail* queries of $t^1_{i,k}$ are updated consequently and $\widetilde{t}^{\gamma}_{i,k}$ is removed from the set of sequential clusters. The algorithm continues comparing the new cluster $t^1_{i,k}$ with the remaining sequential clusters. When all the sequential clusters have been considered, the oldest sequential cluster available is used to build a new cluster $t^2_{i,k}$, and so on and so forth.

The algorithm iterates this procedure until no more sequential clusters are left.

In the worst case, the complexity of QC-HTC is still quadratic in the number of queries in $\phi_{i,k}$. However, there are frequent cases in which the complexity is much smaller. We have seen that 52.8% of the time-gap sessions contain one task only. In this case, it is very likely that this task is found after the first step of the algorithm, if each query is sufficiently similar to the next one. Since the first step of the algorithm only requires to compute the similarity between timely-adjacent query pairs, its computational cost is linear in the number of nodes, i.e., $O(n)$, where $n = |V| = |\phi_{i,k}|$ is the number of queries in the session $\phi_{i,k}$. Nevertheless, for multi-tasking sessions, the time complexity of the second step is quadratic in the number of sequential clusters extracted, i.e., $O(m^2)$ if m is the number of sequential clusters. Also in this case, the cost reduction may be significant. Suppose that the number of sequential clusters is $m = \beta|\phi_{i,k}| = \beta n$, with $0 < \beta \leq 1$, then the time complexity of the algorithm is $O(\beta^2 n^2)$. Let the number of sequential clusters be the half of the number of queries, i.e., $m = n/2$ so that $\beta = 1/2$, then the overall algorithm complexity is $O(n^2/4)$, namely four times cheaper than QC-wcc. Still, this is an upper bound of the computational cost, since the QC-htc algorithm does not compute the pair-wise similarities among sequential clusters in advance.

It is worth noting that we first instantiate the query similarity function w of both QC-wcc and QC-htc with the *unsupervised* task-based query similarity functions, i.e., $w = \sigma_1$ and $w = \sigma_2$. Moreover, since experimental evaluation showed QC-wcc and QC-htc outperformed both other clustering-oriented methods, i.e., QC-Means and QC-Scan, as well as state-of-the-art approaches, i.e., QFG introduced by [28], we also instantiate the query similarity function w with the *supervised* task-based query similarity functions, i.e., $w = \sigma_{dt}^4$, $w = \sigma_{nb}^1$ (or, equivalently, $w = \sigma_{nb}^3$), and $w = \sigma_{lr}^*$, where $* \in \{1, 2, 3, 4\}$.

All the obtained results are fully described in Section 4.8.2.

4.8 Experiments

In this Section, we analyze and compare the results obtained with all the task discovery methods we described in Section 4.7 for approaching the TSDP. Moreover, we compare our results with the ones provided by two other task discovery methods: *(i)* the simple time splitting technique TS-26, which is considered as the *baseline* solution and *(ii)* the session extraction method based on the *Query Flow Graph* model proposed by Boldi *et al.* [28], which may be considered as the state-of-the-art approach.

4.8.1 Validity Measures

In order to evaluate and compare all the methods we mentioned, we need to measure the degree of correspondence between manually extracted tasks of the *ground-truth* (Section 4.5) and tasks produced by our algorithms. To this end, we use both *classification-* and *similarity-oriented* measures [131]. In the following, we thus refer to "*predicted classes*" as the task-oriented sessions detected by a specific algorithm, whereas the "*true classes*" just correspond to the task-oriented sessions of the ground-truth.

Classification-oriented approaches measure the degree to which predicted classes correspond to true classes and *F-measure* is one of the most popular score in this category. It combines both *Precision* and *Recall*: the former measures the fraction of a task that consists of objects of a specified class, while the latter measures the extent to which a task contains all objects of a specified class. Thus, globally F-measure evaluates the extent to which a task contains *only* objects of a particular class and *all* objects of that class. Given $p(i, j)$, $r(i, j)$ the precision and recall of task i with respect to class j, the F-measure corresponds to the weighted harmonic mean of $p(i, j)$ and $r(i, j)$:

$$F(i, j) = \frac{2 \times p(i, j) \times r(i, j)}{p(i, j) + r(i, j)}.$$

In order to compute a global F-measure, for each generic time-gap session $\phi_{i,k}$ we first consider the set of its associated *predicted tasks* $\pi(\phi_{i,k}) = \mathcal{T}_{i,k} = \{t^1_{i,k}, t^2_{i,k}, \ldots, t^{|\mathcal{T}_{i,k}|}_{i,k}\}$, i.e., the set of all the task-oriented sessions extracted from $\phi_{i,k}$, by using the partitioning strategy π. Analogously, we take into account the set of *true tasks* $\Theta_{i,k} = \{\theta^1_{i,k}, \theta^2_{i,k}, \ldots, \theta^{|\Theta_{i,k}|}_{i,k}\}$, i.e., the set of task-oriented sessions of the ground-truth.

Moreover, in order for those two sets $\mathcal{T}_{i,k}$ and $\Theta_{i,k}$ to have the same size, i.e., $|\mathcal{T}_{i,k}| = |\Theta_{i,k}|$, we pad both of them with all the "unclassified" queries, namely with all the queries that appear in the original session $\phi_{i,k}$ but that have been somehow discarded either during the automatic and/or the manual clustering. Somehow, this is equivalent to consider discarded queries as singleton clusters, i.e., single task composed of only one query.

Thus, for each predicted task $t^j_{i,k}$ we compute the *maximum* F-measure, i.e., $F_{\max}(t^j_{i,k})$, with respect to the true tasks as follows:

$$F_{\max}(t^j_{i,k}) = \underset{x}{\operatorname{argmax}}\ F(t^j_{i,k}, \theta^x_{i,k}).$$

Globally, the F-measure is averaged on the set of all predicted tasks $\mathcal{T} = \bigcup_{i,k} \mathcal{T}_{i,k}$ with respect to the set of all true tasks $\Theta = \bigcup_{i,k} \Theta_{i,k}$ as follows:

$$F(\mathcal{T}, \Theta) = \frac{w_j \cdot F_{\max}(t^j_{i,k})}{\sum_{j=1}^{|\mathcal{T}|} w_j},$$

where $w_j = |t^j_{i,k}|$.

Besides, similarity-oriented measures consider pairs of objects instead of single objects. Again, let $\phi_{i,k}$ be the generic time-gap session such that $|\phi_{i,k}| > 1$, $\mathcal{T}_{i,k}$ the set of its predicted tasks, and $\Theta_{i,k}$ the set of its true tasks (both padded with discarded queries as described above). Thus, for each $\phi_{i,k}$ we retrieve:

– f_{00} = number of query pairs that are in different true tasks and in different predicted tasks (*true negative*);

– f_{01} = number of query pairs that are in different true tasks but in the same predicted task (*false positive*);

– f_{10} = number of query pairs that are in the same true task but in different predicted tasks (*false negative*);

– f_{11} = number of query pairs that are in the same true task and in the same predicted tasks (*true positive*).

Thus, two different measures are defined:

– *Rand index* $R(\mathcal{T}_{i,k}) = \frac{f_{00}+f_{11}}{f_{00}+f_{01}+f_{10}+f_{11}}$;

– *Jaccard index* $J(\mathcal{T}_{i,k}) = \frac{f_{11}}{f_{01}+f_{10}+f_{11}}$.

A global value of both Rand and Jaccard index, i.e., R and J respectively, might be computed as follows:

$$R = \frac{w_j \cdot R(\mathcal{T}_{i,k})}{\sum_{j=1}^{|\mathcal{T}|} w_j},$$

$$J = \frac{w_j \cdot J(\mathcal{T}_{i,k})}{\sum_{j=1}^{|\mathcal{T}|} w_j},$$

where $w_j = |\phi_{i,k}|$.

As specified before, when computing both Rand and Jaccard index we do not consider time-gap sessions containing only one singleton task, i.e., time-gap sessions containing only one single-query cluster. Anyway, we still took into account time-gap sessions that are composed of a single task with more than one query.

4.8.2 Evaluation

In the following, we show the results we obtain using our two classes of task discovery methods, i.e., *Time-Splitting-t* and *QueryClustering-m*, respectively, and we compare them with a state-of-the-art approach based on the *Query Flow Graph* model.

4.8.2.1 TimeSplitting-t

This set of task discovery methods are exclusively based on the task-based query similarity function described in Section 4.6.1, i.e., σ_{time}. In particular, here we compare three different time splitting techniques: TS-5, TS-15, and TS-26, which use 5, 15, and 26 minutes thresholds $\bar{t}$, respectively.

Table 4.5 shows the results we obtain using those techniques on the ground-truth. The best result in terms of F-measure is found considering the whole time-gap sessions identified with TS-26, without additionally splitting them into shorter time-gap sessions. Hence, we consider TS-26 as the *baseline* approach for addressing the TSDP. Roughly, according to Def. 4.4.1 and Def. 4.4.2, this is equivalent to identify task-oriented sessions with time-gap sessions.

	F-measure	*Rand*	*Jaccard*
TS-5	0.28	**0.75**	0.03
TS-15	0.28	0.71	0.08
TS-26	**0.65**	0.34	**0.34**

Table 4.5: TS-5, TS-15, and TS-26.

4.8.2.2 Query Flow Graph

In order to better evaluate our proposed approaches we decide to compare them to the *Query Flow Graph* (QFG) presented by [28].

QFG is constructed over a training segment of the AOL *top*-500 user sessions. This method uses *chaining probabilities* measured by means of a machine learning method. First, we extract some features from the training Web search engine log and we store them into a compressed graph representation. In particular, we extract 25 different features (i.e., time-related, session, and textual features) for each pair of queries (q, q') that are consecutive in at least one session of the query log.

The validity of QFG is tested on the ground-truth and the results we obtain are showed in Table 4.6. We find the best values using a threshold $\eta = 0.7$. In fact, results do not improve using a greater threshold value.

Moreover, QFG significantly improves the baseline TS-26. In particular, F-measure gains $\approx 16\%$, while Rand and Jaccard gains $\approx 52\%$ and $\approx 15\%$, respectively.

4.8.2.3 QueryClustering-m

We now evaluate all the *clustering-oriented* task discovery methods described in Section 4.7.2.1.

First, we present the results we obtain using the task-based query similarity functions derived from the *unsupervised* approach described in Section 4.6.2 i.e.,

	η	*F-measure*	*Rand*	*Jaccard*
QFG	0.1	0.68	0.47	0.36
	0.2	0.68	0.49	0.36
	0.3	0.69	0.51	0.37
	0.4	0.70	0.55	0.38
	0.5	0.71	0.59	0.38
	0.6	0.74	0.65	0.39
	0.7	**0.77**	**0.71**	**0.40**
	0.8	0.77	0.71	0.40
	0.9	0.77	0.71	0.40

Table 4.6: QFG varying the threshold η.

σ_1 and σ_2. Therefore, we also show the outcomes of two of these task discovery methods, i.e., QC-wcc and QC-htc, when exploiting the *supervised* task-based query similarity functions proposed in Section 4.6.3, i.e., σ^4_{dt}, σ^1_{nb} (or, equivalently, σ^3_{nb}), and σ^*_{lr}, where $* \in \{1, 2, 3, 4\}$.

Unsupervised Task-based Similarity. We start evaluating QC-Means clustering algorithm using both the unsupervised task-based query similarity functions σ_1 and σ_2. We empirically set the radius ρ of this centroid-based algorithm to 0.4 for both similarity functions, i.e., two queries could be part of the same cluster if and only if their similarity is equal to or greater than 0.6. The overall results of this method are showed in Table 4.7.

Concerning σ_1, the best results are obtained by using only the content-based similarity, i.e., with $\alpha = 1$. However, the very best results for QC-Means are found when using σ_2. Here, we significantly improve the baseline TS-26 in terms of F-measure ($\approx 10\%$) and Rand ($\approx 54\%$), while we lose $\approx 21\%$ in terms of Jaccard. Moreover, if we compare the best QC-Means with the best QFG we can notice that QC-Means loses $\approx 6\%$ for F-measure, $\approx 33\%$ for Jaccard but it gains $\approx 4\%$ in terms of Rand.

Then, we analyze QC-Scan algorithm, again using both the similarity functions σ_1 and σ_2. We use several combinations of the two density-based parameters, i.e., *minPts* and *eps*, and we find the best results with *minPts* $= 2$ and *eps* $= 0.4$.

Table 4.8 highlights that QC-Scan provides globally better results than QC-Means for both σ_1 and σ_2. Still, for σ_1 the best results are obtained by using only content-based similarity, i.e., with $\alpha = 1$. However, our proposed conditional function σ_2 reveals a significative improvement with respect to all measures.

QC-MEANS $_{\sigma_1}$				
		F-measure	*Rand*	*Jaccard*
α	$(1-\alpha)$			
1	0	0.71	0.73	0.26
0.5	0.5	0.68	0.70	0.14
0	1	0.68	0.70	0.13

QC-MEANS $_{\sigma_2}$				
		F-measure	*Rand*	*Jaccard*
t	**b**			
0.5	4	**0.72**	**0.74**	**0.27**

Table 4.7: QC-MEANS using *unsupervised* task-based query similarity functions σ_1 and σ_2.

Finally, it is worth noticing that QC-SCAN behaves exactly as QFG, except for the Jaccard where QC-SCAN loses $\approx 53\%$.

QC-SCAN $_{\sigma_1}$				
		F-measure	*Rand*	*Jaccard*
α	$(1-\alpha)$			
1	0	0.77	0.71	0.17
0.5	0.5	0.74	0.68	0.06
0	1	0.75	0.68	0.07

QC-SCAN $_{\sigma_2}$				
		F-measure	*Rand*	*Jaccard*
t	**b**			
0.5	4	**0.77**	**0.71**	**0.19**

Table 4.8: QC-SCAN using *unsupervised* task-based query similarity functions σ_1 and σ_2.

The third algorithm we consider is QC-WCC. Table 4.9 shows the results we find using this algorithm either with σ_1 and σ_2 and by varying the pruning threshold η. In particular, concerning σ_1 we only consider the best convex combination when $\alpha = 0.5$.

The best results with σ_1 are obtained when $\eta = 0.2$, while even better results are found with σ_2 when $\eta = 0.3$. In this last case, the overall evaluation is significantly higher than the baseline TS-26 but also than the state-of-the-art approach QFG. Concerning TS-26, the best QC-WCC gains $\approx 20\%$, $\approx 56\%$, and $\approx 23\%$ in terms of F-measure, Rand, and Jaccard, respectively. Moreover, QC-WCC improves also the results of QFG, gaining $\approx 5\%$ for F-measure, $\approx 9\%$ for Rand, and $\approx 10\%$ for Jaccard.

QC-HTC is the last algorithm we introduce and represents one of the novel contribution of our work. The results we get using this approach with both similarity functions σ_1 and σ_2 and by varying the pruning threshold η are showed in Table 4.10. As for QC-WCC, regarding σ_1 we only consider the best convex combination when $\alpha = 0.5$. Again, the best results with σ_1 are obtained when $\eta = 0.2$, while the global best results are found with σ_2 when $\eta = 0.3$. As the table shows, the overall results

QC-WCC $_{\sigma_1}$ ($\alpha = 0.5$)			
η	*F-measure*	*Rand*	*Jaccard*
0.1	0.78	0.71	0.42
0.2	**0.81**	**0.78**	**0.43**
0.3	0.79	0.77	0.37
0.4	0.75	0.73	0.27
0.5	0.72	0.71	0.20
0.6	0.75	0.70	0.14
0.7	0.74	0.69	0.11
0.8	0.74	0.68	0.07
0.9	0.72	0.67	0.04

QC-WCC $_{\sigma_2}$ ($\mathbf{t} = 0.5$, $\mathbf{b} = 4$)			
η	*F-measure*	*Rand*	*Jaccard*
0.1	0.67	0.45	0.33
0.2	0.78	0.71	0.42
0.3	**0.81**	**0.78**	**0.44**
0.4	0.81	0.78	0.41
0.5	0.80	0.77	0.37
0.6	0.78	0.75	0.32
0.7	0.75	0.73	0.23
0.8	0.71	0.70	0.15
0.9	0.69	0.68	0.08

Table 4.9: QC-WCC using *unsupervised* task-based query similarity functions σ_1 and σ_2.

are very close to the ones obtained with QC-WCC. In particular, QC-HTC improves TS-26 by gaining $\approx$ 19%, $\approx$ 56%, and $\approx$ 21% in terms of F-measure, Rand, and Jaccard, respectively. Therefore, QC-HTC provides better results than QFG and gains $\approx$ 4% for F-measure, $\approx$ 9% for Rand, and $\approx$ 8% for Jaccard.

Supervised Task-based Similarity. Another valuable contribution of this work concerns the *supervised* approach for computing the task-based query similarity functions, as described in Section 4.6.3.

Roughly, a set of query similarity functions are *learned* by training a family of classifiers on a set of both *internal* and *external* query log features. This contrasts with the *unsupervised* approach, where query similarity functions are *directly* derived from the query log data without any supervised learning step.

Thus, here we also evaluate how this new approach for measuring the task relatedness between query pairs impacts on the effectiveness of the two best-performing clustering-oriented task discovery methods, i.e., QC-WCC and QC-HTC.

As a reminder, supervised task-based similarity functions affect the way in which we build the similarity graph either in QC-WCC and in QC-HTC. Indeed, an edge between a query pair (q_i, q_j) is created whenever the considered classifier assigns the class attribute "`same_task = yes`" to (q_i, q_j). Moreover, the weight assigned to each created edge corresponds to the prediction accuracy value provided by the classifier.

