Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

On Two Web IR Boosting Tools: Clustering and Ranking

Antonino Gullì

Supervisor
Prof. Paolo Ferragina

Referee
Prof. Dino Pedreschi

Referee
Prof. Francesco Romani

May 6, 2006

Abstract

This thesis investigates several research problems which arise in modern Web Information Retrieval (WebIR). The Holy Grail of modern WebIR is to find a way to organize and to rank results so that the most "relevant" come first. The first break-through technique was the exploitation of the link structure of the Web graph in order to rank the result pages, using the well-known Pagerank and Hits algorithms. This link-analysis approaches have been improved and extended, but yet they seem to be insufficient inadequate to provide satisfying search experience.

In many situations a flat list of ten search results is not enough, and the users might desire to have a larger number of search results grouped on-the-fly in folders of similar topics. In addition, the folders should be annotated with meaningful labels for rapid identification of the desired group of results. In other situations, users may have different search goals even when they express them with the same query. In this case the search results should be personalized according to the users' on-line activities. In order to address this need, we will discuss the algorithmic ideas behind SnakeT, a hierarchical clustering meta-search engine which personalizes searches according to the clusters selected on-the-fly by users.

There are also situations where users might desire to access fresh information. In these cases, traditional link analysis could not be suitable. In fact, it is possible that there is not enough time to have many links pointing to a recently produced piece of information. In order to address this necessity, we will discuss the algorithmic and numerical ideas behind a new ranking algorithm suitable for ranking fresh type of information, such as news articles.

When link analysis suffices to produce good quality search results, the huge amount of Web information asks for fast ranking methodologies. We will discuss numerical methodologies for accelerating the eingenvector-like computation, commonly used by link analysis.

An important result of this thesis is that we show how to address the above predominant issues of Web Information Retrieval by using clustering and ranking methodologies. We will demonstrate that both clustering and ranking have a *mutual reinforcement* property that has not yet been studied intensively. This property can be exploited to boost the precision of both the methodologies.

Water, water, every where,
And all the boards did shrink;
Water, water, every where,
Nor any drop to drink.

Coleridge's The Rime of the Ancient Mariner.

to Leonardo, Lorenzo and Francesca

Acknowledgements

First of all I want to thank my supervisor, Paolo Ferragina. I feel grateful to Paolo for all our motivating discussions on several Web Information Retrieval (WebIR) problems. I enjoyed both his strong algorithmic knowledge and his suggestions to orientate my research towards the Clustering and Ranking problems, which are discussed in this Ph.D.. My knowledge of WebIR is more complete thanks to our inspiring discussions on these topics. Where do you find the time to do all these things, Paolo? Life has it ups and downs, and I would like to thank you for the opportunities you gave to me.

Gianna Del Corso and Francesco Romani deserve my gratitude for teaching me how to apply the methodologies of Numerical Analysis to WebIR. Our collaboration has been really fruitful, since it allowed me to convert many real problems in formal numerical models. Francesco is a night worker and our best ideas came in early morning emails. Gianna is a splendid person and i would like to thank her for advising me on many things.

I learned my useful insights from the discussions with my co-authors, Gianna Del Corso, Paolo Ferragina, Francesco Romani, and Alessio Signorini, during the submission of scientific papers to journals and conferences. Thank you all for your patience.

In addition, I want to thank Fabrizio Luccio, Linda Pagli, Roberto Grossi, and Giovanni Manzini for the prolific working environment they created within the Algorithmic Research Group at University of Pisa. I want to thank Andrea Maggiolo-Schettini for his great advices.

Then, I want to thank Apostolos Gerasoulis Executive VP, Search Technology with Ask.com. I feel grateful to Apostolos for being interested in the research themes presented in this Thesis, and in my past working experience. Apostolos offered me the opportunity to join Ask.com Research, and to have a leading role in one of the top four algorithmic search engines in the world. In addition, I want to thank all the guys of Ask.com Research and Ask.com R&D Europe for helping me in writing a new and exciting page in the book of my life. A particular thank to Tony Savona, Monica Mori, Elizabeth Hawkins, Luigi Carnevale and Filippo Tanganelli, who are sharing this experience from the very beginning. I want to thank Ask.com management: Steve Berkowitz, Tuoc Luong, Tao Yang, and Jim Lanzone for giving me the opportunity of working for Ask. A special thank to Kelly Mullane, Eve Chaurand-Fraser and Glen Sunnergren.

A special thank you to my internal referees Francesco Romani and Dino Pedreschi, and to my external referees Andrei Broder, VP of emerging search technology with Yahoo! Research, and Tomasz Imielinski VP of search technology with Ask Jeeves. They provided me with helpful comments on my Thesis. I have learned a lot from them.

Finally, a special thought to my family, for their love and support. In particular, to my wife, Francesca, for following me in any new idea which I think about. Leonardo and Lorenzo, you are my future and the best things i have made in my life. *Io sono sempre con voi.*

Contents

Introduction

What do you know and how do you know it? Ayn Rand

The Web is a rapidly expanding hyper-linked collection of unstructured information. The lack of rigid structure and the enormous volume of information pose tremendous challenges to commercial search engines, and create several intriguing themes of research for the academic community. The number of search engines' users is also increasing. [231] reports that in the U.S. the average daily use of search engines jumped from 49.3 million users to 60.7 million users, from September 2004 to September 2005 - an increase of 23%.

In order to find useful information, two paradigms are well-established in traditional *Information Retrieval*. *Searching* is a discovery paradigm which is useful for a user who knows precisely what to look for, while *browsing* is a paradigm useful for a user who is either unfamiliar with the content of the data collection or who has casual knowledge of the jargon used in a particular discipline. Browsing and searching complement each other, and they are most effective when used together [143].

The goal of a modern *Web search engine* is to retrieve documents considered "relevant" to a user query from a given collection. Nowadays, a user query is modeled as a set of keywords extracted from a large dictionary of words; a document is typically a Web page, `pdf`, postscript, `doc` file, or whatever file that can be parsed into a set of tokens. *Global* search engines serve as de facto Internet portals, *local* search engines are embedded in numerous individual Web sites, and browsing is the most common activity on the Web, due to the hyper-linked structure that provides access to a large quantity of information in a restricted space. In addition to traditional Information Retrieval issues, we may identify at least five specific obstacles which Web searching and browsing must overcome.

The first key difficulty in solving the above retrieval problem relies on the characterization of the adjective "*relevant*". Modern search engines have spent a lot of effort in ranking Web objects, providing valuable access to the information contained on the Internet. A breakthrough piece of technology has been introduced by adopting social network theories and link analysis, such as Pagerank and Hits, largely adopted by modern search engines. Nevertheless, in many situations traditional link analysis is not the perfect solution because the problem of ranking search results is inherently complex. One aspect of this complexity derives from the different types of queries submitted to search engines. Narrow topic queries are queries for which very few resources exist on the Web, and which present a "needle in the haystack" challenge for search engines. On the other hand, broad topic queries pertain to topics for which there is an abundance of information on the Web, sometimes as many as millions of relevant resources (with varying degrees of relevance). Broad topic queries are the most difficult to solve since the vast majority of users show *poor patience*: they commonly browse through the first ten results (i.e. one screen) hoping to find there the "right" document for their query. In order to find relevant results, another aspect to take into consideration is the *spamming phenomenon*. With the pervasive use of the Web, it is crucial for business sites to be ranked highly by the major search engines. There are quite a few companies who sell this kind of expertise (also known as "search engine optimization") and actively research ranking algorithms and heuristics of search engines, and know how many keywords to place (and where) in a Web page so as to improve the page's ranking (which has a direct impact on the page's

visibility). An interesting book to read on the subject is [109]. Search engine optimization is a legitimate activity, while spamming is a malicious one. Search engines and spammers are engaged in an endless fight, from one side, to improve their ranking algorithms and, from the other side, to exploit them. Unfortunately, it is very difficult to distinguish between optimization and spamming. For instance, one of the difficulties of the fight against spam is that there are perfectly legitimate optimizations (e.g. using synonyms as keywords) that might trigger anti-spam defenses.

The second key difficulty is related to the *huge quantity* of information available. It goes without saying that the quality of the search engines is influenced by the completeness and freshness of the index which should have few outdated pages and broken hyper-links. Unfortunately, the explosion of digital information available on the Web is making it impossible, on the one hand, to index the whole set of existing Web pages and, on the other hand, to restrict the number of relevant documents to return to the user, since this number grows in proportion to the Web size. The first difficulty can be partially alleviated by the use of *meta-search engines* [64] which exploit a pool of individual search engines to collect a larger set of relevant answers for a given user query [15]. However the use of *multiple sources*, each one exploiting its own ranking policy, makes the retrieval of relevant documents even harder to deal with because of the necessity to *merge* multiple ranked lists into one unique ranked list [69].

The third key difficulty is that the relevance of a document is a *subjective and time-varying concept*. In fact, the same set of keywords may abstract different user needs that may also vary over time according to the context in which the user is formulating his/her own query. As an example, for the query "search engine" a researcher may be interested in finding scientific papers, whereas a student may be interested into easy-to-read descriptions; nonetheless, the same researcher might be interested in "popular works" in the case that (s)he has to prepare a presentation for a high-school.

The fourth key difficulty is that users often aim to *find fresh information*. This is particularly true in the presence of unanticipated events such as a "Tsunami", or the death of a celebrity, or a terrorist attack. In this case, interest is focused on news articles which cannot simply be ranked by adopting link analysis techniques. In fact, when a news article is posted, it is a fresh type of information with almost no hyper-link pointing to it.

The fifth key difficulty is that often search requests *cannot be expressed clearly in just a few keywords*. This is due to the phenomena of "synonymy" (several words may correspond to the same concept) and "polysemy" (one word may have several different meanings). For instance, "python" is a polysemic word having at least three different meaning ('the reptile, the "monty python' show, and the programming language), while "car" and "automobile" are two synonyms. As another example, one user can could be interested in the helicopter "apache", another in the native Americans "apache" population, and yet another in the "apache" web server.

Modern WebIR needs to
1. discover relevant documents
2. handle an huge quantity of information
3. address subjective and time-varying search needs
4. find fresh information
5. deal with poor quality queries

Figure 1: Some difficulties in solving the retrieval problem

Despite all of these difficulties, the advent of the Web has turned IR into a key enabling technology. Nowadays, Web search engines like Ask Jeeves [213], Google [223], Yahoo [237], Teoma [235], AllTheWeb [209] and Altavista [210] are tools used by more than 400 million people daily. Still a lot of work needs to be done. All the above issues impose the creation of smart software systems which guide the users and make their search activities simple and intuitive. Personalization (e.g. Eurekster [221], Yahoo! [193] and Google [192]), collaborative filtering (e.g. Amazon [211]), query-terms suggestion and question answering (e.g. Ask Jeeves), user behaviour profiling (e.g. Epinions [220]), and fast and efficient crawling techniques (e.g. Archive [212]) are just some of the

tools provided by modern search engines, now at the third generation, to ultimately match the "user needs" as best possible. From an academic point of view, proposals for new algorithmic methodologies to efficiently and effectively access information and mathematical models to assess the validity of these methodologies, are necessary.

During our Doctoral Thesis, we will show that many of the above difficulties could be overcome with the use of *smart tools for "automatic knowledge extraction"* from the set of returned answers. In fact, in many situations a flat list of ranked results is not enough and these tools can dynamically post-process the retrieved documents, analyzing them, and then suggesting "on the fly" possible query refinements for focusing the query. For example, a query for "iraq" should suggest refinements such as "oil food program", "bbc news conflict", "analysis middle east", while a query for "bush" should suggest refinements such as "white house" as well as "43rd president". We will also show that some of the above difficulties could be overcome adopting new methodologies for fast Web rank computation and integrating link analysis with models suitable for ranking fresh information.

Next generation WebIR tools

We believe that the next generation of WebIR tools should take advantage of two technologies "Clustering" and "Ranking", which show a *mutual reinforcement* property.

Clustering is a process which receives a set of documents as input, and groups them based on their similarity. It is distinct from classification, in which an a priori taxonomy of categories is available beforehand, and where the documents are placed in their proper category. Clustering, in contrast, is a process where the categories are part of the (discovered) output, rather than part of the input. Clustering is a useful post-processing technique applied in the presence of broad queries. In fact, when no pre-imposed classification scheme is available (like in the heterogenous Web), automatic clustering is crucial for organizing a huge amount of answers in a collection of *browsable and dynamically organized hierarchies*. The browsable nature of the hierarchy helps in identifying time-varying interests for a given subject. This process has also another important advantage: it can improve the search experience by labeling the clusters with meaningful sentences. These meaningful sentences are an interesting alternative to the flat list of search results currently returned by the most important search engines. In short, clustering can act as a *booster* which helps the user in solving on-the-fly the polysemy and synonymy problems and extracts hidden knowledge from retrieved texts. Browsing can be used to discover topics to search for, and search results can be organized for a more refined browsing session. The first goal of our Doctoral Thesis, has been to investigate the use of Web page clustering as an innovative WebIR tool which can help users to search the Web. In particular, we will focus on Web snippet Clustering, a particular kind of clustering of search engines' answers which aims to group Web pages even if just a synthesis (the snippet) of them is available. We remark that some industrial solutions for Web clustering exist, namely Vivisimo, but there is no public information available regarding their clustering algorithms.

Ranking is the process which estimates the quality of a set of results retrieved by a search engine. Traditional IR has developed boolean, probabilistic, or vector-space models, aiming to rank the documents based on the content of the collection. Modern WebIR exploits the link structure of the Web. Note that links provide a positive critical assessment of a Web page's content which originates from outside of the control of the page's author (as opposed to assessments based on Web page's textual content, which is completely under the control of Web page's author). This makes the information extracted from informative links less vulnerable to manipulative techniques such as spamming. It goes without saying that one can use the large quantity of academic publications about Web Ranking available in literature [27, 124, 138, 19]. Nevertheless, as we already pointed out, many aspects remain to be investigated. The other goal of our Doctoral Thesis is to study and design new methodologies for fast Web rank computation and to integrate link analysis with models suitable for ranking fresh information.

We remark, that Clustering and Ranking methodologies are closely related and that this relationship has never been deeply investigated before in literature, as in this present Thesis. We claim

that Clustering and Ranking are WebIR tools linked by a mutual reinforcement relationship. In one direction, a good ranking strategy can provide a valuable base of information for clustering in a dynamically organized hierarchy. In the opposite direction, a good cluster strategy can provide a valuable base for altering the rank of the retrieved answer set, by emphasizing hidden knowledge meanings or solving synonymy and polysemy problems which are not captured by traditional text or link based analysis. In addition, clustering algorithms can be used in order to extract, on the user behalf, a knowledge which goes behind the traditional flat list of about ten results.

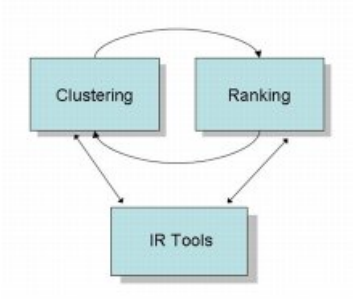


Figure 2: Smart WebIR tools exploit the mutual reinforcement relation between clustering and ranking methodologies.