According to the performance evaluation of the classifiers we propose above in Section 4.6.3.3, we run both the QC-WCC and QC-HTC algorithms using the three best task-based query similarity functions: σ^4_{dt}, σ^1_{nb} (or, equivalently, σ^3_{nb}), and σ^*_{lr},

QC-HTC $_{\sigma_1}$ ($\alpha = 0.5$)			
η	*F-measure*	*Rand*	*Jaccard*
0.1	0.78	0.72	0.41
0.2	**0.80**	**0.78**	**0.41**
0.3	0.78	0.76	0.35
0.4	0.75	0.73	0.25
0.5	0.73	0.70	0.18
0.6	0.75	0.70	0.13
0.7	0.74	0.69	0.10
0.8	0.74	0.68	0.06
0.9	0.72	0.67	0.03

QC-HTC $_{\sigma_2}$ ($\mathbf{t} = 0.5$, $\mathbf{b} = 4$)			
η	*F-measure*	*Rand*	*Jaccard*
0.1	0.68	0.56	0.32
0.2	0.78	0.73	0.41
0.3	**0.80**	**0.78**	**0.43**
0.4	0.80	0.77	0.38
0.5	0.78	0.76	0.34
0.6	0.77	0.74	0.30
0.7	0.74	0.72	0.21
0.8	0.71	0.70	0.14
0.9	0.68	0.67	0.07

Table 4.10: QC-HTC using *unsupervised* task-based query similarity functions σ_1 and σ_2.

where $* \in \{1, 2, 3, 4\}$. These similarity scores are used for computing the weighting edge similarity function w of our graph-based algorithms.

Table 4.11 shows and compares the results obtained both with QC-WCC and QC-HTC using the supervised query similarity function σ^4_{dt}. Concerning QC-WCC the best results are provided when $\eta = 0.8$, while QC-HTC gets its best outcomes when $\eta = 0.7$.

Similarly, Table 4.12 shows and compares the results obtained both with QC-WCC and QC-HTC using the supervised query similarity function σ^1_{nb} (or, equivalently, σ^3_{nb}). In both cases, best F-measure and Rand values are obtained when $\eta = 1.0$ whereas best Jaccard results are obtained when $0.0 \leq \eta \leq 0.7$. However, all the validity measures lose significantly with respect to QC-WCC and QC-HTC using σ^4_{dt}. Moreover, differently from using σ^4_{dt}, in which best results in terms of all the validity measures seem to focus around a unique value of the threshold η, i.e., $\eta = 0.8$ and $\eta = 0.7$, respectively, here it appears there is not a so strong relationship between the overall best results and η.

Therefore, Table 4.13 shows and compares the results obtained both with QC-WCC and QC-HTC using the supervised query similarity function σ^*_{lr}. Either QC-WCC and QC-HTC reach their best outcomes when the threshold $\eta = 0.7$. However, even in this case all the validity measures lose significantly with respect to QC-WCC and QC-HTC using σ^4_{dt}. Anyway, as for σ^4_{dt} here is a clear relationship between the best validity measures and the value of η.

Globally, the very best results for both QC-WCC and QC-HTC when using a supervised similarity function are provided by exploiting σ^4_{dt}. In both cases, this leads to similar results obtained when QC-WCC and QC-HTC use the unsupervised

QC-WCC using σ^4_{dt}			
η	*F-measure*	*Rand*	*Jaccard*
0.0	0.76	0.69	0.43
0.1	0.76	0.69	0.43
0.2	0.76	0.69	0.43
0.3	0.76	0.69	0.43
0.4	0.76	0.69	0.43
0.5	0.76	0.69	0.43
0.6	0.78	0.77	0.46
0.7	0.79	0.78	0.45
0.8	**0.80**	**0.79**	**0.45**
0.9	0.80	0.79	0.42
1.0	0.71	0.70	0.13

QC-HTC using σ^4_{dt}			
η	*F-measure*	*Rand*	*Jaccard*
0.0	0.76	0.73	0.42
0.1	0.76	0.73	0.42
0.2	0.76	0.73	0.42
0.3	0.76	0.73	0.42
0.4	0.76	0.73	0.42
0.5	0.76	0.73	0.42
0.6	0.78	0.79	0.44
0.7	**0.79**	**0.79**	**0.43**
0.8	0.79	0.79	0.42
0.9	0.78	0.78	0.38
1.0	0.68	0.69	0.10

Table 4.11: QC-WCC vs. QC-HTC using *supervised* task-based query similarity function σ^4_{dt}.

QC-WCC using σ^1_{nb} or σ^3_{nb}			
η	*F-measure*	*Rand*	*Jaccard*
0.0	0.65	0.36	**0.33**
0.1	0.65	0.36	**0.33**
0.2	0.65	0.36	**0.33**
0.3	0.65	0.36	**0.33**
0.4	0.65	0.36	**0.33**
0.5	0.65	0.36	**0.33**
0.6	0.65	0.36	**0.33**
0.7	0.65	0.37	**0.33**
0.8	0.64	0.40	0.32
0.9	0.65	0.48	0.30
1.0	**0.75**	**0.73**	0.24

QC-HTC using σ^1_{nb} or σ^3_{nb}			
η	*F-measure*	*Rand*	*Jaccard*
0.0	0.65	0.38	**0.33**
0.1	0.65	0.38	**0.33**
0.2	0.65	0.38	**0.33**
0.3	0.65	0.38	**0.33**
0.4	0.65	0.38	**0.33**
0.5	0.65	0.38	**0.33**
0.6	0.65	0.38	**0.33**
0.7	0.65	0.39	**0.33**
0.8	0.64	0.42	0.31
0.9	0.65	0.50	0.30
1.0	**0.75**	**0.72**	0.22

Table 4.12: QC-WCC vs. QC-HTC using *supervised* task-based query similarity function σ^1_{nb} or σ^3_{nb}.

QC-WCC using σ_{lr}^*			
η	*F-measure*	*Rand*	*Jaccard*
0.0	0.65	0.50	0.30
0.1	0.65	0.50	0.30
0.2	0.65	0.50	0.30
0.3	0.65	0.50	0.30
0.4	0.65	0.50	0.30
0.5	0.65	0.50	0.30
0.6	0.70	0.64	0.30
0.7	**0.77**	**0.75**	**0.31**
0.8	0.76	0.73	0.24
0.9	0.74	0.70	0.15
1.0	0.73	0.66	0.00

QC-HTC using σ_{lr}^*			
η	*F-measure*	*Rand*	*Jaccard*
0.0	0.65	0.51	0.30
0.1	0.65	0.51	0.30
0.2	0.65	0.51	0.30
0.3	0.65	0.51	0.30
0.4	0.65	0.51	0.30
0.5	0.65	0.51	0.30
0.6	0.68	0.65	0.28
0.7	**0.76**	**0.75**	**0.30**
0.8	0.75	0.73	0.24
0.9	0.74	0.70	0.14
1.0	0.73	0.66	0.00

Table 4.13: QC-WCC vs. QC-HTC using *supervised* task-based query similarity function σ_{lr}^* ($* \in \{1, 2, 3, 4\}$).

similarity function σ_2. Table 4.14, which gives an overview and compares the best results found with each examined approach, highlights these similar behaviors.

On a side note, all the tables show that the clustering validity measures, i.e., F-measure, Rand, and Jaccard, are the same until the similarity threshold η reaches the value 0.5. Since this threshold gives a lower-bound on the weights of the edges to be considered during the clustering step, this means that there is no prediction whose accuracy value is less than 0.5 for the class attribute "`same_task = yes`".

Finally, Table 4.15 clearly points out the benefit of exploiting collaborative knowledge like Wikipedia. QC-HTC was able to capture and group together two queries that are completely different from a content-based perspective, but that are strictly semantically-related, using the similarity function σ_2. Indeed, "Cancun" is one of the region where the "Hurricane Wilma" impacted during the 2005 season (note the cross reference inside the corresponding Wikipedia article[5]). Moreover, "Los Cabos" and "Cancun" are both in Mexico despite they are far away from each other. It might be the case, of course with no absolute certainty, the user was looking for the relative position of Los Cabos from Cancun just to understand if Los Cabos was struck by the hurricane as well.

[5]`http://en.wikipedia.org/wiki/Cancun`

		F-measure	*Rand*	*Jaccard*
TS-26 (*baseline*)		0.65	0.34	0.34
QFG $_{best}$ (*state of the art*)		0.77	0.71	0.40
unsupervised similarity σ_2	QC-Means $_{best}$	0.72	0.74	0.27
	QC-Scan $_{best}$	0.77	0.71	0.19
	QC-wcc $_{best}$	**0.81**	0.78	0.44
	QC-htc $_{best}$	0.80	0.78	0.43
supervised similarity σ_{dt}^4	QC-wcc $_{best}$	0.80	**0.79**	**0.45**
	QC-htc $_{best}$	0.79	**0.79**	0.43

Table 4.14: Best results obtained with each task discovery method using both *unsupervised* and *supervised* similarity.

QC-htc $_{\sigma_1}$ ($\alpha = 1$)		QC-htc $_{\sigma_2}$ (0.5, 4)	
Query ID	Query String	Query ID	Query String
		63	los cabos
		64	cancun
65	hurricane wilma	65	hurricane wilma
68	hurricane wilma	68	hurricane wilma

Table 4.15: The impact of Wikipedia: σ_1 vs. σ_2.

4.9 Summary

In this Chapter, we have addressed some research challenges for making next-generation Web search engines able to satisfy new and complex user needs. In particular, we claimed that people increasingly formulate queries to Web search engines for being helped in accomplishing their daily *tasks*, instead of simply retrieving Web pages. To this extent, Web search engines' *query logs* represent the most suitable source of information for verifying such belief because they record a huge amount of user search activities by means of issued queries, the timestamp at which a certain query has been submitted, the clicked results in response to a query, the rank of a clicked result, etc.

Here, we have discussed how to discover user search sessions from a very large, long-term Web search engine query log, whose final aim is to accomplish a task, i.e., *task-oriented sessions*. Therefore, we have formally defined the *Task-oriented Session Discovery Problem* (TSDP) as the problem of best partitioning a stream of user queries into several subsets of queries which are all related to the same search task. Roughly, this has dealt with two issues: *(i)* it has required a robust measure for evaluating the *task relatedness* between any two queries, i.e., *task-based query similarity* and *(ii)* it has needed an effective method for actually discovering task-oriented sessions using the above measure of task relatedness. Concerning *(i)*, we have proposed and compared both *unsupervised* and *supervised* approaches for devising several task-based query similarity functions. Moreover, we have tackled *(ii)* by introducing a set of *clustering-oriented* approaches that exploit the above task-based query similarity functions, i.e., *query clustering* methods specifically tuned for actually discovering task-oriented sessions.

All the proposed solutions have been evaluated on a manually built *ground-truth*, namely a task-oriented partitioning of the queries stored in a real Web search engine log performed by human annotators. In particular, two of the clustering-oriented methods we have proposed, i.e., QC-WCC and QC-HTC, have shown to perform better than state-of-the-art approaches.

The "take-away message" of this work is that to effectively and efficiently extract tasks from long term sessions contained in query log one has to adopt the following method that in our experiments produced the best results. For each user, firstly split the sequence of queries into subsequences consisting of queries whose gap, in terms of submission time, is less than 26 minutes. Then, for queries in each session we build a graph consisting of queries connected by an edge if and only if the *similarity* between these two queries is above a threshold $\eta = 0.3$. The similarity is a combination of *content-based* and *semantic-based* metrics. The first is obtained by combining two distinct metrics, i.e., the Jaccard's score on query tri-grams and the classical edit-distance between the two queries. These metrics are put together through a convex combination with coefficient equal to 0.5. The semantic-based metric is obtained by considering what we called the "*wiki-based*" similarity and consists in evaluating the cosine distance between the two vectors obtained by "*wikifying*" the two queries. Instead of using a convex combination of the queries we resorted in using the following strategy. If the content-based similarity is below a threshold **t** = 0.5 then we resort to use the semantic-based metric and we take the maximum of these two similarities as the weight of the edge in the graph. Finally, tasks are represented by the connected components of the resulting graph.

5 Search Task Recommendation

"I find that a great part of the information I have was acquired by looking up something and finding something else on the way."

Franklin Pierce Adams
Reader's Digest (1960)

In this Chapter, we address the second research challenge already sketched in Section 3.6.2, namely the design of a novel recommendation mechanism that goes beyond traditional query suggestion schemes. In particular, the proposed work leverages on the results described in Chapter 4 for building up a model of user search tasks, which in turn is exploited for generating *task recommendations*. The contents of this Chapter are based on a research paper, i.e., "*Beyond Query Suggestion: Recommending Tasks to Search Engine Users*" [84], which has been submitted to the 5th ACM International Conference on Web Search and Data Mining (WSDM 2012).

Today, most popular *Web search engines* provide *query suggestions* as a valuable mechanism for supporting their users. Typically, this aims to help people in better specifying their actual search goals, thus to quicker satisfy their information needs. So far, recommendations generated for a certain query are strictly related to the query itself, namely to the same *search task* that the query is likely to represent. However, this often unfits the search behaviors of users, which instead are used to issue streams of queries possibly related to distinct *interleaved* search tasks. Moreover, the "composition" of each single search task (e.g., "*booking a flight*", "*reserving a hotel room*", "*renting a car*", etc.) usually subsumes a *complex* task, namely a *mission*, that the user aims to accomplish throughout the Web search engine (e.g., "*planning a travel*"). Therefore, we claim that recommendation mechanisms should integrate the common "*query-by-query*" paradigm with a novel "*task-by-task*" suggestion scheme. Indeed, once detected the "small task" behind a sequence of few user queries, suggesting queries or Web pages that could be possibly part of another yet related task might realize a better user search experience. For example, a user searching for a suitable hotel in New York might be recommended with information about Broadway shows that are planned to be played during the same dates, MTA subway or Broadway shows, conjecturing that she is willing to organize her vacation, i.e., a complex task. In this Chapter, we propose an innovative recommender application, namely a *task recommender system*, which generates *task suggestions* on the basis of a model of user search tasks learned from the historical activities recorded in query logs. Experimental results show that our solution is able to suggest tasks that are different from the ones that users are currently performing, but related to those which users are going to look for in the next future. Moreover, even if generated recommendations are not part of future and actually requested user tasks, we show that they could anyway represent useful "*surprising*" hints.

5.1 Introduction

People heavily put their trust in *Web search engines* for satisfying everyday information needs. A key factor for the popularity of today's Web search engines is their user-friendly interfaces [13], which allow users to formulate information needs by means of simple lists of *keywords*. However, keyword-based queries are not always an effective descriptor of actual user search intents. A first problem occurs because the ability of each single user to phrase effective queries to a Web search engine highly depends on her subjective skills. It might be the case that two users, looking for exactly the same information need, formulate two completely different queries, on the basis of their familiarity with the specific terminology of the knowledge domain.

Often queries are refined by adding or modifying keywords until the user satisfies her search goal. In order to support users in the search process, Web search engines have started to add mechanisms and features aimed to help people to quickly satisfy their needs. In this regard, *query suggestion* plays a significative role by recommending lists of alternative yet related queries that are likely to better specify user search tasks. Indeed, query suggestion techniques focus specifically on driving users towards their "*immediate*" goals, i.e., suggesting queries that could shorten the task they are trying to perform. These techniques, in fact, neglects that people issue streams of queries representing *interleaved* search tasks, which often are part of a more *complex* task that they aim to accomplish throughout the Web search engine, as we showed in Section 4.5.

In order to explain this concept, consider Alice who starts interacting with her favorite Web search engine by submitting the query "`new york hotel`". This clearly represents a simple search task, i.e., Alice presumably needs to "*reserve a hotel room in New York*". Current query suggestion mechanisms provide alternative related queries by only focusing on the task behind this original single query. According to this approach, candidate suggested queries may be, for example, "`cheap new york hotels`", "`times square hotel`", "`waldorf astoria`", etc. Although these recommended queries are all related to the original query, they still refer to the *same* search task.

Now, let us suppose we are able to discover *what* and *how* user search tasks are composed together to accomplish even more complex tasks, which in the following are referred to as *missions* [67]. If this is the case, we could use such inferred knowledge for conjecturing the *mission* behind Alice's queries. To some extent this means that, besides recognizing that the current Alice's task is related to "*booking a hotel in New York*", the Web search engine would also be able to predict that her global mission is concerned with "*planning a travel to New York*". On the basis of this prediction, we could provide Alice with suggestions that are not *only* related to her current task (as in traditional *query recommendation*), but that also refer to *other* tasks being part of her mission as a whole (*task recommendation*). In our example, this means recommending to Alice those tasks having underpinning queries such as "`mta subway`", "`broadway shows`", "`jfk airport shuttle`", etc.

In order to realize the scenario described above, three key research challenges have to be addressed: *(i)* finding an effective method for discovering interleaved search tasks from the stream of queries stored in Web search engine query logs, *(ii)* exploiting the collective knowledge provided by Web search engine users in order to build a *synthesized* representation of user search tasks, and *(iii)* devising how people are used to compose each individual search task for realizing their bigger missions, thus for generating novel *task recommendations*.

The first research issue has been exhaustively addressed and described in Chapter 4, where several approaches and techniques have been proposed in order to discover *task-oriented sessions*. To this end, in the following of this Chapter we refer to the QC-HTC *query clustering* approach for discovering user search tasks, which has proven to outperform other state-of-the-art solutions yet keeping down its computational cost [83].

5.1.1 Contribution

The main contribution of this Chapter refers to the second and third research challenges stated above, which somehow complete the roadmap we already sketched in [132, 133]. In particular, we introduce a *synthesized* representation of search tasks built by exploiting the large number of search tasks performed by several users and extracted from the search sessions stored on a query log. Moreover, we propose a *graph-based* model for describing how people link together search tasks in order to satisfy more complex missions. Finally, we describe how this model might be exploited for generating task recommendations to Web search engine users.