Organization of the Thesis

The rest of this Thesis is organized as follows. Chapters 1-3 provide an overview of the themes investigated in this Thesis. Namely, Chapter 1 is an introduction to Information Retrieval and Web Information Retrieval, Chapter 2 presents the most important clustering algorithms given in literature, and Chapter 3 discusses the most important ranking algorithms.

The Holy Grail of modern WebIR is to find a way to organize and to rank results so that the most "relevant" results come beforehand. When a flat list of search results is not enough, users might desire to have search results grouped on-the-fly for similar topics into labeled folders and to personalize the list of results according to their on-line activities. In this way, the results are not biased towards the most popular meaning of a query on the Internet, but they are focused on the topics of interest for the users. This is the subject of Chapter 4, where we will discuss SnakeT our personalized hierarchical clustering meta search engine. SnakeT is a complete solution that offers both hierarchical clustering and folder labeling with variable-length sentences. SnakeT also offers a peculiar form of search results' personalization that is fully adaptive, preserves privacy, scalable, and non intrusive for the underlying search engines. In addition, SnakeT is the first example of a software system that exploits the mutual relationship between ranking and clustering. In one direction, the better the rank provided by the underlying search engines is, the better could be the quality of the clusters produced by our search engine. In the other direction, the better the quality of the clusters is, the better the results the rank personalization produced by SnakeT are. Vivisimo and Microsoft have developed commercial software with many similarities to the line of research of SnakeT. There is no information about the algorithms used by Vivisimo and the prototype discussed in [187] is no longer available since Microsoft is now commercializing a product deriving from this prototype. In addition, our proposal improves those solution since we suggest using hierarchical clustering in order to let users personalize their results on-the-fly.

When the freshness of information is the major goal, it is possible that there will not be enough time to have many links pointing to a recently produced piece of news. This is the subject of Chapter 5, where we will discuss our methodology for ranking a stream of news articles gathered by an experimental news search engine that we have developed. This methodology can be used for any stream of information which evolves over time, such as blogs and publications. We propose a framework for news ranking which models: (1) the generation of a stream of news articles; (2)

news article clustering by topic; (3) the evolution of a news story over time. The proposed ranking algorithm rates news information, seeking out the most authoritative sources and identifying the most current events in the particular category which the news article belongs to. Each of the aforementioned measures take into account time parameters and do not require a predefined sliding observation window over the stream of news. VIVISIMO, GOOGLE and ASK JEEVES have developed commercial software with many similarities to the news ranking line of research discussed in this Thesis, but there is no public information available about the industrial methodologies they are adopting. We remark that a patent [227] has been independently filed by GOOGLE for ranking these types of fresh content, at the same time as our publications were published.

When link analysis suffices to produce good quality search results, the huge amount of Web information [96] asks for fast ranking methodologies. This is the subject of Chapter 6 where we will discuss an approach for accelerating the computation of Pagerank, one of the well-know ranking algorithms from an academic point of view. Pagerank has also become a useful paradigm of computation in many Web search algorithms, such as spam detection or trust networks, where the input graphs have different levels of granularity (HostRank) or different link weight assignments (internal, external, etc.). For each algorithm, the critical computation is a Pagerank-like vector of interest. Thus, methods for accelerating these kinds of algorithms are very important. Using numerical techniques, we will demonstrate that: (i) this problem may be transformed into an equivalent linear system of equations, where the coefficient matrix is as sparse as the Web graph itself; (ii) we can exploit the structure of the sparse coefficient matrix and its reducibility; (iii) we can reorder the coefficient matrix so that many numerical iterative methods for a linear system solution may benefit from the increased data locality and may reduce the number of iterations required. A similar line of research for fast ranking has been independently developed by YAHOO! [87, 86] at the same time as our publications were published.

Modern WebIR Needs	**A possible solution**	**Chapter**	**My publications**
Group search results by similar topics	Web snippet clustering	4	[73, 74, 75, 95]
Personalize search results according to the users activities	Personalized Web snippet clustering	4	[76]
Access to fresh pieces of information	Ranking a stream of news articles	5	[53, 94]
Select the most relevant results, from a huge volume of Web objects	Fast Web ranking algorithm	6	[51, 54, 52, 96]

Figure 3: A summary of the results contained in this Doctoral Thesis

Chapter 1

An Overview of Web Information Retrieval

It is your mind that creates this world, Siddhartha Buddha

Abstract

In this Chapter we will review some elements of Information Retrieval (IR). In particular, in Section 1.1 we will focus our attention on the most important models defined in classical Information Retrieval, while in Section 1.2 we will introduce the key aspects of modern WebIR and discuss why it is much more complex than traditional Information Retrieval. A large part of this complexity is due to the size of the Web, which will be discussed in Section 1.3.

Information Retrieval (IR) is not a recent discipline. In the 1960's, Gerard Salton developed SMART, an experimental information retrieval system. SMART has been a test–bed for algorithms which perform automatic indexing and retrieval of full–text documents. A lot of theoretical models from natural language processing, statistical text analysis, word–stemming, stoplists, and information theory has been experimented in the system. He showed that the traditional task of IR was to retrieve the most "relevant" set of documents from a collection of documents, for a given query. He also assumed, as usual in traditional IR, that the collection was controlled, in the sense that no document was created for spamming – created with the intent of being selected for un–related queries, relatively small and almost never changing.

In 1995 everything changed with the creation of the Web. Web objects are heterogenous since they can be of different types: Web pages, audio, video, news articles, usenet, blogs, to name a few. Web objects are un–controlled collections, in the sense that billions of authors create them independently and, very often, they create them for spamming. In addition, Web objects are the largest collection of information ever created by humans, and this collection changes continuously when new objects are created and old ones removed. In order to adapt to this changed scenario, a new discipline has been created: Web Information Retrieval. It uses some concepts of traditional IR, and introduces many innovative ones. In this Chapter, we will discuss IR and Web IR pointing out the difference between these two disciplines.

1.1 An overview of traditional IR

In traditional IR documents have been represented in the so called vector space model [183]. Documents are *tokenized* in words, some terms are possibly filtered against a static defined *stop–list*, sometimes they are *stemmed* to extract a canonical form, and represented as a vector in Euclidean space. Each canonical token represents an axis in this space, and each document is a vector in the space. If the term t appears $n(t, d)$ times in document d, then the t–th coordinate of d is just $n(t, d)$.

Sometimes it could be useful to normalize the length of a document to the unit using one amongst the well-know norms $||d||_1 = \sum_t n(t,d)$, $||d||_2 = \sqrt{\sum_t n(t,d)^2}$, or $||d||_\infty = \max_t n(t,d)$. Following the notation of [37], we define the "term frequency" as $TF(t) = \frac{n(t,d)}{||d||}$. This simple representation doesn't capture the fact that some terms are more important than others, according to the specific collection analyzed. For instance in a collection of documents about the evolution of human kind, the term "Neanderthal" is much more important than the often used term "Man". If the term t occurs N_t times out of N_C documents belonging to a collection C, then $\frac{N_t}{N_C}$ measures the rarity of t and so its relative importance to the collection. The "inverse document frequency" is defined as $IDF(t,C) = 1 + \log \frac{N_t}{N_C}$. TFIDF is a measure often used in information retrieval to express the relevance of a term t in a document d of a collection C:

$$\text{TFIDF}(t,d,C) = \frac{n(t,d)}{||d||} * (1 + \log \frac{N_t}{N_C}) \tag{1.1}$$

The weighted vector space assigns the value $\text{TFIDF}(t,d,C)$ to the t-th coordinate of the vector $\vec{d}$ representing the document d. A query q can be expressed introducing a vector $\vec{q}$ in the Euclidean space. The query can use boolean or more complex operators to express the relationship between the terms contained in the queries and those contained in the set $\mathcal{R}$ of retrieved documents. Documents in $\mathcal{R}$ can be ranked according to their affinity with q. A measure often used is the so called *"cosine measure"*. For each $d \in \mathcal{R}$ we compute $\text{cosine}(d,q) = \frac{d.q}{||d||_2\ ||q||_2}$ and sort the documents according to the increasing value of the cosine measure. From a geometrical point of view, $\text{cosine}(d,q)$ measures the cosine of the angle between vectors $\vec{d}$ and $\vec{q}$. The nearer the vectors are, the less the cosine of the angle is.