5.1.2 Organization

The rest of the Chapter is organized as follows. Section 5.2 describes related work concerning two overlapping research fields that have been extensively approached, i.e., *recommender systems* and *query suggestion*. However, to the best of our knowledge, no past works have yet addressed the problem tackled in this Chapter. In

Section 5.3, we propose a theoretical model through which we outline the three key steps for building our task recommender system. Section 5.4 presents our *synthesized* representation of search tasks through the *aggregation* of similar search tasks mined from the query log. Then, in Section 5.5 we propose some task recommendation solutions, which leverage on the user search task model introduced. Section 5.6 discusses the experimental evaluation of our task recommender system by showing its effectiveness in suggesting tasks that are different yet related to the ones that users are currently performing. Such recommended tasks predict the set of search intents, which users are going to look for in the next future. Furthermore, even when generated task recommendations are not actually part of future users' searches, we show that they could anyway represent "*surprising*" hints. Indeed, those suggestions might be considered as the most "valuable" for the user, since they are composed of tasks that are both useful and unexpected. Finally, Section 5.7 summarizes the main contribution of this work and points out some possible future research directions.

5.2 Related Work

Although to the best of our knowledge the problem addressed in this Chapter has not yet been investigated, it however concerns two overlapping research fields that have been extensively approached from different perspectives, i.e., *recommender systems* and *query suggestion*.

Recommender systems are widely used in several domains and they have been revealed to be successful especially in electronic commerce. Roughly, they can be divided in two broad classes on the basis of the approach which they rely on, i.e., *content filtering* and *collaborative filtering*. Content filtering approaches generate their recommendations on the basis of the *content* of the items to be suggested. They face serious limitations when dealing with multimedia content and, more importantly, their suggestions are not influenced by the human-perceived *quality* of contents. On the other side, collaborative filtering solutions are based on the *preferences* expressed by other users.

Query suggestion techniques address specifically the problem of recommending queries to Web search engine users and they propose solutions and evaluation metrics specifically tailored to the Web search domain.

During last years, several different techniques have been proposed, yet they all have in common the exploitation of usage information recorded in Web search engine query logs [120]. Many approaches extract the information used from the plain set of queries recorded in the log, although there are several works that take into account the chains of queries that belong to the same search session [105]. The first category contains techniques that employ clustering algorithms to determine groups of related queries that lead users to similar documents [139, 10, 21]. The most "representative" queries in the clusters are then returned as suggestions. Other solutions employ the reformulations of the queries issued by previous users [69], or propose as suggestions the frequent queries that lead in the past users to retrieve similar results [15].

Baeza-Yates *et al.* [14] exploit click-through data as a way to provide recommendations. The method proposed by the authors is based on the concept of *Cover Graph* (CG). A CG is a bipartite graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u that was an answer for the query. Suggestions for a query q are thus obtained by accessing the corresponding node in the CG and by extracting the related queries sharing more URLs. Experimental results show that the sharing of clicked URLs is effective for devising alternative yet related queries.

Moreover, Fonseca *et al.* [47] use an association rule mining algorithm to devise frequent query patterns. These patterns are inserted in a query relation graph which allows "concepts" (e.g., queries that are synonyms, specializations, generalizations, etc.) to be identified and suggested.

Boldi *et al.* introduce the concept of *Query Flow Graph* (QFG), an aggregated representation of the information contained in a query log [28]. A QFG is a digraph in which nodes are queries and the edge connecting node q_i to q_j is weighed by the probability that users issue query q_j after issuing q_i. Authors highlight the utility of their model in two applicative scenarios, i.e., *logical sessioning* and *query recommendation.*

Baraglia *et al.* [17] propose a model for query recommendation, which address a newly introduced problem, i.e., the *Search Shortcut Problem* (SSP), which basically consists in recommending *successful* queries, namely those queries that in the past allowed other users to satisfy similar information needs. Moreover, Baraglia *et al.* [18, 19] present a study of the effects of time on recommendations generated by using the QFG introduced by [28]. Indeed, user search interests change over time and the knowledge extracted from query logs may suffer of aging effects, as new interesting search topics appear. In order to overcome this issue, authors propose a novel incremental algorithm for keeping up-to-date the QFG recommendation model, without having to rebuild it from scratch every time freshest query log data happen. On the basis of these results, Broccolo *et al.* [31] propose two novel query recommendation algorithms that *incrementally* update the model on top of which suggestions are generated, taking care of each new processed query.

Jones *et al.* discussed another approach based on query refinement/substitution [69]. Here, the goal is to generate a new query to replace a user's original ill-formed search query in order to enhance the relevance of retrieved results. Such technique includes a number of tasks, like spelling error correction, word splitting, word merging, word stemming, phrase segmentation, and acronym expansion.

Finally, the importance of rare query classification and suggestion recently attracted a lot of attention from the information retrieval community [42, 34]. To this end, Broccolo *et al.* [32] propose an efficient and effective recommendation algorithm that "*covers*" rare queries, i.e., *long-tail* queries, by exploiting the Search Shortcut model introduced in [17].

5.3 Anatomy of a Task Recommender System

In this Section, we show the overall process of generating *task recommendations* to users of Web search engines. Our strategy is based on the historical search data stored on query logs by operating the following three steps:

1. The first step, called *Task Discovery*, reveals the *interleaved* search tasks performed by each user whose search sessions are recorded in the query log [83] and it has been deeply described in Chapter 4.

2. The second step, called *Task Synthesis*, provides an *aggregated* representation of the user tasks previously discovered by exploiting the knowledge coming from the large number of user search tasks contained in all the identified task-oriented sessions.

3. The last step, called *Task Modeling*, relies on the *synthesized tasks* obtained by the second step to build a *user task model*, that is a *graph-based* model of task pairs relatedness, which is finally used to generate task suggestions.

In the following, we explore these three aspects separately.

1) Task Discovery. This step starts from a Web search engine query log storing queries submitted by users in the past, along with other information, such as userIDs, timestamps, clicked URLs, etc. We denote such log with $\mathcal{QL}$, while $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$ is the set of distinct users which submitted the queries. Furthermore, let $q_i \in \mathcal{QL}$ be a generic query issued by user u_i and $q_{i,j} \in \mathcal{QL}$ be the j-th query issued by u_i.

Moreover, let $\mathcal{S}_i = \langle q_{i,1}, q_{i,2}, \ldots, q_{i,|\mathcal{S}_i|}\rangle$ be the sequence of *all* the queries $q_i \in \mathcal{QL}$ issued by user $u_i \in \mathcal{U}$, chronologically ordered, so that $\mathcal{QL} = \bigcup_{i=1}^{N} \mathcal{S}_i$. Although each $\mathcal{S}_i$ represents the "flat" sequence of queries issued by u_i, we argue that $\mathcal{S}_i$ might contain several interleaved search tasks, each one referred to a specific information need. As described in Chapter 4, in [83] we devised a technique that allows to detect, for each user u_i, the set of interleaved search tasks Θ_i, $\Theta_i = \{\theta_i^1, \theta_i^2, \ldots, \theta_i^{|\Theta_i|}\}$, which u_i performs within $\mathcal{S}_i$. Each Θ_i is a *task-oriented session* and it may be viewed as a *partitioning* of $\mathcal{S}_i$, i.e., a division of $\mathcal{S}_i$ into non-empty disjoint sets that completely cover $\mathcal{S}_i$.

More formally, $\Theta_i \subset 2^{\mathcal{S}_i}$, such that:

– $\forall \theta_i^j \in \Theta_i,\ \theta_i^j \neq \emptyset$;

– $\forall \theta_i^j,\ \theta_i^k \in \Theta_i,\ \theta_i^j \neq \theta_i^k \implies \theta_i^j \cap \theta_i^k = \emptyset$;

– $\bigcup_{j=1}^{|\Theta_i|} \theta_i^j = \mathcal{S}_i$.

In order to build our recommender system, we need to consider each Θ_i as a *sequence* of tasks rather than a simple set. To this end, we exploit the information about the chronological order of tasks. However, sorting by time an interleaved task-oriented session is not immediate, due to the presence of *multi-tasking*: a generic task $\theta_i^j \in \Theta_i$ might be chronologically overlapped with another task $\theta_i^k \in \Theta_i$. We address this issue by adopting the straightforward solution of chronologically ordering the tasks by looking at the timestamps of the first query occurring in each task.

Anyway, whether the temporal information on search task submission is taken into account or not, we can define the union set of all the user tasks as follows:

$$\Theta = \bigcup_{i=1}^{N} \Theta_i.$$

In this Chapter, we only sketch the technique we adopted to obtain the task-oriented "*sessioning*" Θ_i of each S_i. Those readers who are interested in knowing further details about such technique are invited to refer to Chapter 4. Roughly, we exploit a *query clustering* algorithm, which allows us to avoid any supervised learning step. Each resulting cluster of queries, which univocally identifies a user search task, contains queries that are not necessarily issued sequentially. Thus, a similar clustering-based approach allows us to deal with the multi-tasking search behavior of users.

Two core aspects are particularly crucial, namely how the *similarity* between any two queries is measured and what suitable clustering algorithm we have to adopt. Concerning the first issue, in [83] we defined a set of *task-based query similarity* functions, which exploit *temporal*, *content-based* and *semantic-based* query features. Moreover, in the same work we evaluated several clustering algorithms, which use the above similarity functions.

In this work, we use the best solution devised in [83] to discover the global set Θ of task-oriented sessions: the QC-HTC query clustering method in combination with a specific task-based similarity function. QC-HTC relies on a graph-based heuristic and it has proven to outperform other state-of-the-art solutions yet holding down the overall computational cost.

2) Task Synthesis. In this step, starting from the user search tasks Θ extracted from query log $\mathcal{QL}$, we resort to recognize *similar* tasks across different users and eventually replace each $\theta_i^j \in \Theta_i$ with a *synthesized* task $T_h \in \mathcal{T}$, where $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$. Each T_h may be considered as a representative for an aggregation composed of the similar tasks performed by several distinct users. It is worth noting that this step reduces the overall *space of tasks*, because, in general, we can state that $|\mathcal{T}| \ll |\Theta|$.

More formally, this step aims to define a surjective function $f : \Theta \longmapsto \mathcal{T}$, so that we can rewrite each task-oriented session Θ_i in terms of the tasks in $\mathcal{T}$:

$$\Theta'_i = \bigcup_{j=1}^{|\Theta_i|} f(\theta_i^j). \tag{5.1}$$

If necessary, we can maintain the chronological order the tasks in Θ_i after the mapping to the synthesized tasks in $\mathcal{T}$. In Section 5.4, we propose a specific task synthesis function.

3) Task Modeling. In order to build the user task model, we create a weighted directed graph $G_{\mathcal{T}} = (\mathcal{T}, E, w)$, where the set of synthesized tasks $\mathcal{T}$ corresponds to the nodes of the graph and E is the set of edges of the graph connecting pairs of tasks $T_i, T_j \in \mathcal{T}$.

Roughly, there exists an edge between two tasks if and only if they are "related" one to each other. To this end, we adopt a *weighting function* $w : \mathcal{T} \times \mathcal{T} \longmapsto [0, 1]$ for measuring the "*task relatedness*" between each pair of tasks. Therefore, the set of edges in the graph is composed of all the pairs of nodes whose relatedness is greater than 0, i.e., $E = \{(T_i, T_j) \in \mathcal{T} \mid w(T_i, T_j) > 0\}$.

Note that the overall resulting graph depends on the choice of the weighting function. Indeed, several measures may be used for computing the relatedness between any two search tasks. Section 5.5 discusses the functions we used to weight the graph.

The primary goal of this graph-based model is to represent user search behaviors on a "*task-by-task*" perspective instead of on a traditional "*query-by-query*" basis, thus understanding how users possibly "compose" several different search tasks on the basis of their task relatedness for achieving a *mission*.

Finally, this graph-based task model is used to generate suitable *task recommendations* to Web search engine users. Given a user who is interested in (or has just performed) a task $T_i \in \mathcal{T}$, we can simply look at the graph $G_{\mathcal{T}}$ and retrieve $\mathcal{R}_m(T_i)$, i.e. the set of m nodes that are the most related to T_i. More formally, we define task recommendations $\mathcal{R}_m$ for a given set of tasks as follows:

Definition 5.3.1 (Task Recommendations $\mathcal{R}_m$)
Let $G_{\mathcal{T}} = (\mathcal{T}, E, w)$ be the graph-based model of user search tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$, where an edge $e \in E$ is weighted with a relatedness function w of its incident nodes. Given the set of tasks $\mathcal{T}^* \subseteq \mathcal{T}$ already performed by a user, the top-m recommended tasks are given by the sorted set $\mathcal{R}_m(\mathcal{T}^*) \subseteq \mathcal{T}$, such that:

- *$|\mathcal{R}_m(\mathcal{T}^*)| = m$;*
- if $T_i \in \mathcal{T}^*$ and $T_j \in \mathcal{R}_m(\mathcal{T}^*)$ then $w(T_i, T_j) \geq w(T_h, T_k)$ for all $T_h \in \mathcal{T}^*$ and $T_k \in \mathcal{T} \setminus \mathcal{R}_m(\mathcal{T}^*) \setminus \mathcal{T}^*$.

5.4 Task Synthesis

Each user task $\theta_i^j \in \Theta$, discovered as previously described in Section 5.3, is a *set* of queries that have some similarities and are related to the same activity. In the following, we describe how discovered search tasks might be represented.

5.4.1 Basic Task Representation

The immediate way of representing a search task is constituted by a "virtual document", i.e., a *task document*, corresponding to the union set of all the queries, which the task itself is composed of. More formally, given the generic task $\theta_i^j \in \Theta$, its task document representation is defined as follows:

$$td_i^j = \bigcup_{q \in \theta_i^j} q.$$

Eventually, we come up with a global set of task documents $\mathcal{TD} = \bigcup_{\theta_i^j \in \Theta} td_i^j$. However, two users looking for exactly the same task might formulate two different set of queries, depending on their familiarity with the specific terminology of a knowledge domain. Thus, the representation of a task as a mere set of queries may suffer of a sparsity issue, namely the probability of two users sharing the same search task, i.e., *exactly* the same set of queries, is usually very small. In this regard, next Section is devoted to describe how task sparseness may be alleviated.

5.4.2 Task Document Clustering

In order to overcome the sparsity issue of user search tasks, we exploit the knowledge extracted from query logs to "*collapse*" the set of discovered tasks into another set of *synthesized* tasks $\mathcal{T}$. To this end, we propose to *aggregate*, namely cluster, the task documents contained in $\mathcal{TD}$.

In this way, the problem of finding a synthesized representation of search tasks may be reduced to the problem of clustering similar (virtual) text documents, which is a well-investigated research topic [128, 149, 150]. In general, we can resort to use any one of the various clustering algorithms for textual documents proposed so far. In Section 5.6, we shall describe our choice for the textual clustering methods we have tested in the experiments we conducted.

More formally, let $\mathcal{T} = \{T_1, T_2, \ldots T_K\}$ be the identifiers associated with a set of disjoint clusters of task documents. Therefore, let $c : \mathcal{TD} \longmapsto \mathcal{T}$ be a surjective function defined as follows:

$$c(td_i^j) = T_h \text{ if the cluster identified by } T_h \text{ contains } td_i^j. \tag{5.2}$$

From function c we can easily derive function f, discussed in Section 5.4, which allows each task-oriented session Θ_i to be re-written in terms of the synthesized tasks in $\mathcal{T}$:

$$\forall \theta_i^j \in \Theta_i,\ f(\theta_i^j) = c(td_i^j), \tag{5.3}$$

where td_i^j is obtained from the original search task θ_i^j, as specified in Section 5.4.1.

5.5 Task Modeling

The set of synthesized tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$ can be in turn represented by using the *graph-based* model proposed in Section 5.3, i.e., $G_{\mathcal{T}} = (\mathcal{T}, E, w)$.

A key point for building the weighted digraph $G_{\mathcal{T}}$ deals with devising a suitable function w for estimating how much any two tasks are likely to be related to each other. The resulting graph can thus be exploited for understanding any possible relationship among different search tasks.

Furthermore, according to Def. 5.3.1, edges between search tasks might be harnessed for providing task recommendations for each $T_i \in \mathcal{T}$.

In this work, we propose four different edge weighting schemes, which result in four distinct user task models.

5.5.1 Random-based (*baseline*)

The simplest weighting score associated with the graph-based model $G_{\mathcal{T}} = (\mathcal{T}, E, w)$ is the one which equally assigns the same constant weight to *all* of its edges. More formally, given $|\mathcal{T}| = K$ then any edge (T_i, T_j) is weighed as follows:

$$w(T_i, T_j) = w_{random}(T_i, T_j) \ = \ \frac{1}{K-1}. \tag{5.4}$$

Using this edge weighting scheme for generating task recommendations is somehow equivalent to provide *random* lists of suggestions. Indeed, according to Def. 5.3.1, generating recommendations for a task $T_i \in \mathcal{T}$ involves retrieving the m neighbors of T_i in the corresponding graph $G_{\mathcal{T}}$ whose edges have the m highest weight values, i.e., the top-m neighboring nodes. However, since all the edges $(T_i, T_j) \in E$ are equally-labeled with the constant value $\overline{w} = 1/(K-1)$, the resulting set of task recommendations $\mathcal{R}_m(T_i)$ corresponds to a *random subset* of all the neighbors $\mathcal{R}_m(T_i)$ of T_i in $G_{\mathcal{T}}$.

In the following, we will consider this random-based recommendation strategy as the *baseline* of our experiments.

5.5.2 Sequence-based

An immediate way to enhance the weighting scheme described above is to consider information about the *frequency* of tasks when computing their relatedness thus giving somehow more “importance” to highly-frequent tasks. Moreover, up to now we have not taken into account the chronological order of tasks as they were issued by users. In other words, each task-oriented session Θ_i, as well as the same session Θ'_i after the task synthesis phase, have been considered as simple *sets* of tasks, without any information about their issuing time.