Several measures are defined to assess the quality of IR tools. For each query q belonging to a query set Q, an exhaustive set of relevant documents $D_q \subseteq D$ is manually identified. In response to the query q, the IR tool provides a ranked list of documents $(d_1, d_2, \ldots d_n)$. We can compute a ranked 0/1 relevance list $(r_1, r_2, \ldots r_n)$, where $r_i = 1$ if $d_i \in D_q$ and 0 otherwise. *Precision* is the number of relevant documents divided by the total number of documents retrieved. We define the precision at rank k as:

$$\text{precision}(k) = \frac{\sum_{i \leq k} r_i}{k}$$

Recall is defined as the number of relevant documents retrieved, divided by the total number of relevant documents in a collection. We define the recall at rank k as:

$$\text{recall}(k) = \frac{\sum_{i \leq k} r_i}{|D_q|}$$

Generally, there is a trade-off between precision and recall. If $k = 0$, precision is by convention 1 but recall is 0. To increase the recall, we can inspect more documents (by increasing k), but this will possibly reduce the precision since we can encounter more irrelevant documents. A measure often used in the literature to combine precision and recall and get one unique values is the F-measure:

$$F_b = \frac{(b^2+1)(P \times R)}{b^2 P + R}$$

if $b = 0$ then F is the precision P for a fixed k, if $b \to \infty$ then F is the recall R for a k fixed, if $b = 1$ then recall and precision are equally weighted, if $b = 0.5$ then recall is half as important as the precision, if $b = 2$ then recall is twice important as the precision. It has been noted that the use of precision and recall is difficult in WebIR [27], since the huge amount of documents makes it almost impossible to manually judge which documents are "relevant" for a broad queries.

Other models of document representation can be devised. For instance, probabilistic models are proposed [81], such as the boolean or the multinomial. Their description is out of the scope of this Thesis. Anyway, we remark one common aspect. In spite of minor variations, all these models regard documents as multi-sets of terms without paying attention to the order and proximity between terms. Therefore they are called a *bag-of-words* model. In Chapter 4 we introduce various extensions to the bag-of-words model, where the order of words is exploited to cluster the documents and to facilitate the extraction of meaningful labels which denote them.

1.2 Modern WebIR

Modern WebIR is a discipline which has exploited some of the classical results of Information Retrieval developing innovative models of information access. A recent Forrester Research report showed that 80% of Web surfers discover the new sites that they visit through search engines [225] (such as Ask, Google, MSN or Yahoo). Therefore, search engines have established as a revolutionary working metaphor. If someone needs information about a book, an address, a research paper, a flight ticket, or almost any other topic, they just make a query on a search engine. In this paragraph we briefly review the architecture of a typical search engine, the interested reader can refer to [27, 183, 37]. In the rest of this Thesis we consider the Web graph G_{Web} as a dynamically evolving graph where the pages are its nodes and the hyper-textual links are its edges.

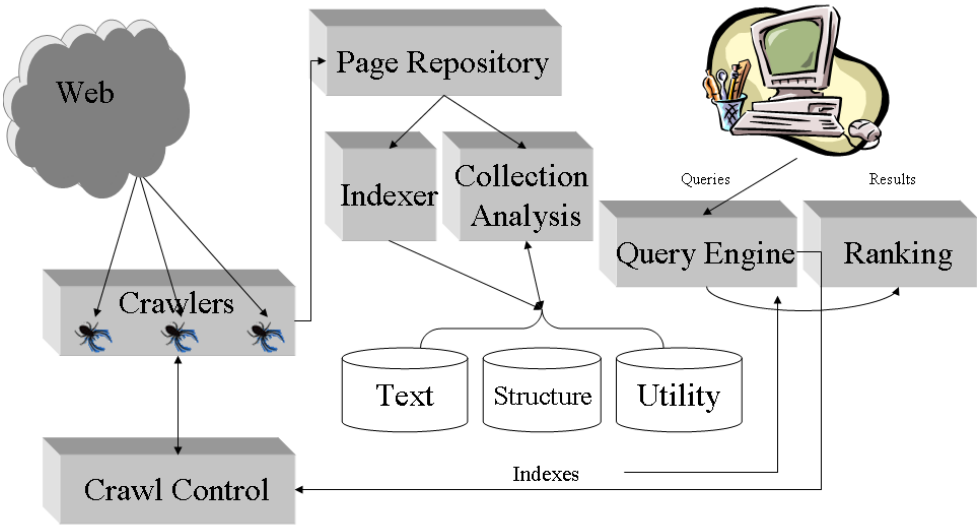


Figure 1.1: Typical architecture of a search engine

Crawlers are distributed agents which gather information from the Web. They crawl the Web graph visiting the pages according to some policies (BFS, DFS, random, topic focused, prioritized) and store the pages in a local *Page Repository*. From time to time, the pages are *indexed* and *analyzed* to produce a fast indexable representation of both the documents (nodes of G_{Web}) and the link structures (edges of G_{Web}). Both the textual and structural indexes are then analyzed to *rank* the documents stored in the repository. For efficiency reasons, part of this ranking process can be performed off line, before the query is submitted through the query engine. Several search engines model each Web page p as a *virtual document* consisting of two sets of terms: The set $A(p)$ formed by the terms contained in p, and the set $B(p)$ formed by the terms contained in the (anchor) text surrounding each hyper-link that points to p. We refer to $A(p)$ as the *content* of p, and to $B(p)$ as the *context* of p. The virtual document for p is then given by $A(p) \cup B(p)$. The motivation which underlies the use of $B(p)$ is that anchor texts often provide a precise, yet concise, description of the Web page p. In search-engine literature, the use of anchor texts is well established [27, 104].

Nowadays, modern search engines index billions of objects on distributed platforms of thousands of commodity PCs running Linux. The index is often organized in a two tier structure and is replicated by thousands of autonomous clusters of servers. This distribution is needed to sustain the peak load of thousands of queries per second. There are many academic papers about search engines ranging from crawling [169, 34], Web graph topology [32], indexing and infrastructure [183, 27, 11], efficient query processing [30], caching [141] and ranking [25].

Now, we discuss the data structures commonly used for indexing Web objects. These data structures will be used in the rest of this Thesis both for clustering and ranking.