In order to consider each task-oriented session Θ_i as a *sequence* rather than a set, we need to take care of the time ordering between tasks. This can be done by considering the issuing time of the chronologically-first queries of every task $\theta_i^j \in \Theta_i$, thus obtaining the sequence $\Theta_i = \langle \theta_i^1, \theta_i^2, \ldots, \theta_i^{|\Theta_i|} \rangle$. This ordering is also maintained after the task synthesis step that transforms each Θ_i into Θ'_i.

By considering sessions as “sequences” we can define another measure of task relatedness, which in turn could be exploited as another weighting edge function. We define the support of a sequence $\langle T_i, T_j \rangle$, namely *seq-supp*$(\langle T_i, T_j \rangle) \in [0, 1]$, as the fraction of task-oriented session Θ'_i where T_i appears before T_j, which leads us to the following task relatedness score:

$$w(T_i, T_j) = w_{\textit{seq-supp}}(T_i, T_j) \;=\; \textit{seq-supp}(\langle T_i, T_j \rangle). \tag{5.5}$$

5.5.3 Association-Rule based

After the task synthesis step, the collection of all the task-oriented sessions may be considered as a *transactional database*, whose *items* are exactly the synthesized task identifiers. Analogously to the classical market basket analysis, where transactional databases composed of sets of purchased products can be analyzed for devising better and more fruitful marketing strategies, here we can analyze our transactional database of task-oriented sessions to reveal interesting patterns and relatednesses between user search intents.

We can thus derive a measure of task relatedness by following the original definition formulated by Agrawal *et al.* for the problem of *association rule mining* from transactional databases [4]. According to this approach, let $\mathcal{I} = \{i_1, i_2, \ldots, i_K\}$ be a set of K binary attributes, called *items*. Moreover, let $\mathcal{D} = \{t_1, t_2, \ldots, t_N\}$ be a set of N transaction, called *database*. Each transaction $t_i \in \mathcal{D}$ has a unique identifier and contains a subset of the items in $\mathcal{I}$.

A *rule* is defined as an implication of the form $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{I}$ are two sets of items, i.e., *itemsets*, occurring in the transactions of the database $\mathcal{D}$. The itemsets X and Y, where $X \cap Y = \emptyset$, are called *antecedent* (or LHS) and *consequent* (or RHS) of the rule, respectively.

In order to select interesting rules, various measures of significance and interest can be used. The best-known measures are *support* and *confidence*. The *support* of a rule $X \Rightarrow Y$, namely $supp(X \cup Y) \in [0,1]$, is defined as the fraction of transactions that contain the itemset $X \cup Y$. The *confidence* of a rule, i.e., $conf(X \Rightarrow Y) \in [0,1]$, is defined as:

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}.$$

The confidence of a rule can be viewed as the conditional probability $P(Y|X)$, namely the probability of finding the RHS of the rule in those transactions that already contain the LHS.

For our purposes, it is enough to mine association rules, where both the LHS and RHS of each rule are *singleton* sets, i.e. rule of the form $T_i \Rightarrow T_j$, where $T_i, T_j \in \mathcal{T}$ and $T_i \neq T_j$.

Finally, given the *minimum support* (*min_supp*) and the *minimum confidence* (*min_conf*) thresholds used for learning a proper set of association rules, we are able to derive the two following edge weighting functions:

$$w(T_i, T_j) = w_{ar\text{-}supp}(T_i, T_j) = supp(T_i \Rightarrow T_j). \tag{5.6}$$

$$w(T_i, T_j) = w_{ar\text{-}conf}(T_i, T_j) = conf(T_i \Rightarrow T_j). \tag{5.7}$$

5.6 Experiments

In the following, we describe how we actually generate task suggestions using our graph-based user task model in combination with the four weighting schemes proposed in Section 5.5. Moreover, we evaluate and compare the quality of task recommendations provided by our technique. Firstly, we measure the ability of our task recommender to correctly foresee what tasks a user might want to do in future search sessions (Section 5.6.1.3). Furthermore, we present a user study aimed at evaluating the capability of our technique to provide "*surprising*" recommendations (Section 5.6.3). Finally, we provide some anecdotal evidences showing, in practice, what kind of recommendations are generated by our method (Section 5.6.4).

5.6.1 Experimental Setup

All the experiments are conducted on the 2006 AOL query log, which is our target data set $\mathcal{QL}$. This Web search engine query log is a very large and long-term collection consisting of about 20 million worth of Web queries issued by about 657,000 users over a period of 3 months (from 03/01/2006 to 05/31/2006)[1].

We first pre-process the query log in order to clean it from empty and "nonsense" queries as well as for removing *stop-words*.

[1]`http://sifaka.cs.uiuc.edu/xshen/aol_querylog.html`

Indeed, we consider the set $\mathcal{QL}_{top-600} \subset \mathcal{QL}$, which contains the 600 user sessions with the highest number of queries issued during the first week of logging. This data set consists of $58,037$ queries, meaning about 97 queries per user over a week on average, i.e., about 14 queries per user every day. Then, we randomly partition $\mathcal{QL}_{top-600}$ in two disjoint subsets $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{A}$ composed of $48,257$ queries corresponding to 500 user sessions is used as *training* and $\mathcal{B}$ composed of $9,780$ queries corresponding to 100 user sessions is the *test* data set.

In $\mathcal{A}$ the *longest* user session, i.e., the session with the highest number of queries, is composed of $1,774$ queries, while the *shortest* session contains only 2 queries. We exploit this data set for building our graph-based user task model, following the three steps described in Sections 5.3.

On the other hand, in the testing data set $\mathcal{B}$ the longest user session consists of 454 queries, whereas the shortest user session is composed of 7 queries. This data set is in turn used for evaluating the quality of our proposed recommendation mechanisms.

5.6.1.1 Task Discovery

We extract the set of all the task-oriented sessions from $\mathcal{A}$, i.e., $\Theta_{\mathcal{A}}$, using our query clustering method QC-HTC presented in a previous work of ours [83], thus obtaining a total amount of $8,301$ search tasks. The maximum number of discovered tasks for a single user search session is 163, whereas the minimum is only 1. Therefore, on average each user asks for about 17 tasks, meaning that every day a user performs about 2.5 search tasks.

5.6.1.2 Task Synthesis

We build a set of *task documents* $\mathcal{TD}_{\mathcal{A}}$ from the original set of tasks $\Theta_{\mathcal{A}}$ discovered before. In particular, each task is represented by a virtual text document that is obtained throughout the union set of all its composing queries. We then implement the task synthesis function c described in Section 5.4.2 as a traditional *text document clustering* function. To this end, we use the CLUTO[2] toolkit for clustering our collection of task documents $\mathcal{TD}_{\mathcal{A}}$. In particular, there are three different parameters we have to deal with: *(i)* the cluster method, *(ii)* the objective function, and finally *(iii)* the number K of output clusters.

Concerning the first parameter, we select the following three clustering methods: **rb**, **rbr**, and **direct**. The clustering method **rb** computes the desired K-way clustering solution by performing a sequence of $K-1$ repeated bisections. In this approach, the original data set is first clustered into two groups, then one of these groups is selected and bisected further. This process continues until the desired number of clusters is found. During each step, the cluster is bisected so that the resulting 2-way clustering solution optimizes the chosen clustering criterion function.

[2] `http://glaros.dtc.umn.edu/gkhome/views/cluto`

The clustering method **rbr** works very similar to the above approach but, at the end, the overall solution is *globally* optimized with respect to the selected objective function. Finally, in the **direct** method the desired K-way clustering solution is computed by *simultaneously* finding all K clusters. In general, computing a K-way clustering directly is slower than clustering via repeated bisections. In terms of quality, for reasonably small values of K (usually less than $10 - 20$), the direct approach leads to better clusters than those obtained via repeated bisections. However, as K increases, the repeated-bisecting approach tends to be better than direct clustering.

The second aspect to address involves the choice of a proper criterion function. In particular, we restrict our selection to the following three objective functions: *i2*, *e1*, and *h2*. The first function, i.e., *i2*, tends to maximize the *intra-cluster similarity*, whereas *e1* is designed to minimize the *inter-cluster similarity* (or, analogously, to maximize the separation between distinct clusters), so that lower scores are better. Lastly, the criterion function *h2* mixes up both the two functions above, thereby it measures the ratio of *intra/inter cluster similarity*, i.e., *i2/e1*. Such score has to be maximized, namely it is arguable to have the most similar intra-cluster measurement (*i2*) over the most separated inter-cluster measurement (*e1*). The various criterion functions can sometimes lead to significantly different clustering solutions. In general, the *i2* and *h2* criterion functions lead to very good clustering solutions, whereas the *e1* function leads to solutions that contain clusters that are of comparable size. However, the choice of the right criterion function depends on the underlying application area and an experimental phase is mandatory before one can select the appropriate for her needs.

Finally, concerning the last parameter, i.e., the number K of final clusters to be produced, we start with $K = 20$ clusters growing up to $K = 1500$ by incrementing K of 40 at each step.

In order to devise a suitable number K of final clusters to be produced, we perform several clustering runs using different combinations of the above three parameters. Therefore, in the following Figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9, we show the curve progress of the selected objective functions by changing the number K of clusters and the clustering approach.

Intuitively, by increasing the number of final clusters K to be produced, the *i2* objective function *monotonically* increases as well. Each plot depicted in Figure 5.1, Figure 5.4, and Figure 5.7 shows in fact that a blind use of intra-cluster similarity as a stopping criterion will cause trouble. Indeed, the maximum intra-cluster similarity is asymptotically obtained when the number of clusters K reaches the number of total task documents to be clustered, i.e., when each cluster is a *singleton set* containing only one task document since then the members of the cluster are as similar to each other as possible (actually, they are *self-similar*).

Moreover, by increasing the number of final clusters K to be produced, the *e1* objective function *monotonically* decreases (Figure 5.2, Figure 5.5, and Figure 5.8). Again, this intuitively makes sense. When there is just one cluster, its centroid will be essentially the same as the centroid of the whole collection, meaning that the

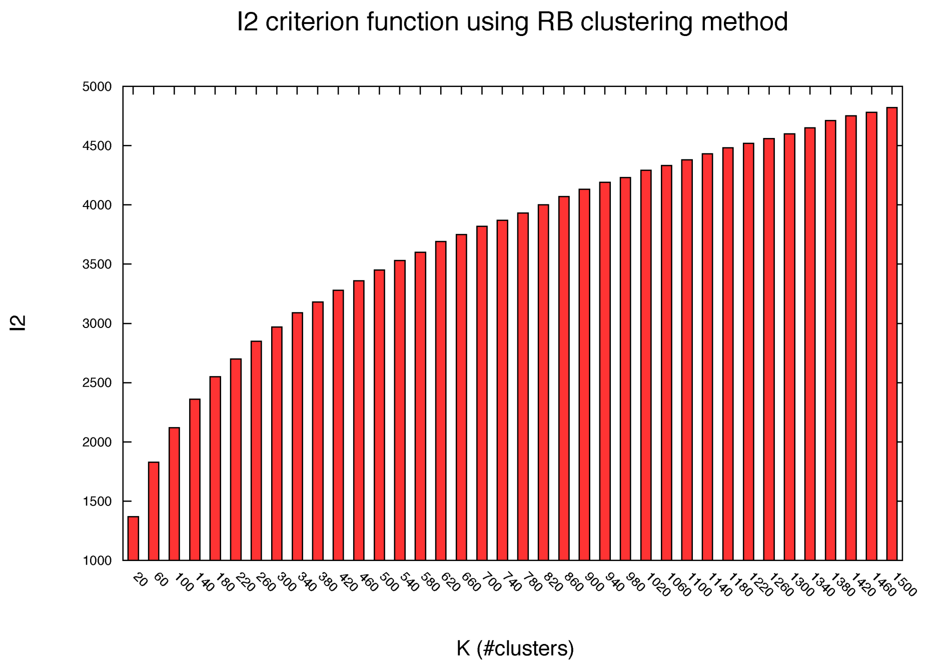


Figure 5.1: Clustering using the **rb** method: curve progress of the *i2* criterion function by varying the number K of produced clusters.

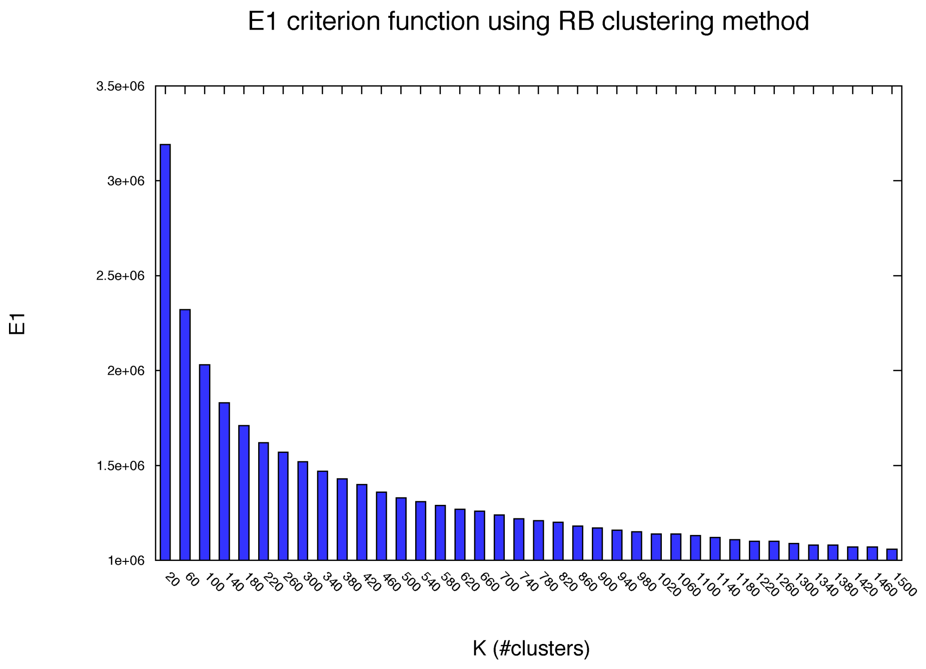


Figure 5.2: Clustering using the **rb** method: curve progress of the *e1* criterion function by varying the number K of produced clusters.

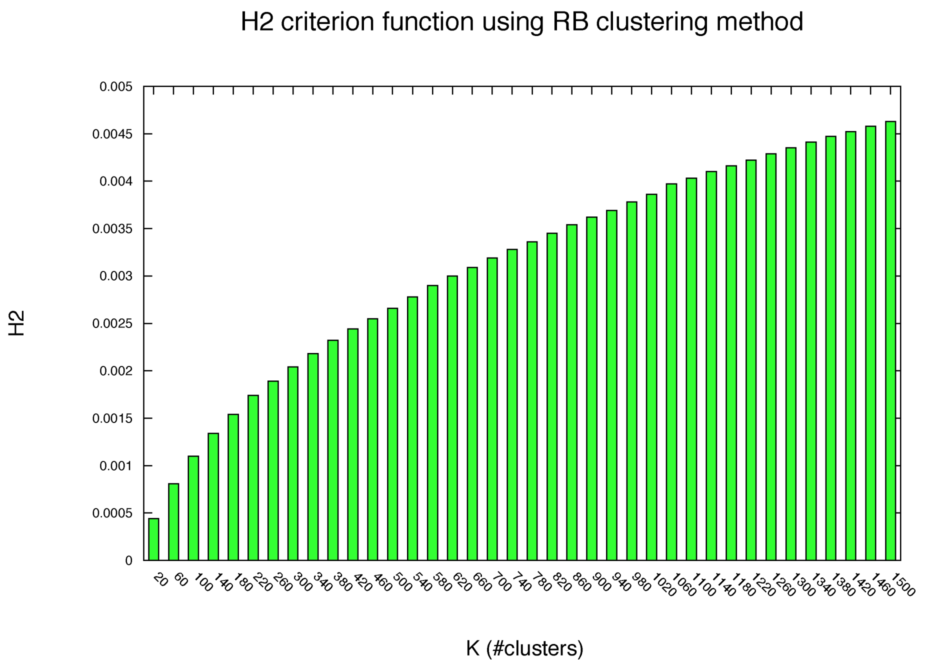


Figure 5.3: Clustering using the **rb** method: curve progress of the *h2* criterion function by varying the number K of produced clusters.

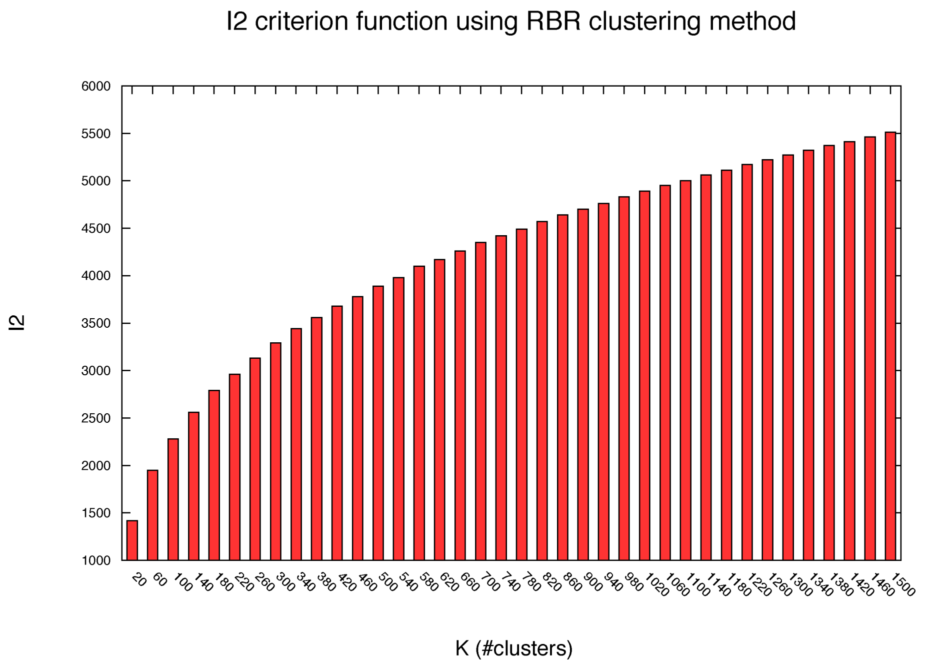


Figure 5.4: Clustering using the **rbr** method: curve progress of the *i2* criterion function by varying the number K of produced clusters.