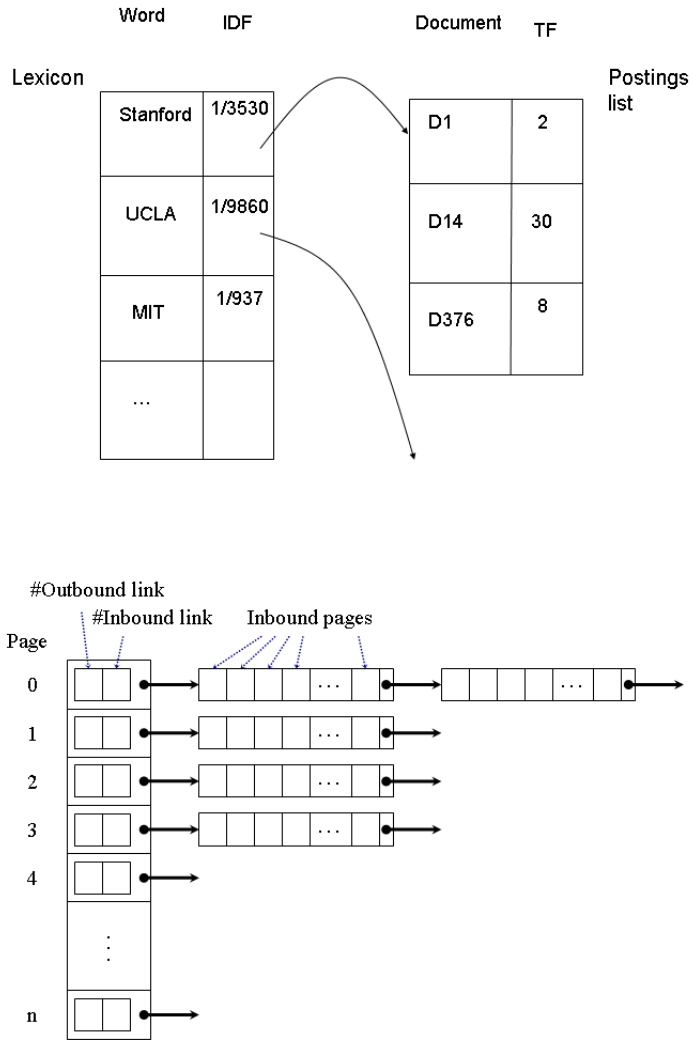


Figure 1.2: An example of an inverted list and graph indexing.

Inverted Lists: Inverted Lists are used by search engines to answer word-based queries. The idea is very simple: all the distinct tokens extracted by the documents are stored in a *lexicon*. Each token in the lexicon points to a variable length list (the *posting list*) which contains the identifier of the documents (and optionally the position in the text) where the word occurs. Optionally, other information is inserted in the inverted list. For instance, the lexicon can store the IDF measure. In Figure 1.2 we show an example of an inverted list.

Graph indexing: Search engines need to store and to rapidly access the link structure of the Web. Figure 1.2 describes a data structure which stores, for each Web page, the inbound links.

Different indexing data structures can be defined, according to the type of queries which should be supported. Generally, the search engine should provide rapid access to those nodes which are pointed to by a given node, or to those nodes which point to a given node. This kind of indexing data structure is useful to provide on-line services (such as "who is linking to my pages?"), as well as a tool for supporting the computation of social network algorithms. In this way, the tool can also be used internally by the search engine itself. Many techniques have been proposed in the literature to compress a Web graph [24] and advanced graph indexing techniques have been presented in [17].

1.3 Size of the Web

This issues is crucial in understanding why modern Web Information Retrieval is so difficult. [1] Google claims to index more than 8 billion pages, MSN claims about 5 billion pages, Yahoo! at least 4 billion and Ask/Teoma more than 3 billion. Two sources for tracking the growth of the Web are [203, 201], although they are not kept up to date. Estimating the size of the whole Web is quite difficult, due to its dynamic nature. [2] Nevertheless, it is possible to assess the size of the publically indexable Web, defined as *"the part of the Web which is considered for indexing by the major engines"* [167]. In 1997, Bharat and Broder [15] estimated the size of the Wed indexed by Hotbot, Altavista, Excite and Infoseek (the largest search engines at that time) at 200 million pages. They also pointed out that the estimated intersection of the indexes was less than 1.4%, or about 2.2 million pages. Furthermore, in 1998, Lawrence and Giles [226, 134] gave a lower bound of 800 million pages. These estimates are now obsolete.

In a study [96], we revised and updated the estimated size of the indexable Web to at least 11.5 billion pages as of the end of January 2005. We also estimated the relative size and overlap of the largest four Web search engines. Google is the largest engine, followed by Yahoo!, Ask/Teoma, and MSN. We adopted the methodology proposed in 1997 by Bharat and Broder [15], but extended the number of queries used for testing from 35,000 in English, to more than 438,141 in 75 different languages. Our experiments were conducted on a cluster of 43 Linux servers, requiring about 70Gb of bandwidth and more than 3600 machine-hours. We included Google, MSN, Yahoo! and Ask/Teoma as test beds for our experiments, since these engines claim to have the largest indexes of the Web. We remark that an estimate of the size of the Web is useful in many situations, such as when compressing, ranking, spidering, indexing and mining the Web.

1.3.1 Estimating sizes and overlaps of search engines

We review [15] and point out where our approach differs. Suppose we have two search engines A and B with size $s(A)$ and $s(B)$, respectively, and intersection $A\&B$. Let $Pr(A)$ represent the probability that an element belongs to the set A, and let $Pr(A\ \&\ B|A)$ represent the conditional probability that an element belongs to both sets given that it belongs to A. Then, $Pr(A\&B|A) \approx s(A\&B)/s(A)$ and similarly, $Pr(A\&B|B) \approx s(A\&B)/s(B)$, and therefore the relative size is $s(A)/s(B)$ that is approximately $Pr(A\&B|B)/Pr(A\&B|A)$. The methodology estimates the ratio between sizes $\frac{s(A)}{s(B)}$ by the ratio between the fraction of URLs sampled from B found in A and the fraction of URLs sampled from A found in B. It also estimates the overlap (fraction of search engine A's index, indexed by search engine B) as a fraction of the URLs sampled from A found in B.

In order to implement this idea, a procedure for picking pages uniformly at random from the index of a particular engine is necessary - i.e. a *sampling procedure* -, and a procedure for determining whether a particular page is indexed by a particular engine is also necessary - i.e. a *checking procedure*. We can refer the interested reader to [15] for a discussion on the bias of this methodology.

[1] At the time of writing of this Thesis, the search engines' size war had re-started again [204]

[2] According to Andrei Broder, the size of the whole Web depends strongly on whether his laptop is on the Web, since it can be configured to produce links to an infinite number of URLs!

Sampling: Bharat and Broder suggested a query-based sampling procedure. A set of disjunctive and conjunctive queries is submitted to the target search engine and an URL at random is selected from the top 100 results. They built the query lexicon by indexing 300,000 documents, extracted from the Yahoo! directory, and creating a lexicon of about 400,000 terms. The terms were combined to form 4 trials for a total of 35,000 queries.

We have extended this approach, by indexing the whole DMOZ.com directory - more than 4 million pages - and obtaining a set of 2,190,702 terms. The DMOZ directory contains pages in more than 75 languages, unlike subset of the Yahoo! directory used by Bharat and Broder. We sorted the terms by occurrence and divided the sorted list into blocks of 20 terms. From each block we extracted one query term for each search engine in the test bed, obtaining a total of 438,141 one-term queries. Queries were divided into many trials and submitted to Google, MSN, Yahoo! and Ask/Teoma. For each query, at least one URL at random was selected from the first 100 results.

Checking: Bharat and Broder suggested a query-based checking procedure. For each URL u in the sampling set, they extracted the k most discriminant terms from the downloaded Web page, using a TFIDF measure. Then, these k terms were submitted to each search engine to check if theycould find u. They tested 35,000 URLs.