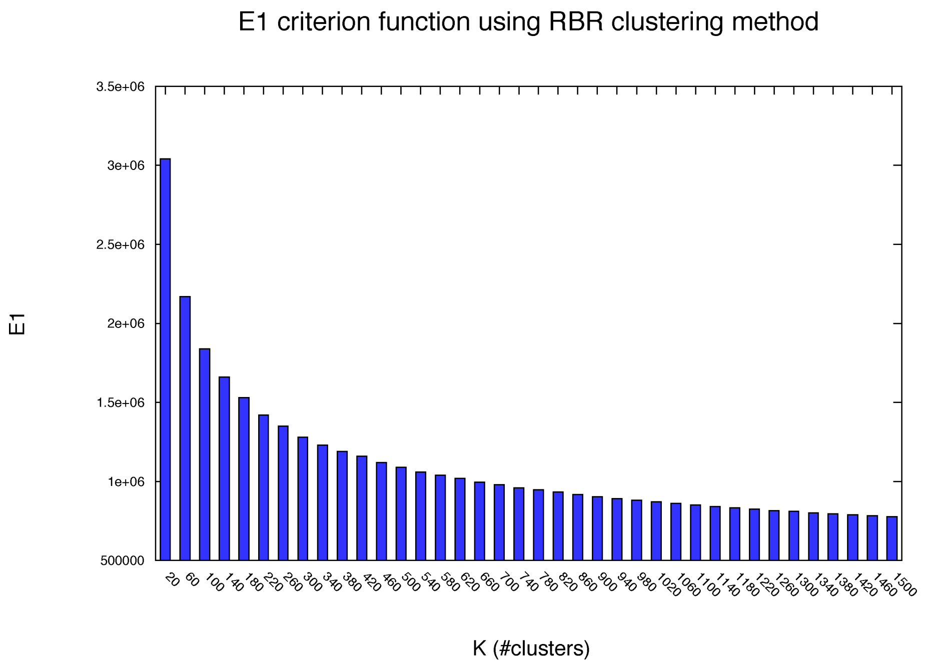


Figure 5.5: Clustering using the **rbr** method: curve progress of the *e1* criterion function by varying the number K of produced clusters.

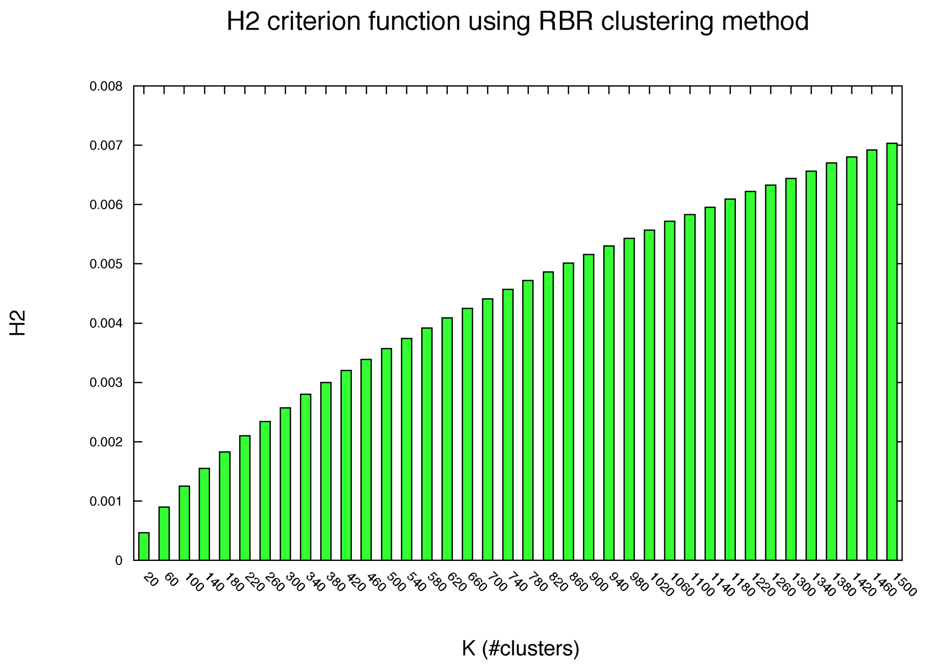


Figure 5.6: Clustering using the **rbr** method: curve progress of the *h2* criterion function by varying the number K of produced clusters.

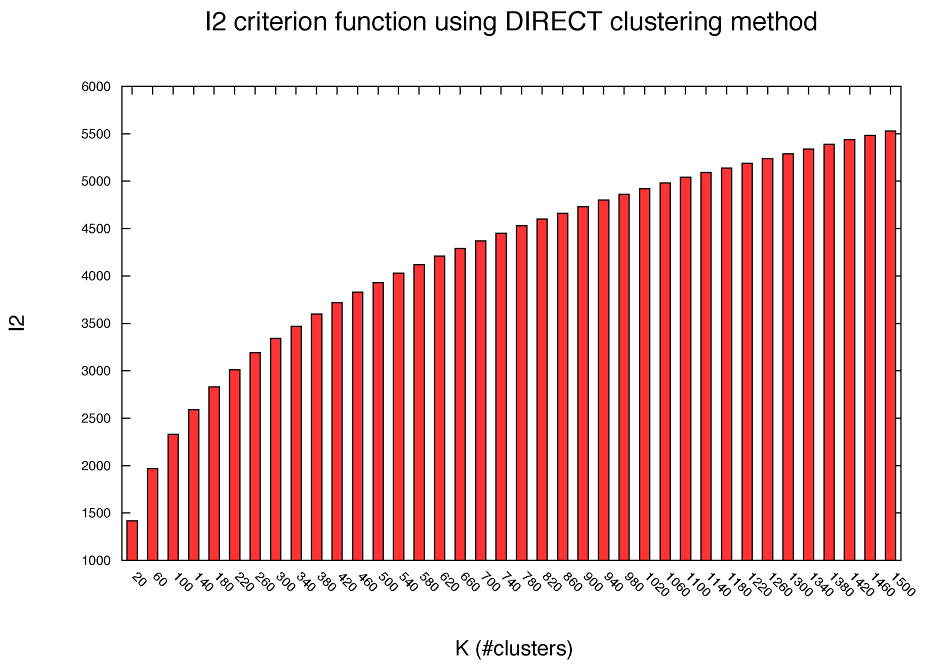


Figure 5.7: Clustering using the **direct** method: curve progress of the *i2* criterion function by varying the number K of produced clusters.

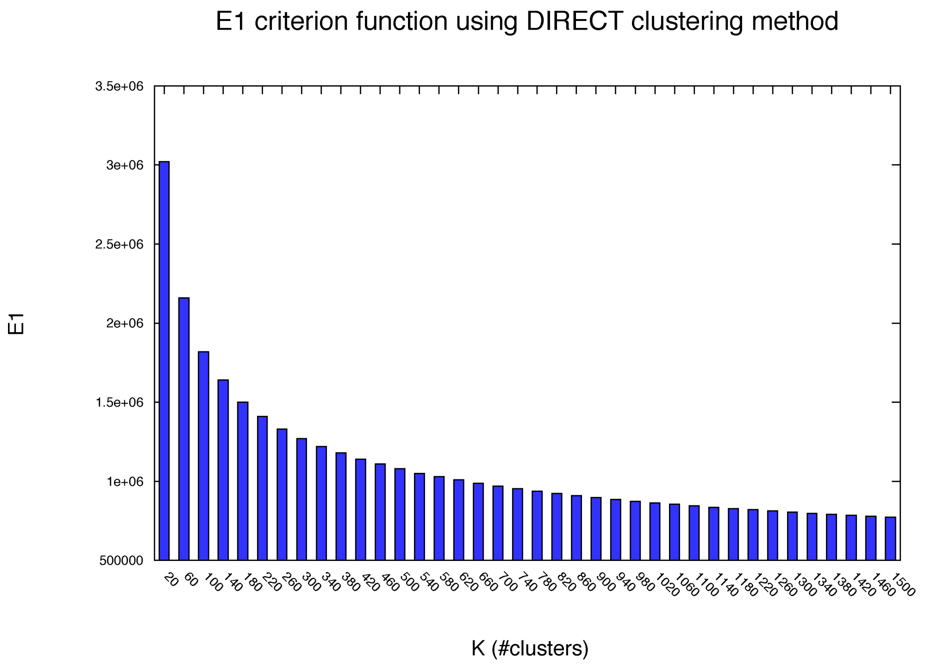


Figure 5.8: Clustering using the **direct** method: curve progress of the *e1* criterion function by varying the number K of produced clusters.

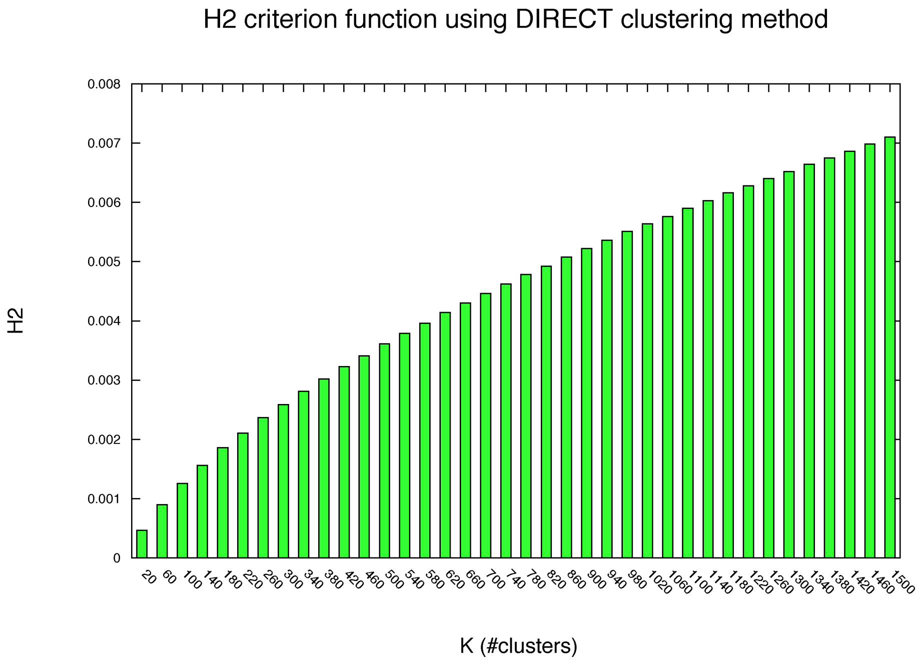


Figure 5.9: Clustering using the **direct** method: curve progress of the *h2* criterion function by varying the number K of produced clusters.

separation is low (i.e., the inter-cluster similarity is high). Conversely, when there are more clusters, their centroids will generally be more different from the centroid of the collection (i.e., the inter-cluster similarity falls down).

From the two considerations above, it follows also that the *h2* objective function *monotonically* increases as the number of output clusters increase (Figure 5.3, Figure 5.6, and Figure 5.9) because it is computed as the ratio of a monotonically-increasing function (i.e., *i2*) and a monotonically-decreasing function (i.e., *e1*).

Of course, these results force us to devise a proper *trade-off* in order to select the best combination of parameters. Indeed, no matter what the chosen objective function is, the optimal solution would always be the one that produces as many clusters K as the total number of task documents to be clustered. Since this strategy appears unfeasible, to this end we identify $K = 1024$ as a reasonable value for the number of final clusters to be produced in combination with the **rbr** clustering method and the *i2* criterion function. In fact, for $K > 1024$ the curve progress of the criterion function appears to increase slower enough than the quick growing noticed for $K < 1024$. Eventually, we obtain a set of synthesized tasks $\mathcal{T} = \{T_1, T_2, \dots, T_{1024}\}$.

5.6.1.3 Task Modeling

According to the graph-based user task model, at least four different graphs can be built over the resulting set of synthesized tasks $\mathcal{T}$. These graphs differ from each other on the basis of the chosen approach for computing their edge weighting scores, as described in Section 5.5.

We build four graphs according to the edge weighting functions presented in Eq. 5.4, Eq. 5.5, Eq. 5.6, and Eq. 5.7, respectively. Furthermore, concerning the last two graphs obtained by using the association-rule weighting scheme, we compute the *frequent itemsets* of the transaction database of tasks throughout the *Apriori* algorithm [4] and varying the min_supp threshold from 0.004% up to 100%. Therefore, from the frequent itemsets discovered before, we extract the set of all association rules by changing the min_conf threshold from 5% up to 100%.

On the basis of these graph-based models built on top of the training data set $\mathcal{A}$, we generate task recommendations for the users of the testing data set $\mathcal{B}$.

First, we extract the set of task-oriented sessions $\Theta_{\mathcal{B}}$ from $\mathcal{B}$, which contains a total amount of 1,762 search tasks. The maximum number of tasks discovered for a single user search session is 97, whereas the minimum is 1. Therefore, on average each user asked for about 17.6 tasks, meaning that every day a user accomplished about 2.5 search tasks. Each discovered task $\theta_b^k \in \Theta_{\mathcal{B}}$ is in turn represented using the union set of all its queries as described in Section 5.4.1, thus resulting in another set of task documents $\mathcal{TD}_{\mathcal{B}}$.

Eventually, each task-oriented session $\Theta_b \in \Theta_{\mathcal{B}}$ is expressed as a sequence of synthesized tasks Θ'_b:

$$\Theta_b = \langle \theta_b^1, \theta_b^2, \ldots, \theta_b^{|\Theta_b|} \rangle \rightsquigarrow \Theta'_b = \langle T_{b_1}, T_{b_2}, \ldots, T_{b_{|\Theta_b|}} \rangle \subseteq \mathcal{T}.$$

To this end, for each task $\theta_b^k \in \Theta_b$ we estimate which synthesized task $T_{b_c} \in \mathcal{T}$ is the most suitable *candidate* to contain it. Thus, we consider the task document associated with each original search task, i.e., $td_b^k \in \mathcal{TD}_{\mathcal{B}}$. Given that each synthesized task is a cluster of task documents, we find the similarity between a task and the generic synthesized task T_h as follows:

$$sim(\theta_b^k, T_h) = sim(td_b^k, T_h) = \underset{td_i^j \in T_h}{\text{argmax}}\{cos\_sim(\mathbf{td_b^k}, \mathbf{td_i^j})\},$$

where $cos\_sim(\mathbf{td_b^k}, \mathbf{td_i^j})$ is the *cosine similarity* between the two weighted term vectors $\mathbf{td_b^k}$ and $\mathbf{td_i^j}$, which represent the corresponding task documents td_b^k and td_i^j, respectively. Globally, the candidate synthesized task for containing θ_b^k is computed as follows:

$$T_{b_c} = \underset{T_h \in \mathcal{T}}{\text{argmax}}\{sim(\theta_b^k, T_h)\}.$$

Finally, task recommendations are provided according to the suggestion scheme described in Section 5.3. In particular, once we detect the candidate synthesized task T_{b_c} for each θ_b^k, we are able to lookup the graph $G_{\mathcal{T}}$ and suggest the top-m neighboring tasks $\mathcal{R}_m(T_{b_c})$, i.e., the m neighboring tasks of the candidate T_{b_c} whose edges have the highest weighting values.

5.6.2 Evaluating Recommendation Precision

Evaluating the *quality* of provided suggestions is a well-known issue in traditional recommender systems. Indeed, the primary aim of recommender systems is to enhance users' *satisfaction*. Therefore, measuring the overall quality of these systems actually means estimating user's fulfillment. A universally logical measure of user's satisfaction is really hard to devise and most approaches focus on measuring the accuracy using offline evaluation methods. However, "when evaluating a recommender algorithm, it may be inappropriate to use only offline evaluation. Since the recommender algorithm is generating recommendations for items that the user does not already know about, it is probable that the data set will not provide enough information to evaluate the quality of the items being recommended. If an item was truly unknown to the user, then it is probable that there is no rating for that user in the database. If we perform a live user evaluation, ratings can be gained on the spot for each item recommended." [57]

We start evaluating the quality of our task recommender system by measuring the *precision* of generated suggestions.

Each task-oriented session Θ_b can be expressed as a sequence Θ'_b of candidate synthesized tasks as follows:

$$\Theta'_b = \langle T_{b_1}, T_{b_2}, \ldots, T_{b_{|\Theta_b|}} \rangle \subseteq \mathcal{T}.$$

Therefore, we divide each Θ'_b in two disjoint subsequences, i.e., Θ'_{b1} and Θ'_{b2}, such that $\Theta'_{b1} = \langle T_{b_1}, \ldots, T_{b_{\lceil |\Theta_b|/3 \rceil}} \rangle$, i.e., $|\Theta'_{b1}| = \lceil 1/3 \cdot |\Theta'_b| \rceil$ and $\Theta'_{b2} = \Theta'_b - \Theta'_{b1}$. The first subsequence is used to generate task-oriented recommendations, which in turn are evaluated on the second subsequence as follows. For each search task $T_{b_i} \in \Theta'_{b1}$ we are able to provide the set of top-m suggested synthesized tasks, i.e., $\mathcal{R}_m(T_{b_i})$. Thus, we obtain the set of all the generated suggestions for the tasks in Θ'_{b1} as follows.

According to Def. 5.3.1, the set of all the generated suggestions for the tasks in Θ'_{b1} is $\mathcal{R}_m(\Theta'_{b1})$. Generally, since not every task in Θ'_{b1} is able to produce suggestions, at most we have m unique recommended tasks, i.e., $|\mathcal{R}_m(\Theta'_{b1})| \leq m$. Moreover, we obtain the set of tasks $T(\Theta'_{b2})$ contained in the remaining portion of the original session as follows:

$$T(\Theta'_{b2}) = \bigcup_{T_{b_j} \in \Theta'_{b2}} T_{b_j}.$$

Thus, for each task-oriented session Θ'_b we compute the *precision* $p(\Theta'_b)$:

$$p(\Theta'_b) = \frac{|\mathcal{R}_m(\Theta'_{b1}) \cap T(\Theta'_{b2})|}{|\mathcal{R}_m(\Theta'_{b1})|}. \tag{5.8}$$

Of course, the intersection between the two sets of synthesized tasks, i.e., $S_m(\Theta'_{b1})$ and $T(\Theta'_{b2})$, takes only into account the task identifiers and does not care about the actual lexical content of the tasks themselves. Moreover, a global value for the precision is straightforwardly obtained by simply averaging the precision of all the task-oriented sessions in the collection:

$$p(\Theta_{\mathcal{B}}) = \frac{\sum_{\Theta'_b \in \Theta_{\mathcal{B}}} p(\Theta'_b)}{\sum_{\Theta'_b \in \Theta_{\mathcal{B}}}}. \tag{5.9}$$

In order to take care of the number of generated suggestions, we also introduce a measure for describing the proportion of tasks $T_{b_i} \in \Theta'_{b1}$ that are able to provide at least one item to recommend. More formally, we define the *coverage* $c(\Theta'_b)$, as follows:

$$c(\Theta'_{b1}) = \frac{|\{T_{b_i} \in \Theta'_{b1} \mid \mathcal{R}_m(T_{b_i}) > 0\}|}{|\Theta'_{b1}|}. \tag{5.10}$$

As for the precision, we compute a global coverage value by simply averaging the coverage obtained in each task-oriented session:

$$c(\Theta_{\mathcal{B}}) = \frac{\sum_{\Theta'_b \in \Theta_{\mathcal{B}}} c(\Theta'_{b1})}{\sum_{\Theta'_b \in \Theta_{\mathcal{B}}}}. \tag{5.11}$$

Finally, in our experiments we focus on evaluating the behavior of the task recommendation algorithms both on medium-size and large-size task-oriented sessions, i.e., sessions whose size is between 9 and 19 tasks, and sessions containing more than 19 tasks, respectively. These sessions globally account for about 79% of the total. We do not consider sessions shorter than 9 tasks as they do not contain enough information to fairly compute the precision as we have defined above (i.e., the second third of a session shorter than 9 contains less then, or exactly, 5 tasks).

The set of following figures shows the results, in terms of precision and coverage, obtained by the various graph-based models for both medium- and large-size sessions. The various plots report results obtained by testing different model instances, each built by using a given parameter setting, e.g., a specific minimum support or confidence for a model based on the association rule weighting scheme. Note that each model instance is characterized by a given coverage with respect to the test dataset, and this coverage decreases when a model instance becomes stricter. For example, when we increase the support parameter of the model based on the association rules we also obtain less rules, and, as a consequence, a reduced number of edges in the graph model.

In order to easily compare the different models and their instances, we thus plotted the precision obtained by each model instances as a function of the corresponding coverage. More specifically, we generated the top-1, top-3, and top-5 suggestions, respectively, for the tasks of the first thirds of these test sessions. In all these cases, the random-based approach, which is considered as our *baseline*, provides the lowest values of precision as expected. Moreover, it is worth saying that all the results presented for this baseline approach have been selected as the best of 10 runs.

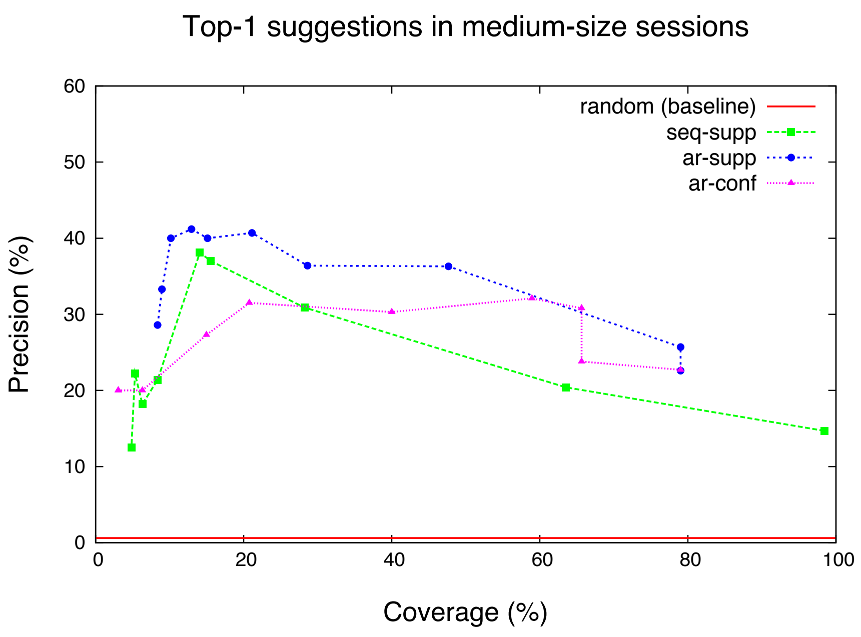


Figure 5.10: Precision vs. Coverage for top-1 recommendations in medium-size sessions.

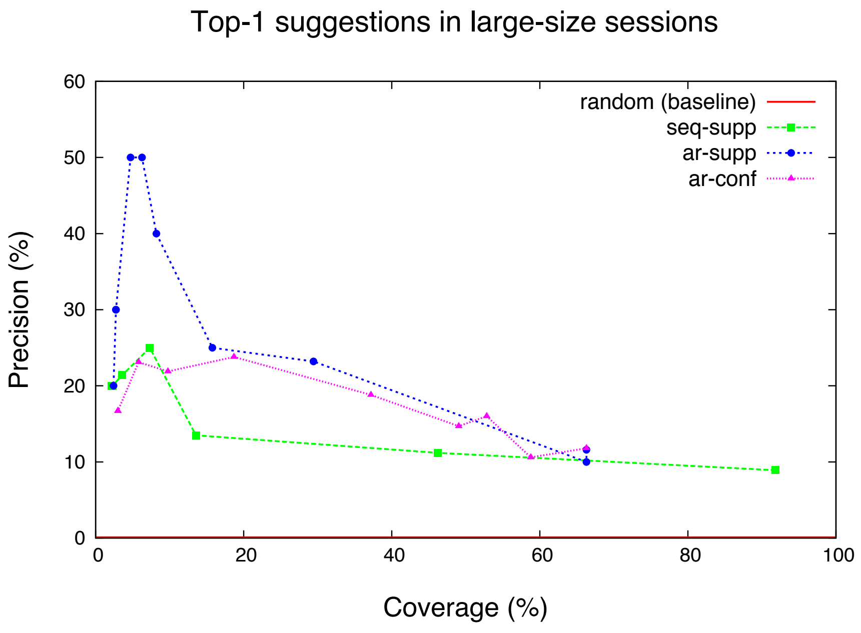


Figure 5.11: Precision vs. Coverage for top-1 recommendations in large-size sessions.

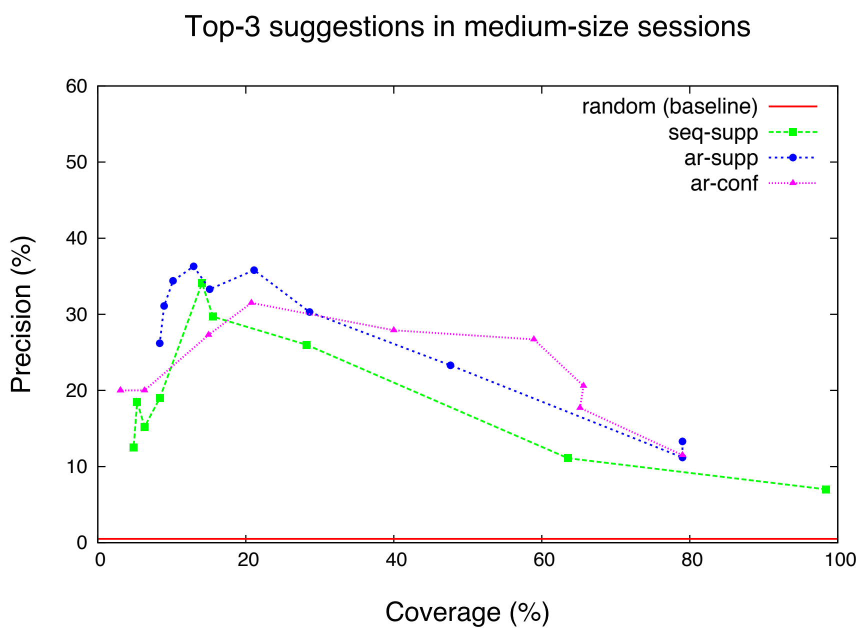


Figure 5.12: Precision vs. Coverage for top-3 recommendations in medium-size sessions.

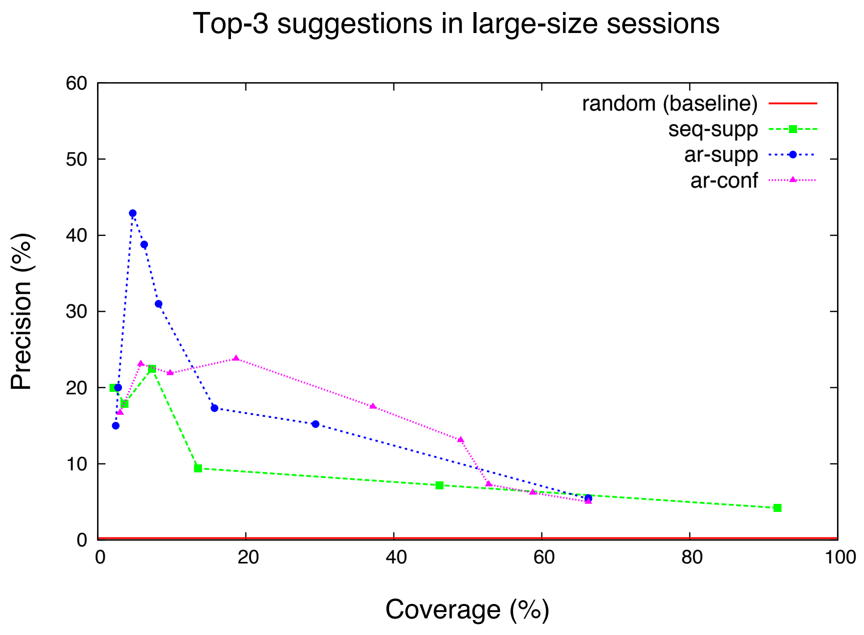


Figure 5.13: Precision vs. Coverage for top-3 recommendations in large-size sessions.

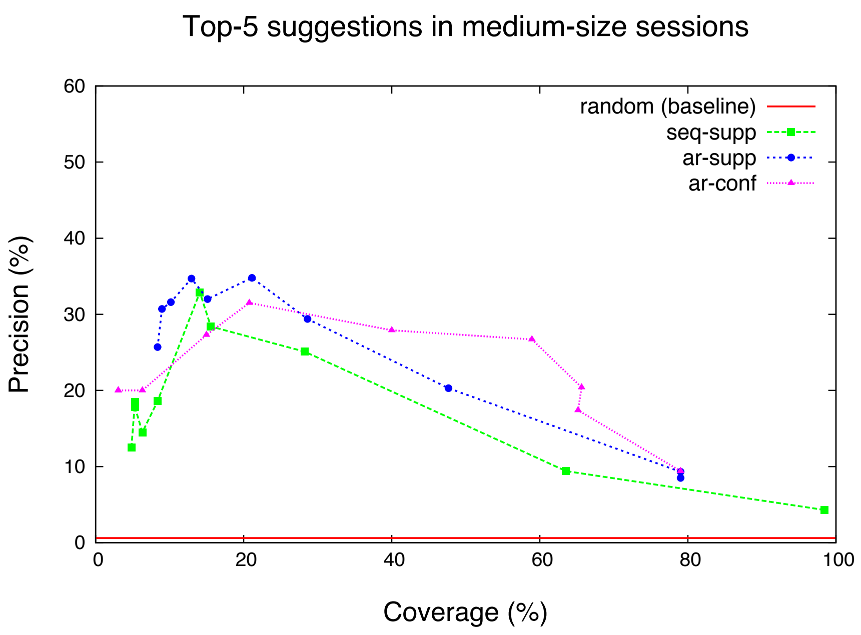


Figure 5.14: Precision vs. Coverage for top-5 recommendations in medium-size sessions.

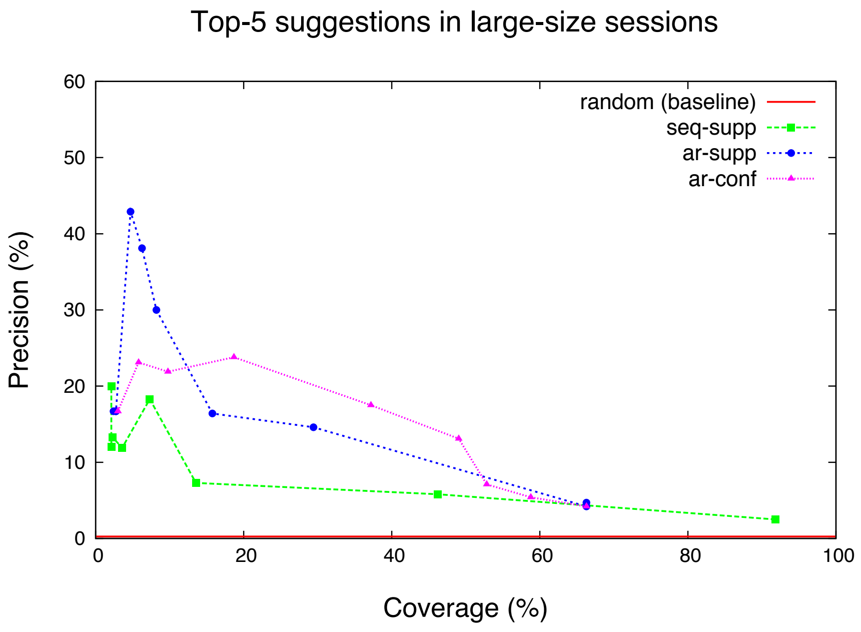


Figure 5.15: Precision vs. Coverage for top-5 recommendations in large-size sessions.

Figure 5.10 and 5.11 depict the precision of top-1 recommendations for medium-size and large-size sessions, respectively. Our proposed methods, and especially the one based on the association-rule weighting scheme described in Eq. 5.6, highlight remarkable values of precision. In medium-size sessions, we can observe a maximum accuracy of about 41.2% when the coverage is around 13.0%. However, even when the coverage reaches approximately 47.7%, precision still remains around 36.3%. In large-size sessions, precision reaches about 50.0% when the coverage is only around 6.3%. Nevertheless, precision is still above the 20%, i.e., 23.2%, whenever coverage reaches about 30.0%.

Figure 5.12 and 5.13 show the precision of top-3 recommendations for medium-size and large-size sessions, respectively. Here, for smaller values of coverage, i.e., above 25% in medium-size sessions and above 15% in large-size sessions, the association-rule weighting scheme described in Eq. 5.6 gains the highest values of precision, i.e., 36.2% and 42.9%, respectively. However, by increasing the coverage, the other association-rule weighting scheme proposed in Eq. 5.7 behaves better. Indeed, this method obtains a precision of 26.7% even when the coverage reaches 58.9% for medium-size sessions, while precision is around 17.5% when the coverage is 37.2% for large-size sessions.

Finally, Figure 5.14 and 5.15 describe the precision of top-5 recommendations for medium-size and large-size sessions, respectively. This last case is highly similar to the top-3 recommendations described above. Roughly, for smaller values of coverage, the most precise approach is the association-rule weighting scheme using Eq. 5.6, whereas for greater values of coverage the weighting scheme described in Eq. 5.7 provides better results.

5.6.3 User Study

The major drawback of the precision metric used in the evaluation of suggestions presented in the section above is that it is not able to measure the "*surprise*" factor. Informally, when a recommended task is not going to be performed by the user in the remainder of the session, the recommendation is deemed to be useless. Indeed, recommended tasks that were not actually performed in the session might still be interesting and useful for the user. Therefore, we resort to conduce a user study to further investigate the usefulness, as well as the surprise, of the recommendations generated.

We selected a pool of assessors made up of 10 people chosen amongst students and researchers in our institute. Each assessor is, obviously, highly skilled in using web search engines. We asked them to carefully evaluate suggestions looking not only at the relevance of the suggestion itself, but also if they retain the suggestion, somewhat, surprising. In other words, we wanted assessors to check wether recommendations make users think of possibly new tasks to perform after the one they were currently performing.

For a randomly chosen subset of 50 task-oriented sessions in the test set $\mathcal{B}$ (see Section 5.6.1), we generated the top-3 recommendations starting from 1/3 of the tasks present in each session. The recommended tasks have then been presented along with the actual head of the user session to at least three different assessors, who were asked to rate each recommendation as: *useless*, *useful*, *useful and surprising.*

From this process we can draw the following figures. Firstly, an average of 72.9% of recommendations have been marked either 'useful' or 'useful and surprising' by at least two out of the three assessors (the 68.3% of the recommendations have been marked at least useful by all the three assessors). Among the useful recommendations, more than a half of them (57.1%) have been marked useful and surprising by at least two assessors.