We adopted a simplified form of checking. In fact, every engine in our test bed provides an interface to check directly if a URL is indexed. Unfortunately this requires having a well-formed URL to check: thus, a lot of care was taken in normalizing the URLs. In particular, we exploited several heuristics for checking dynamically-generated Web pages and we filtered those pages not recognized by the originating engine after normalization. As a result, the effective number of checked URLs was 486,752. Experimental results have confirmed that for each search engine this data set is large enough to provide a stable estimate of its searchable Web coverage (see Figure 1.3). This also confirms that the use of one-term queries is reasonable.

Both for sampling and for checking we have exploited Helios, a flexible and open source meta-search engine described in paper [95] and still developed during the Ph.D. Each sampling query was submitted to Helios, which forwarded it to the search engines in parallel. A similar process was used for checking the URLs. Results were retrieved, parsed and saved locally.

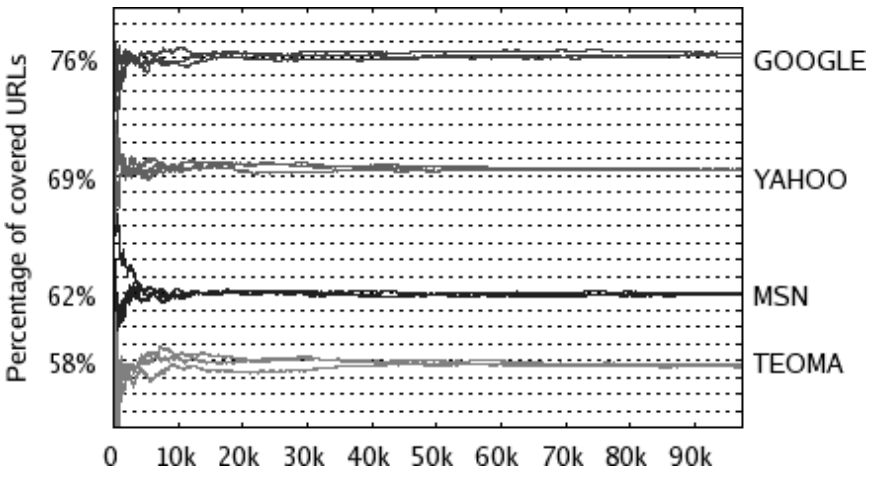


Figure 1.3: % URLs covered as the data set increases

Figure 1.3 shows that coverage does not change significantly as the number of checked URLs exceeds twenty thousand. Google covers around 76.2% of our sampling data set, Yahoo! covers around 69.3%, MSN covers around 61.9% and Ask/Teoma covers around 57.6%.

Figure 1.4 represents the relative pairwise overlaps and the relative sizes between engines obtained in each of the trials T1, T2 and T3.

In order to reconcile the different pairwise size ratios, we used the least squares method to

		T1	Rel.	T2	Rel.	T3	Rel.
Google	M	55.80%	1.41	55.27%	1.42	55.23%	1.42
	T	35.56%	1.65	35.89%	1.62	35.96%	1.62
	Y	55.63%	1.22	56.60%	1.20	56.04%	1.22
MSN	G	78.40%	0.71	78.48%	0.70	78.42%	0.70
	T	49.56%	0.87	49.57%	0.87	49.87%	0.86
	Y	67.38%	0.73	67.28%	0.73	67.30%	0.74
Ask/Teoma	G	58.83%	0.60	58.17%	0.62	58.20%	0.62
	M	42.99%	1.15	42.95%	1.15	42.68%	1.17
	Y	54.13%	0.84	53.70%	0.84	54.13%	0.83
Yahoo!	G	67.96%	0.82	67.71%	0.84	68.45%	0.82
	M	49.33%	1.37	49.38%	1.36	49.56%	1.36
	T	45.21%	1.20	45.32%	1.19	44.98%	1.20

Figure 1.4: Pairwise overlap, relative size (3 trials).

compute a best-fit solution on an over-constrained system. In particular, we estimated the engine sizes so that the sum of the squared differences between the resulting estimates for the pairwise overlap was minimized.

Another possible approach for estimating the engine size is to minimize the sum of the absolute values of the differences between the claimed engine sizes and the size calculated using relative values. This computation can has been carried out using linear programming. The objective function is to minimize the sum of the differences between the claimed engine sizes and the size calculated using relative values. This results in 12 constraints like $size(a)/rel\_size(a,b) - size(b) \leq d_i$ for $1 \leq i \leq 12$, where each d_i represents a pairing of engines. We use the declared search engine size as a lower boundary for each engine variable. Our experiments showed that using just two lower boundaries, the engine sizes are stable, confirming the results of the above least squares method.

Then, we expressed the size of each search engine as a ratio to the largest engine size (here, Google) (see Figure 1.5).

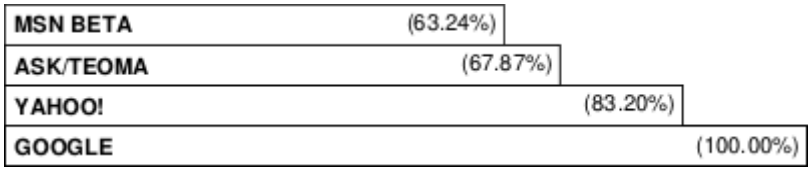


Figure 1.5: Estimated relative size per search engine

Figure 1.6 graphically represents the percentage of the indexable Web that lies in each search engine's index and in their respective intersections.

1.3.2 Estimating the indexable Web

As suggested by [201], we assumed that the indexes of search engines were constructed independently. For any given engine E, we averaged the fraction of every other search engine index which appears also to be indexed by E. For instance, Google appears to index around 68.2% of any other search engine, MSN indexes around 49.2%, Ask/Teoma indexes around 43.5% and Yahoo! indexes about 59.1%. We can consider these as representative of each engine's coverage of the indexable Web at large.

Furthermore, we can compute the size of the indexable Web by exploiting both the relative size estimated in previous subsection and the absolute size declared by each search engine in our testbed. Averaging these values, we have estimated the indexable Web to be approximately 11.5

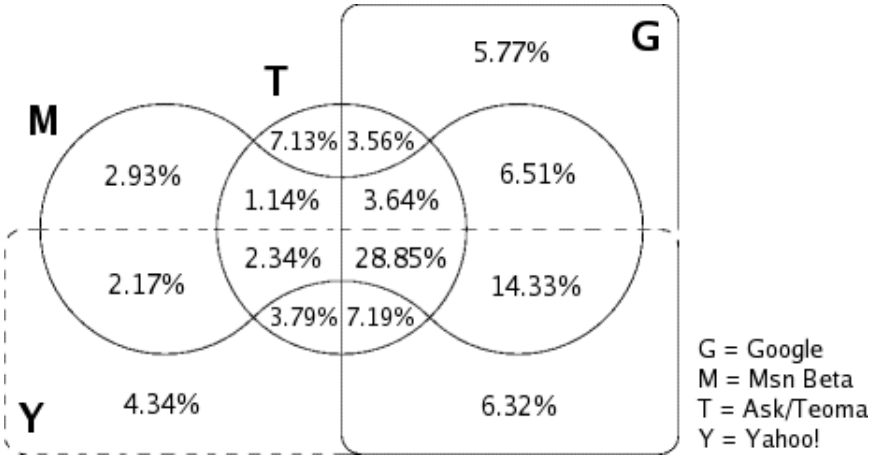


Figure 1.6: Distribution of results across the engines.

billion pages. As reported in Figure 1.6, the estimated intersection of all four indexes is 28.85%, or about 2.7 billion pages, and their union is about 9.36 billion pages.

The research line proposed in this Section was later extended in a paper [12], which evaluates different sampling approaches. It is worth noting that all methods based on capture-recapture are heavily biased in favor of content rich pages and top-ranked pages. This could raise many problems. First at all, the sampling itself is non-uniform. Second, it privileges those engines which index the whole Web page instead of just a part of it. The above paper proposes different solutions to these problems.