Mixing results from the previous section and the user study, one could conclude that: *(i)* our method is able to understand what possibly a user might want to do in the future and suggest it, *(ii)* in more than 50% of the cases, recommendations generated were also helpful to make users think of novel task related to the ones she was performing.

This last result shows that evaluating the quality of our task recommender system by only considering its ability of predicting tasks that actually users perform during their search sessions could be a really conservative measure of the effectiveness of our approach.

5.6.4 Anecdotal Evidences

It is worth noting that our recommendation mechanism lays at a higher level of abstraction with respect to traditional query suggestion schemes. Therefore, in order to highlight that our task recommender provides valuable suggestions, we also show some examples derived from our testing data set, which aim to highlight the two distinct abilities mentioned at the end of the previous Section.

First, the ability of our technique in "predicting" tasks that users are going to search for in the next future, is highlighted in Table 5.1. The left column of this table describes a search task performed by a user by means of a set of issued queries (appearing below the task "label"). To this end, it is worth remarking that task labels have been manually generated by authors. Automatic labeling of tasks is an interesting and challenging problem but it is out of the scope of this work. We plan to address the automatic task labeling problem in the near future. Instead, on the right column is showed a suggested task that actually the same user asks for during her next searches by issuing those queries that appear <u>underlined</u>. Moreover, since each suggested task is a *synthesized* representation of tasks performed by several users, there are other queries that are part of the suggested task and that could be also recommended. Those suggested queries might be either classified as relevant (indicated with *italic* typeface) or not relevant.

Another remarkable result that our technique is able to provide concerns with its capability of generating so-called *surprising* suggestions. This second aspect is described in Table 5.2. Here, the left column of the table still presents a performed search task, whereas the right column contains some queries that are part of a task which is suggested to the user but that she actually does not ask for during her search session. In this case, it is clearly evident that suggesting tasks containing queries such as "`baby shower games`" to a user interested in "`baby fables`" could be somehow helpful or, at least, interesting. Similarly, tasks which contain queries like "`university tuition`" is surely useful and appreciated by those users looking for "`duke university`".

In the tables below, the * symbol refers to recommendation that is either not relevant or relevant only for the current task.

Performed Task/Query	Recommended Tasks
Home Furnitures	**Home Gardening**
	cottage garden
beach house	cottage garden roses
···	*decor garden*
···	*best garden blogs*
beach house vanity	*vegetable garden ideas*
	open garden
	antiques store*
	book stores*
Kitchen Decor	**Kitchen Supplies**
dining room	stoves
···	*country stoves*
···	*country cooking stoves*
	country music gossips*
	canyon country parkway*

Table 5.1: Recommended and performed tasks.

5.7 Summary

So far, *query suggestions* provided by Web search engines aim to help users in better formulating their immediate information need. In this Chapter, we presented the first proposal for a novel *task-oriented recommendation* solution, which generates *task suggestions* on the basis of the long-term *mission* that users are expected to subsume with their queries. This ambitious goal was pursued by mining interleaved search tasks from the stream of past queries recorded in query logs and by aggregat-

Performed Task/Query	**Recommended Tasks**
Child Entertainment	**Child Games**
	baby shower games
fables	*baby horse*
···	**Child Dressing**
···	*baby gap*
baby fables	**Child Health**
	baby emotional disease
	cuddling couch picture*
	husband became parents boy*
University	**University Sports**
university	*university sports*
···	*university basketball*
···	**University Information**
duke university	*university tuition*

Table 5.2: Recommended and *surprising* tasks.

ing these tasks in similarity-based clusters. Somehow, those clusters represent the collective knowledge provided by Web search engine users.

Several graph-based models for user task search behaviors were built from the above clusters and used to provide task recommendations. The experiments, conducted by exploiting a real large-scale query log, were aimed to measure the ability of the different models devised in predicting and covering the actual long-term user search intents stored in the log. We have shown that the solution proposed generates task recommendations resulting in good precision and coverage figures, remarkably higher than the baseline consisting in the random suggestion of tasks. Moreover, even if generated task recommendations are not strictly part of actually occurring future users' requests, we have shown that they could anyway represent useful *surprising* hints.

Many research directions need to be further investigated in the future. First, different methods for aggregating user tasks have to be studied in order to address the data sparsity problem. Moreover, new objective evaluation metrics have to be studied for assessing the task recommendations provided. Finally, a rigorous user study should be accurately designed and deployed in order to evaluate the real impact of this new kind of "high-level" recommendations on the behavior of Web search engine users.

Conclusions

In this dissertation we have presented two research challenges that we claim next-generation Web search engines should deal with in order to enhance the overall user search experience. The need for addressing such challenges derives from a shift in what actually users nowadays expect from modern Web search engines whenever they submit their requests.

At the beginning, people interacting with Web search engines were mostly interested in selecting a manageable set of Web pages that hopefully contain information, which were relevant to their needs. Thereby, originally the main research issues on this domain concerned how well-known *Information Retrieval* and *Digital Libraries* approaches could be replicated on, and possibly adapted to, Web-scale retrieval systems such as Web search engines.

Today, instead, users are increasingly asking Web search engines for accomplishing their daily *tasks* in an easier way (e.g., "*planning holidays*", "*obtaining a visa*", "*organizing a birthday party*", etc.). Therefore, modern Web search engines are requested to effectively "driving" users towards their everyday activities, thus moving from Web documents retrieval tools to "run-time supports" of users' Web lives.

Both the challenges we have devised in this dissertation require to understand and extract meaningful patterns of interaction between users and Web search engines. To this end, the most valuable source of information from which such *knowledge* may be extracted is certainly represented by *query logs*, which roughly record user search activities by means of streams of timestamped events (e.g., issued queries, click actions, etc.).

First, we successfully applied query log mining techniques for discovering actual user search sessions "hidden" inside the raw stream of queries stored in Web search engine query logs, whose final aim is to perform a task, i.e., *task-oriented sessions*. Furthermore, we have formally defined the *Task-oriented Session Discovery Problem* (TSDP) as the problem of best partitioning a stream of user queries into several subsets of queries which are all related to the same search task. Roughly, this has dealt with two issues: *(i)* it has required a robust measure for evaluating the *task relatedness* between any two queries, i.e., *task-based query similarity* and *(ii)* it has needed an effective method for actually discovering task-oriented sessions using the above measure of task relatedness.

Concerning *(i)*, we have proposed and compared both *unsupervised* and *supervised* approaches for devising several task-based query similarity functions. Moreover, we have tackled *(ii)* by introducing a set of *clustering-oriented* approaches that exploit the above task-based query similarity functions, i.e., *query clustering* methods specifically tuned for actually discovering task-oriented sessions.

All the proposed solutions have been evaluated on a manually built *ground-truth*, namely a task-oriented partitioning of the queries stored in a real Web search engine log performed by human annotators.

Overall, the "take-away message" of this work is that long-term sessions stored in query logs are really composed of several, sometimes interleaved, search tasks. Moreover, such tasks may be effectively and efficiently discovered by adopting the following strategy, which in our experiments produced the best results. For each user, we firstly split the sequence of queries into subsequences consisting of queries whose gap, in terms of submission time, is less than 26 minutes[3]. Then, for queries in each shorter session, we build a graph consisting of queries connected by an edge if and only if the *similarity* between these two queries is above a threshold $\eta = 0.3$. The *task-based query similarity* results from a combination of *content-based* and *semantic-based* metrics. The first is obtained by combining two distinct metrics, i.e., the Jaccard's score on query tri-grams and the classical edit-distance between the two queries. These metrics are put together through a convex combination with coefficient equal to 0.5. The semantic-based metric is obtained by considering what we called the "*wiki-based*" distance and consists of evaluating the cosine distance between the two vectors obtained by "*wikifying*" the two queries. Instead of using a convex combination of the queries we resorted in using the following strategy. If the content-based similarity is below a threshold $\mathbf{t} = 0.5$ then we resort to use the semantic-based metric and we take the maximum of these two similarities as the weight of the edge in the graph. Finally, tasks are represented by the *connected components* of the resulting graph.

This first contribution, fully described in Chapter 4, is based on three research papers, i.e., "*Detecting Task-based Query Sessions Using Collaborative Knowledge*" [81], "*Identifying Task-based Sessions in Search Engine Query Logs*" [83], and "*Discovering User Tasks in Long-Term Web Search Engine Logs*" [82], respectively.

The second research challenge we have devised was inspired by our first provided contribution and concerned the design of a novel recommendation mechanism that goes beyond traditional query suggestion schemes.

Undoubtedly, *query suggestion* represents a valuable mechanism that most popular Web search engines today adopt for supporting their users. Usually, this mechanism works by recommending queries to users in order to help them in better specifying their actual search goals, thus for quicker satisfying their needs.

So far, the recommendations generated for a certain query are strictly related to the query itself, namely to the same *search task* that the query is likely to represent. Unfortunately, this often unfits the real search behaviors of users, which, as showed in our first contribution, instead tend to issue streams of queries possibly related to distinct *interleaved* search tasks. Moreover, the "composition" of each single search task (e.g., "*booking a flight*", "*reserving a hotel room*", "*renting a car*", etc.) usually subsumes a more *complex* task, namely a *mission*, that the user aims to accomplish

[3]This value might be different depending on the data set used.

throughout the Web search engine (e.g., "*planning a travel*"). For example, a user searching for a suitable hotel in New York, i.e., a simple task, might be recommended with information about MTA subway or Broadway shows, conjecturing that she is willing to organize her vacation, i.e., a complex task.

In order to achieve this ambitious goal, we first leveraged on the results described in Chapter 4 for mining interleaved search tasks from query logs. Instead of considering user search process on a "*query-by-query*" perspective, we looked at it from a higher level of abstraction, namely on a "*task-by-task*" perspective. To this end, we proposed a model of user search tasks for representing more complex user behaviors, while interacting with Web search engines. This model described *what* small tasks are mostly searched for and *how* users typically "combined" them in order to achieve bigger missions.

Furthermore, due to the sparseness of user search tasks, we decided to aggregate discovered tasks in similarity-based clusters that represent the collective knowledge provided by Web search engine users. Several graph-based models for user task search behaviors were built from the above clusters and used to generate novel *task recommendations*. In fact, given a portion of the query stream, the Web search engine may first recognize the actual small task behind that subsequence of queries. Eventually, it was able to generate lists of recommendations, which were not only related to the subsequence of queries that originated the suggestions but also to a complex mission, which the small task could be part of, according to our model of users' search tasks.

The experiments, conducted by exploiting a real large-scale query log, were aimed to measure the ability of the different models devised in predicting and covering the actual long-term user search intents stored in the log. We showed that the solutions proposed were able to generates task recommendations resulting in good precision and coverage figures, which were remarkably higher than the baseline consisting in the random suggestion of tasks. Moreover, even if generated task recommendations are not strictly part of actually occurring future users' requests, we have shown that they could anyway represent useful *surprising* hints.

This second contribution, fully described in Chapter 5, is based on the research paper "*Beyond Query Suggestion: Recommending Tasks to Search Engine Users*" [84].

Bibliography

[1] E. Adar. User 4xxxxx9: Anonymizing query logs. In *Proceedings of the Workshop on Query Log Analysis at the 16th International World Wide Web Conference*, WWW '07, New York City, NY, USA, 2007. ACM.

[2] E. Adar, D. Weld, B. Bershad, and S. Gribble. Why we search: Visualizing and predicting user behavior. In *Proceedings of the 16th International World Wide Web Conference*, WWW '07, pages 161–170, New York City, NY, USA, 2007. ACM.

[3] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, June 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, Waltham, MA, USA, 1994. Morgan Kaufmann Publishers.

[5] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the web. *ACM Transactions on Internet Technology (TOIT)*, 1:2–43, August 2001.

[6] V. Authors. About web analytics association, retrieved on December 2010.

[7] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. In *Proceedings of the 23rd IEEE International Conference on Data Engineering*, ICDE '07, pages 6–20, Washington, DC, USA, 2007. IEEE Computer Society.

[8] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proceedings of the 30th ACM SIGIR International Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 183–190, New York City, NY, USA, 2007. ACM.

[9] R. Baeza-Yates, A. Gionis, F. P. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. Design trade-offs for search engine caching. *ACM Transactions on the Web (TWEB)*, 2(20):1–28, October 2008.

[10] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Proceedings of the International Workshop on Clustering Information over the Web (in conjunction with EDBT 2004)*, volume 3268 of *Lecture Notes in Computer Science*, pages 588–596, Berlin, Heidelberg, 2004. Springer-Verlag.

[11] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Ranking boosting based in query clustering. In *Proceedings of the 2nd Atlantic Web Intelligence Conference (AWIC 2004)*, volume 3034 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2004. Springer-Verlag.

[12] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Improving search engines by query clustering. *Journal of the American Society for Information Science and Technology (JASIST)*, 58(12):1793–1804, October 2007.

[13] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[14] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 76–85, New York City, NY, USA, 2007. ACM.

[15] E. Balfe and B. Smyth. Improving web search through collaborative query recommendation. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, pages 268–272. IOS Press, 2004.

[16] J. Bar-Ilan. Access to query logs an academic researcher's point of view. In *Proceedings of the Query Log Analysis: Social And Technological Challenges Workshop at the 16th International World Wide Web Conference*, WWW '07, New York City, NY, USA, 2007. ACM.

[17] R. Baraglia, F. Cacheda, V. Carneiro, D. Fernandez, V. Formoso, R. Perego, and F. Silvestri. Search shortcuts: a new approach to the recommendation of queries. In *Proceedings of the 3rd ACM International Conference on Recommender Systems*, RecSys '09, pages 77–84, New York City, NY, USA, 2009. ACM.

[18] R. Baraglia, C. Castillo, D. Donato, F. M. Nardini, R. Perego, and F. Silvestri. Aging effects on query flow graphs for query suggestion. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management*, CIKM '09, pages 1947–1950, New York City, NY, USA, 2009. ACM.

[19] R. Baraglia, F. M. Nardini, C. Castillo, R. Perego, D. Donato, and F. Silvestri. The effects of time on query flow graph-based models for query suggestion. In *Adaptivity, Personalization, and Fusion of Heterogeneous Information*, RIAO

'10, pages 182–189, Paris, France, 2010. Le Centre De Hautes Etudes Internationales d'Informatique Documentaire.

[20] L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The Google cluster architecture. *IEEE micro*, 23(2):22–28, March/April 2003.

[21] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 407–416, New York City, NY, USA, 2000. ACM.

[22] S. M. Beitzel, E. C. Jensen, A. Chowdhury, O. Frieder, and D. Grossman. Temporal analysis of a very large topically categorized web query log. *Journal of the American Society for Information Science and Technology (JASIST)*, 58:166–178, January 2007.

[23] 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 *Proceedings of the 27th ACM SIGIR International Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 321–328, New York City, NY, USA, 2004. ACM.

[24] 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 *Proceedings of the 5th IEEE International Conference on Data Mining*, ICDM '05, pages 42–49, Washington, DC, USA, 2005. IEEE Computer Society.

[25] 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 Transactions on Information Systems (TOIS)*, 25:1–30, April 2007.

[26] N. J. Belkin. Interaction with texts: Information retrieval as information-seeking behavior. In *Information Retrieval*, Lecture Notes in Computer Science, pages 55–66, Berlin, Heidelberg, 1993. Springer-Verlag.

[27] T. Berners-Lee. Information management: A proposal. Technical report, CERN, Genf, 1989.

[28] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *Proceedings of the 17th ACM International Conference on Information and Knowledge Management*, CIKM '08, pages 609–618, New York City, NY, USA, 2008. ACM.

[29] P. Boldi, F. Bonchi, C. Castillo, D. Donato, and S. Vigna. Query suggestions using query-flow graphs. In *Proceedings of the Workshop on Web Search Click Data*, WSCD '09, pages 56–63, New York City, NY, USA, 2009. ACM.

[30] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998.

[31] D. Broccolo, O. Frieder, F. M. Nardini, R. Perego, and F. Silvestri. Incremental algorithms for effective and efficient query recommendation. In *Proceedings of the 17th International Conference on String Processing and Information Retrieval*, SPIRE '10, pages 13–24, Berlin, Heidelberg, 2010. Springer-Verlag.

[32] D. Broccolo, L. Marcon, F.M. Nardini, R. Perego, and F. Silvestri. An efficient algorithm to generate search shortcuts. Technical report, ISTI-CNR, Pisa, Italy, May 2010.

[33] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36:3–10, September 2002.

[34] A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan. Online expansion of rare queries for sponsored search. In *Proceedings of the 18th International World Wide Web Conference*, WWW '09, New York City, NY, USA, 2009. ACM.

[35] A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *Proceedings of the 30th ACM SIGIR International Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 231–238, New York City, NY, USA, 2007. ACM.

[36] D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 554–560, New York City, NY, USA, 2006. ACM.

[37] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th International World Wide Web Conference*, WWW7, pages 65–74, Amsterdam, The Netherlands, 1998. Elsevier Science Publishers B. V.

[38] S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *Proceedings of the 14th International World Wide Web Conference*, WWW '05, pages 2–11, New York City, NY, USA, 2005. ACM.

[39] A. Cooper. A survey of query log privacy-enhancing techniques from a policy perspective. *ACM Transactions on the Web (TWEB)*, 2(4):1–27, October 2008.

[40] S. Cucerzan and R. W. White. Query suggestion based on user landing pages. In *Proceedings of the 30th ACM SIGIR International Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 875–876, New York City, NY, USA, 2007. ACM.

[41] H. Cui, J. R. Wen, J. Y. Nie, and W. Y. Ma. Probabilistic query expansion using query logs. In *Proceedings of the 11th International World Wide Web Conference*, WWW '02, pages 325–332, New York City, NY, USA, 2002. ACM.