The size of the Web poses serious issues for determining fresh and relevant content on the Internet. The goal of this Thesis is to point out some innovative algorithms and numerical methodologies for addressing these issues.

Chapter 2

An Overview of Web Snippet Clustering

Good judgment comes from experience. Experience comes from bad judgment, Jim Horning

Abstract

In this Chapter, we will review classical clustering methodologies following the notation adopted in [37]. In particular, we will briefly describe partition based, geometric and probabilistic approaches. Then, we will focus our attention on the new techniques recently published to cluster Web pages. This is an innovative theme of research for WebIR.

Cluster analysis has been successfully exploited in statistics, numerical analysis, machine learning and in other fields. The term *"Clustering"* denotes a wide range of methodologies for identifying hidden common structures in large sets of objects. A cluster is a group of objects whose members are more similar to each other than the members of other clusters. In this case, we say that intracluster similarity is high and intercluster similarity is low. Clustering methods are classified according to four aspects:

- *The structure*: This could be flat (there is no relationship between different clusters), hierarchical (clusters are organized in a tree), or overlapping (objects can be members of more than one cluster).
- *The indexing unit*: Documents are represented by means of a set of words, a so-called *bag of words* representation, or by means of sentences where the order of words is taken into account.
- *The duration*: The clustering is either carried out on top of a persistent collection of documents or on top of documents which exist for a very short period, like the set of search results given as an answer to a particular query submitted to a search engine. Several authors call this *ephemeral* clustering.
- *The algorithm*: It is used to generate the clusters, and could be divisive (starting from a set of objects and splitting them into subsets, possibly overlapping) or agglomerative (starting from individual objects and merging them into clusters).

Until a few years ago, *persistent* clustering was considered the "default" clustering technique, since, as stated by Salton [166], "in normal circumstances, the cluster structure is generated only once, and cluster maintenance can be carried out at relatively infrequent intervals". The *ephemeral* clustering process organizes the documents in groups, which will survive just for the current session. Nowadays, ephemeral clustering is used by several search and meta-search engines to organize their results in fast browsable groups. Surprisingly, ephemeral clustering has been less studied than persistent clustering in literature.

Clustering Proprieties	
Structure	flat, hierarchial, overlapping
Indexing unit	word, sentence, approximate sentences
Duration	ephemeral, persistent
Algorithm	divisive, agglomerative

Figure 2.1: Taxonomy of clustering properties

2.1 Basic definition and notation

We will focus our attention on the case of document clustering. The interested reader can refer to [37, 78, 182] for the use of clustering methodologies in different disciplines. In our discussion, we will follow the notation used in the above books. The usefulness of document clustering lies in the so-called *cluster hypothesis*: given a clustering collection, if the user is interested in a document d_i, (s)he is likely to be interested in other members of the cluster to which d_i belongs.

2.1.1 Similarity and dissimilarity

The key idea of clustering is the definition of a notion of similarity amongst the objects in a group. The main goal of clustering techniques is to partition the documents in k subsets in order to maximize the inter-clusters dissimilarity and to maximize the intra-clusters similarity. Following [78], we say that a function d is a dissimilarity index over a set Ω of documents, if it satisfies the following properties:

1. $d(d_i, d_i) = 0 \ \forall \ d_i \in \Omega$
2. $d(d_i, d_j) = d(d_j, d_i) \ \ ,\forall \ (d_i, d_j) \ \in \Omega \times \Omega$

Note that dissimilarity could not be a distance since it is not required to satisfy the triangle inequality. Given a measure of dissimilarity d, the corresponding similarity s can be easily defined as $s(d_i, d_j) = 1 \ - \ d(d_i, d_j) \ \ \forall \ (d_i, d_j) \in \Omega \times \Omega$. The most common similarity measure between two documents is the cosine coefficient between their associated vectors after normalization:

$$s(\vec{d_i}, \vec{d_j}) = \frac{\vec{d_i}.\vec{d_j}}{||\vec{d_i}||_2 \ ||\vec{d_j}||_2}$$

Another commonly used measure is the Jaccard coefficient defined as follows:

$$s(d_i, d_j) = \frac{|T(d_i) \bigcap T(d_j)|}{|T(d_i) \bigcup T(d_j)|}$$

where $T(d_i)$ is the set of "indexing units" contained in the document d_i. In classical clustering the "bag-of-words" representation is chosen and the indexing unit is a single term. In the context of Web snippet Clustering we will see that more complex forms, such as phrases [186] or lexical constructions [143] are used. In Chapter 4 we will introduce a new indexing unit to identify snippets which are semantically similar but not syntactically the same, since some terms are missing.

2.1.2 Partition approaches

There are two different kinds of partition approaches. Bottom-up clustering starts with a set of documents and groups them together, until the number of partitions is acceptable. Top-down clustering starts with a pre-defined number of partitions and assigns documents to them.

Hierarchical clustering

Hierarchical Clustering is a partition approach which works bottom-up. It groups objects and creates a tree structure between the clusters. Hierarchical Agglomerative Clustering (HAC) is the most frequently used technique of hierarchical clustering. The pseudo-code of hierarchical clustering is shown in Figure 2.2. HAC builds a sequence of partitions of the original set Ω of documents. When the cluster results from a merge operation, its direct descendants are its two sub-clusters from the previous level. Therefore, the result is a hierarchy (or a dendrogram, a tree where any internal node has just two descendants) which organizes documents from the coarsest to the finest partition.

```
Let Ω the group of documents organized in singletons {d};
while  ((|Ω| > 1) AND (merged == 1)){
    merged = 0;
    choose (δ, γ)  ∈  Ω × Ω to maximize the similarity measure s(δ, γ);
    if exist (δ, γ) {
        remove (δ, γ) from Ω × Ω;
        add θ = δ ⋃ γ to Ω;
        merged = 1;
    }
}
```

Figure 2.2: Hierarchical agglomerative clustering pseudo-code.

HAC needs the definition of a similarity measure between individual documents, and also the definition of a similarity between clusters of documents. The similarity between clusters can be defined as:

- *Single link method*: The similarity between two clusters I and J is the maximum of the similarity between a pair (d_i, d_j) where $d_i \in I$ and $d_j \in J$. Therefore, two clusters are similar if they have similar pairs of members. Single link clustering often shows the tendency to form long chains which do not correspond to the intuition that clusters should be compact, and spherical objects.

- *Complete link method*: The similarity between two clusters I and J is the minimum of the similarity between a pair (d_i, d_j) where $d_i \in I$ and $d_j \in J$. Therefore, two clusters are similar if each pair of members are similar. Complete-link clustering is often sensible to outliers (objects which have a loose relationship with the rest of the dataset). Nevertheless, it is usually favored in IR applications, because it tends to produce smaller and more tightly-linked clusters, compared to other methods.

- *Average link method*: Average link method is a compromise between the sensitivity of complete-link clustering to outliers and the tendency of single-link clustering to form long chains. The key idea is to merge in each iteration the pair of clusters with the highest cohesion. The cohesion G of a cluster C is defined as: $G(C) = \frac{1}{n(n-1)}(\gamma(C) - n)$ where $n = |C|$, and $\gamma(C) = \sum_{v \in C} \sum_{w \in C} < v, w >$ and $< v, w >$ is the dot product of v and w.

- *Minimum Variance method*: The minimum variance method (also known as Ward's method [78]), joins the pair of cluster whose merger minimizes the increase in the total within-group error sum of squares, based on the Euclidean distance between centroids. Centroids are virtual objects which represent all the objects belonging to that cluster.