[42] D. Downey, S. Dumais, and E. Horvitz. Heads and tails: studies of web search with common and rare queries. In *Proceedings of the 30th ACM SIGIR International Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 847–848, New York City, NY, USA, 2007. ACM.

[43] D. Eichmann. Ethical web agents. *Computer Networks and ISDN Systems*, 28(1-2):127–136, December 1995.

[44] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '96, pages 226–231, New York City, NY, USA, 1996. ACM.

[45] 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 Transactions on Information Systems (TOIS)*, 24:51–78, January 2006.

[46] C.H. Fenichel. Online searching: Measures that discriminate among users with different types of experiences. *Journal of the American Society for Information Science (JASIS)*, 32(1):23–32, January 1981.

[47] B. M. Fonseca, P. Golgher, B. Pôssas, B. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 696–703, New York City, NY, USA, 2005. ACM.

[48] B. M. Fonseca, P. B. Golgher, E. S. de Moura, and N. Ziviani. Using association rules to discover search engines related queries. In *Proceedings of the 1st Latin American Web Congress*, LA-WEB '03, Washington, DC, USA, 2003. IEEE Computer Society.

[49] G.P.C. Fung, J.X. Yu, P.S. Yu, and H. Lu. Parameter free bursty events detection in text streams. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 181–192, Waltham, MA, USA, 2005. Morgan Kaufmann Publishers.

[50] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 6–12, 2007.

[51] D. Gayo-Avello. A survey on session detection methods in query logs and a proposal for future evaluation. *Information Sciences*, 179(12):1822–1843, May 2009.

[52] N. S. Glance. Community search assistant. In *Proceedings of the 6th ACM International Conference on Intelligent User Interfaces*, IUI '01, pages 91–96, New York City, NY, USA, 2001. ACM.

[53] S. Gordea and M. Zanker. Time filtering for better recommendations with small and sparse rating matrices. In *Proceedings of the 8th International Conference on Web Information Systems Engineering (WISE 2007)*, Lecture Notes in Computer Science, pages 171–183, Berlin, Heidelberg, 2007. Springer-Verlag.

[54] L. Gravano, V. Hatzivassiloglou, and R. Lichtenstein. Categorizing web queries according to geographical locality. In *Proceedings of the 12th ACM International Conference on Information and Knowledge Management*, CIKM '03, pages 325–333, New York City, NY, USA, 2003. ACM.

[55] D. He and A. Göker. Detecting session boundaries from web user logs. In *Proceedings of the 22nd Annual Colloquium on Information Retrieval Research*, BCS-IRSG, pages 57–66, 2000.

[56] D. He, A. Göker, and D. J. Harper. Combining evidence for automatic web session identification. *Information Processing Management (IPM)*, 38:727–742, September 2002.

[57] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22:5–53, January 2004.

[58] L. Hsieh-yee. Effects of search experience and subject knowledge on the search tactics of novice and experienced searchers. *Journal of the American Society for Information Science (JASIS)*, 44:161–174, April 1993.

[59] B. J. Jansen. *Understanding User-Web Interactions via Web Analytics*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009.

[60] B. J. Jansen and A. Spink. An analysis of web searching by european alltheweb.com users. *Information Processing Management (IPM)*, 41:361–381, March 2005.

[61] B. J. Jansen and A. Spink. How are we searching the world wide web?: a comparison of nine search engine transaction logs. *Information Processing Management (IPM)*, 42:248–263, January 2006.

[62] B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32:5–17, April 1998.

[63] B. J. Jansen, A. Spink, C. Blakely, and S. Koshman. Defining a session on web search engines: Research articles. *Journal of the American Society for Information Science and Technology (JASIST)*, 58(6):862–871, April 2007.

[64] B. J. Jansen, A. Spink, and S. Koshman. Web searcher interaction with the dogpile.com metasearch engine. *Journal of the American Society for Information Science and Technology (JASIST)*, 58:744–755, March 2007.

[65] B. J. Jansen, A. Spink, and J. Pedersen. A temporal comparison of altavista web searching: Research articles. *Journal of the American Society for Information Science and Technology (JASIST)*, 56:559–570, April 2005.

[66] A. Järvelin, A. Järvelin, and K. Järvelin. s-grams: Defining generalized n-grams for information retrieval. *Information Processing Management (IPM)*, 43(4):1005–1019, July 2007.

[67] R. Jones and K. L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM International Conference on Information and Knowledge Management*, CIKM '08, pages 699–708, New York City, NY, USA, 2008. ACM.

[68] R. Jones, R. Kumar, B. Pang, and A. Tomkins. I know what you did last summer: query logs and user privacy. In *Proceedings of the 16th ACM International Conference on Information and Knowledge Management*, CIKM '07, pages 909–914, New York City, NY, USA, 2007. ACM.

[69] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 15th International World Wide Web Conference*, WWW '06, pages 387–396, New York City, NY, USA, 2006. ACM.

[70] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[71] S. Koshman, A. Spink, and B. J. Jansen. Web searching on the vivisimo search engine. *Journal of the American Society for Information Science and Technology (JASIST)*, 57:1875–1887, December 2006.

[72] M. Koster. ALIWEB-Archie-like indexing in the Web. *Computer Networks and ISDN Systems*, 27(2):175–182, November 1994.

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

[74] R. Kumar, J. Novak, B. Pang, and A. Tomkins. On anonymizing query logs via token-based hashing. In *Proceedings of the 16th International World Wide Web Conference*, WWW '07, pages 629–638, New York City, NY, USA, 2007. ACM.

[75] T. Lau and E. Horvitz. Patterns of search: analyzing and modeling web query refinement. In *Proceedings of the 7th International Conference on User Modeling*, pages 119–128, Berlin, Heidelberg, 1999. Springer-Verlag.

[76] C. Leacock and M. Chodorow. *Combining Local Context and WordNet Similarity for Word Sense Identification*, chapter 11, pages 265–283. The MIT Press, Cambridge, MA, USA, May 1998.

[77] U. Lee, Z. Liu, and J. Cho. Automatic identification of user goals in web search. In *Proceedings of the 14th International World Wide Web Conference*, WWW '05, pages 391–400, New York City, NY, USA, 2005. ACM.

[78] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th International World Wide Web Conference*, WWW '03, pages 19–28, New York City, NY, USA, 2003. ACM.

[79] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th ACM International Conference on Systems Documentation*, SIGDOC '86, pages 24–26, New York City, NY, USA, 1986. ACM.

[80] K. W. T. Leung, W. Ng, and D. L. Lee. Personalized concept-based clustering of search engine queries. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 20(11):1505–1518, November 2008.

[81] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Detecting task-based query sessions using collaborative knowledge. In *International Workshop on Intelligent Web Interaction in conjunction with the IEEE/WIC/ACM International Conferences on Web Intelligence*, IWI '10, pages 128–131, Washington, DC, USA, 2010. IEEE Computer Society.

[82] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Discovering User Tasks in Long-Term Web Search Engine Logs. *ACM Transactions on Information Systems (TOIS) (under review)*, 2011.

[83] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Identifying task-based sessions in search engine query logs. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 277–286, New York City, NY, USA, 2011. ACM.

[84] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Beyond query suggestion: Recommending tasks to search engine users. WSDM '12 (under review), New York City, NY, USA, 2012.

[85] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[86] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York City, NY, USA, 2008.

[87] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*, volume 59. The MIT Press, Cambridge, MA, USA, 1999.

[88] E. P. Markatos. On caching search engine query results. *Computer Communications*, 24(2):137–143, February 2001.

[89] M. Mat-Hassan and M. Levene. Associating search and navigation behavior through log analysis: Research articles. *Journal of the American Society for Information Science and Technology (JASIST)*, 56:913–934, July 2005.

[90] O.A. McBryan. GENVL and WWWW: Tools for Taming the Web. In *Proceedings of the 1st International World Wide Web Conference*, volume 341 of *WWW1*, pages 79–90, Amsterdam, The Netherlands, 1994. Elsevier Science Publishers B. V.

[91] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *Proceeding of the 17th Conference on Information and Knowledge Management*, CIKM '08, pages 469–478, New York City, NY, USA, 2008. ACM.

[92] D. Milne and I. H. Witten. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *Proceedings of the 22nd Conference on Artificial Intelligence*, AAAI '08, pages 25–30, Menlo Park, CA, USA, 2008. AAAI Press.

[93] H. C. Ozmutlu and F. Çavdur. Application of automatic topic identification on excite web search engine data logs. *Information Processing Management (IPM)*, 41(5):1243–1262, September 2005.

[94] H. C. Ozmutlu, A. Spink, and S. Ozmutlu. Analysis of large data logs: an application of poisson sampling on excite web queries. *Information Processing Management (IPM)*, 38:473–490, July 2002.

[95] S. Ozmutlu, A. Spink, and H. C. Ozmutlu. A day in the life of web searching: an exploratory study. *Information Processing Management (IPM)*, 40:319–345, March 2004.

[96] G. Pant. Deriving link-context from html tag tree. In *Proceedings of the 8th ACM SIGMOD Workshop on Research issues in Data Mining and Knowledge Discovery*, DMKD '03, pages 49–55, New York City, NY, USA, 2003. ACM.

[97] G. Pant and P. Srinivasan. Crawling the web. In *Web Dynamics: Adapting to Change in Content, Size, Topology and Use. Edited by M. Levene and A. Poulovassilis*, pages 153–178, Berlin, Heidelberg, 2004. Springer-Verlag.

[98] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, InfoScale '06, New York City, NY, USA, 2006. ACM.

[99] K. A. R. L. Pearson. The problem of the random walk. *Nature*, 72(1867):342, August 1905.

[100] R. Pemantle. A survey of random processes with reinforcement, February 2007.

[101] B. Poblete, M. Spiliopoulou, and R. Baeza-Yates. Privacy-preserving query log mining for business confidentiality protection. *ACM Transactions on the Web (TWEB)*, 4(3):10:1–10:26, July 2010.

[102] M. F. Porter. *An Algorithm for Suffix Stripping*, pages 313–316. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.

[103] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[104] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 19(1):17–30, January/February 1989.

[105] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *Proceedings of the KDD Cup Workshop at the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '05, pages 239–248, New York City, NY, USA, 2005. ACM.

[106] V. V. Raghavan and H. Sever. On the reuse of past optimal queries. In *Proceedings of the 18th ACM SIGIR International Conference on Research and Development in Information Retrieval*, SIGIR '95, pages 344–350, New York City, NY, USA, 1995. ACM.

[107] W. Reed. The pareto, zipf and other power laws. *Economics Letters*, 74(1):15–19, December 2001.

[108] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, IJCAI, pages 448–453, 1995.

[109] M. Richardson. Learning about the world through long-term query logs. *ACM Transactions on the Web (TWEB)*, 2(4):1–27, October 2008.

[110] S. E. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *TREC*, 1999.

[111] D. E. Rose and D. Levinson. Understanding user goals in web search. In *Proceedings of the 13th International World Wide Web Conference*, WWW '04, pages 13–19, New York City, NY, USA, 2004. ACM.

[112] G. Salton and M. J. Mcgill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York City, NY, USA, 1986.

[113] N. Seco and N. Cardoso. Detecting user sessions in the tumba! Web Log. Technical report, Faculdade de Ciências da Universidade de Lisboa, March 2006.

[114] D. Shen, R. Pan, J. T. Sun, J. J. Pan, K. Wu, J. Yin, and Q. Yang. Q2C@UST: our winning solution to query classification in KDDCUP 2005. *SIGKDD Explorations Newsletter*, 7:100–110, December 2005.

[115] X. Shen, B. Tan, and C. Zhai. Implicit user modeling for personalized search. In *Proceeding of the 14th Conference on Information and Knowledge Management*, CIKM '05, pages 824–831, New York City, NY, USA, 2005. ACM.

[116] X. Shi and C. C. Yang. Mining related queries from search engine query logs. In *Proceedings of the 15th International World Wide Web Conference*, WWW '06, pages 943–944, New York City, NY, USA, 2006. ACM.

[117] S. Siegfried, M.J. Bates, and D.N. Wilde. A profile of end-user searching behavior by humanities scholars: The Getty Online Searching Project Report No. 2. *Journal of the American Society for Information Science (JASIS)*, 44(5):273–291, June 1993.

[118] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33:6–12, September 1999.

[119] F. Silvestri, R. Baraglia, C. Lucchese, S. Orlando, and R. Perego. (Query) History Teaches Everything, Including the Future. In *Proceedings of the 6st Latin American Web Congress*, LA-WEB '08, pages 12–22, Washington, DC, USA, 2008. IEEE Computer Society.

[120] Fabrizio Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 1(1-2):1–174, 2010.

[121] Y. Song and L.W. He. Optimal rare query suggestion with implicit user feedback. In *Proceedings of the 19th International World Wide Web Conference*, WWW '10, pages 901–910, New York City, NY, USA, 2010. ACM.

[122] A. Spink, B. J. Jansen, D. Wolfram, and T. Saracevic. From e-sex to e-commerce: Web search changes. *IEEE Computer*, 35:107–109, March 2002.

[123] A. Spink, H. C. Ozmutlu, and D. P. Lorence. Web searching for sexual information: an exploratory study. *Information Processing and Management (IPM)*, 40:113–123, January 2004.

[124] A. Spink, M. Park, B. J. Jansen, and J. Pedersen. Multitasking during web search sessions. *Information Processing and Management (IPM)*, 42(1):264–275, January 2006.

[125] A. Spink and T. Saracevic. Interaction in information retrieval: selection and effectiveness of search terms. *Journal of the American Society for Information Science (JASIS)*, 48(8):741–761, August 1997.

[126] A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the web: the public and their queries. *Journal of the American Society for Information Science and Technology (JASIST)*, 52:226–234, February 2001.

[127] J. Srivastava, R. Cooley, M. Deshpande, and P. N. Tan. Web usage mining: discovery and applications of usage patterns from web data. *SIGKDD Explorations Newsletter*, 1:12–23, January 2000.

[128] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques, 2000.

[129] J. Surowiecki. *The Wisdom of Crowds.* Anchor, New York City, NY, USA, 2005.

[130] L. Sweeney. k-ANONYMITY: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, October 2002.

[131] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, Boston, MA, USA, May 2005.

[132] G. Tolomei. Search the web x.0: mining and recommending web-mediated processes. In *Proceedings of the 3rd Conference on Recommender Systems*, RecSys '09, pages 417–420, New York City, NY, USA, 2009. ACM.

[133] G. Tolomei, S. Orlando, and F. Silvestri. Towards a task-based search and recommender systems. In *Proceedings of the 26th IEEE International Conference on Data Engineering*, ICDE '10 Workshops, pages 333–336, Washington, DC, USA, 2010. IEEE Computer Society.

[134] V.S. Verykios, E. Bertino, I.N. Fovino, L.P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *ACM SIGMOD Record*, 33(1):50–57, March 2004.

[135] M. Vlachos, S. S. Kozat, and P. S. Yu. Optimal distance bounds for fast search on compressed time-series query logs. *ACM Transactions on the Web (TWEB)*, 4:6:1–6:28, April 2010.

[136] M. Vlachos, C. Meek, Z. Vagena, and D. Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *Proceedings of the International Conference on Management of Data*, SIGMOD '04, pages 131–142, New York City, NY, USA, 2004. ACM.

[137] M. Vlachos, P.S. Yu, V. Castelli, and C. Meek. Structural periodic measures for time-series data. *Data Mining and Knowledge Discovery*, 12(1):1–28, January 2006.

[138] 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 Explorations Newsletter*, 7:117–122, December 2005.

[139] J. R. Wen, J. Y. Nie, and H. Zhang. Query clustering using user logs. *ACM Transactions on Information Systems (TOIS)*, 20(1):59–81, January 2002.

[140] R. W. White, M. Bilenko, and S. Cucerzan. Leveraging popular destinations to enhance web search interaction. *ACM Transactions on the Web (TWEB)*, 2(3):16:1–16:30, July 2008.

[141] 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 '07, New York City, NY, USA, 2007. ACM.

[142] J. L. Xu and A. Spink. Web research: The excite study. In *Proceedings of the World Conference on the WWW and Internet*, WebNet '00, pages 581–585. AACE, 2000.

[143] S. Yizhou, X. Kunqing, L. Ning, Y. Shuicheng, Z. Benyu, and C. Zheng. Causal relation of queries from temporal logs. In *Proceedings of the 16th International World Wide Web Conference*, WWW '07, pages 1141–1142, New York City, NY, USA, 2007. ACM.

[144] O. Zaïane and A. Strilets. Finding similar queries to satisfy searches based on query traces. *Advances in Object-Oriented Information Systems*, 2426:349–359, September 2002.

[145] Y. Zhang, B. J. Jansen, and A. Spink. Time series analysis of a web search engine transaction log. *Information Processing and Management (IPM)*, 45:230–245, March 2009.

[146] Y. Zhang and A. Moffat. Some observations on user search behavior. In *Proceedings of the 11th Australian Document Computing Symposium*. Brisbane, Australia, 2006.

[147] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *Proceedings of the 15th International World Wide Web Conference*, WWW '06, pages 1039–1040, New York City, NY, USA, 2006. ACM.

[148] Q. Zhao, S.C.H. Hoi, T.Y. Liu, S.S. Bhowmick, M.R. Lyu, and W.Y. Ma. Time-dependent semantic similarity measure of queries using historical click-through data. In *Proceedings of the 15th International World Wide Web Conference*, WWW '06, pages 543–552, New York City, NY, USA, 2006. ACM.

[149] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceeding of the 11th Conference on Information and Knowledge Management*, CIKM '02, pages 515–524, New York City, NY, USA, 2002. ACM.

[150] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, June 2004.

[151] George K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Boston, MA, USA, 1949.