Pure Hierarchical clustering needs $O(n^2)$ time for single-link and group-average algorithms, or even up to $O(n^3)$ time for complete-link algorithms. Therefore, hierarchical methods could be unsuitable for large document collections. In the next Section, we will review a pseudo-linear top-down approach.

Hierarchy clustering is much more effective for browsing, because it allows a dynamically created tree to be navigated from general to more specific topics. Leaves of the tree represent the most specific level of separation between the objects, while any internal node in the hierarchy captures an intermediate level of separation. In this way, any path from the root to a specific leaf gives users much more information than a single node. Moreover, hierarchical clustering can produce a number of clusters which is not predefined but is adapted according to the type of objects to group. In Chapter 4 we will introduce SnakeT, our clustering engine which uses an innovative hierarchical clustering algorithm.

K-Means

k-means is a well-known top-down clustering approach. The user is supposed to provide k, the number of desired clusters. Documents are represented using the vector space model. Initially, documents are randomly assigned to the k clusters. For each cluster, a virtual document (*"the centroid"*) is created which represents all the documents belonging to that cluster. The documents are then moved to the cluster which has the most similar centroid. The pseudo-code is shown in Figure 2.3. The overall time complexity is O(kn) since n documents are compared to k centroids, and the number of iterations performed by the `while` cycle of the algorithm is not usually related to n, and k is considered constant.

```
It Assigns documents to the k clusters arbitrarily;
while (other improvements are possible) {
    compute the centroids of the clusters;
    foreach document d {
        find a cluster c whose centroid is the most similar to d;
        assign d to c;
    }
}
```

Figure 2.3: K-Means pseudo-code.

There are different ways to compute the centroids, but the most commonly used one is the creation of a vector, which is the average geometric representation of the documents in the cluster. There are also various termination conditions. The one which is most commonly used halts the computation when documents stop moving in different clusters.

2.1.3 Geometric approaches

Geometric approaches represent the document in a high dimension of Euclidean space. The clustering phase collapses dimensions (i.e. terms) which are likely to co-occur in the document collections. This will result in reduced dimensions. In this class of algorithms we find *Self Organizing Maps (SOM)*, *Multidimensional Scaling*, *Fast Map* and *Latent Semantic Indexing*. In this Chapter, we will describe just the last approach and we suggest that the reader refers to [37] for a thoughtful description of the other algorithms.

LSI: latent semantic indexing

SVD (Singular Value Decomposition) is a general matrix decomposition technique which is often used both for ranking [173] and clustering [37, 189, 157]. Let A be a rectangular $n \times m$ matrix with rank r. It is a well known result in linear matrix algebra that A can be decomposed in a normal form

$$A = U\Sigma V^T$$

where $\Sigma = diag(\sigma_1 \ldots \sigma_r)$, U and V are orthonormal matrices sized $(n \times r)$ and $(r \times m)$ respectively. A matrix U is orthonormal if $U^T U = UU^T = I$. The values $(\sigma_1, \ldots, \sigma_r)$ are the r singular values of A sorted so that $\sigma_1 \geq \ldots \geq \sigma_r$. Let $1 \leq k \leq r$ and let

$$A_k = \sum_{1 \leq i \leq k} \vec{u_i} \sigma_i \vec{v_i} \tag{2.1}$$

where $\vec{u_i}$ and $\vec{v_i}$ are the i-th component of U and V respectively. It can be shown that A_k is the best approximation of A under the Frobenious norm $||A||_F = \sqrt{\sum_{t,d} A[t,d]^2}$. The error is estimated by:

$$\min_{rank(B)=k} ||A - B||_F^2 = ||A - A_k||_F^2 = \sigma_{k+1} + \ldots + \sigma_r$$

Suppose that A is a (document $\times$ term) matrix having documents as columns and the distinct terms as rows. It can be demonstrated [160] that terms co-occurring in the original document collection are likely to be projected together in the vector space spanned by eigenvectors of A_k. Therefore, LSI can be seen as a form of clustering. For instance, synonyms such as "car" and "automobile" are likely to be projected onto the same axis. In this way, this technique captures the *"latent semantic"* meaning of the document collection. Moreover, since

$$A^T A = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T = (\Sigma V^T)^T \Sigma V^T$$

the pairwise document to document similarity $A^T A$ can be computed as $V\Sigma^2 V^T$. A similar demonstration can be shown for the pairwise term to term similarity:

$$AA^T = U\Sigma^2 U^T = (U\Sigma)(U\Sigma)^T$$

Given the matrix A, LSI pre-computes the matrices U, Σ and V^T. In order to query the document collection, LSI first projects the query q to the restricted "LSI space" $q_r = \Sigma^{-1} U^T q$, and then applies a cosine measure to find similar documents in the reduced space, thus taking the advantage of the reduced number of dimensions to speed up the computation. The main problem with SVD and LSI is that this type of matrix decomposition is very expensive and therefore it is difficult to apply it to a large dataset.

2.1.4 Syntactical clustering

In [31], Broder et al. discuss an efficient way to determine the syntactic similarity of near-duplicate documents situated on the Web. Near-duplicate files are similar but not exactly the same.

A contiguous subsequence contained in document D is called a q-gram, or a shingle, and the w-shingling $S(D, w)$ is the set of all unique shingles of size w contained in D. For instance, the 4-shingling of (a,rose,is,a,rose,is,a,rose) is the set { (a,rose,is,a), (rose,is,a,rose), (is,a,rose,is) }. From now on we let shingle size w be fixed. The resemblance r of two documents A and B is defined as the Jaccard coefficient $r(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$. The containment of A in B is defined as $c(A,B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$, where $|A|$ is the size of the set A. The resemblance is a number between 0 and 1, such that when the resemblance is close to 1 it is likely that the documents are "roughly the same". The containment of A in B is a number between 0 and 1 such that, when close to 1, it indicates that A is "roughly contained" within B.

There is an efficient way ti estimate containment and resemblance based on min-wise independent permutations [28]. Let the shingle size be w, and let U be the set of all shingles of size w represented as integers. For instance, if each word is represented as a 32-bit integer and $w = 4$, then U is an unordered set of 128-bit numbers. For a set W contained in U, let s a fixed parameter and let

$$min_s(W) = \begin{cases} \text{the set of the smallest s elements in W,} & if |W| \geq s; \\ W & otherwise \end{cases} \tag{2.2}$$

where "smallest" refers to numerical order on U, seen as a set of integers. In addition, let m be another parameter and let $mod_m(W)$ be the set of elements of W that are $0 \ mod \ m$.

The following theorem holds [31]:

Theorem: Let $\phi : U \rightarrow U$ be a permutation of U chosen uniformly at random. Let $F(A) = mim_s(\phi(S(A)))$ and $V(A) = MOD_m(\phi(S(A))))$. Define $F(B)$ and $V(B)$ analogously. Then $\frac{|min_s(F(A) \cup F(B)) \cap F(A) \cap F(B)|}{|min_s(F(A) \cap F(B))|}$ is an unbiased estimate of the resemblance of A and B, and the value $\frac{|V(A) \cap V(B)|}{|V(A) \cup V(B)|}$ is an unbiased estimate of the resemblance between A and B and the value $\frac{|V(A) \cap V(B)|}{|V(A)|}$ is an unbiased estimate between the containment of A in B.

As a result, after choosing a random permutation, any document D is represented by a sketch consisting only of the set $F(D)$ and/or $V(D)$. The sketches suffice to estimate the resemblance or the containment of any pair of documents without any need for the original files.

The clustering algorithm adopted consists of four steps. First, all the sketches are computed. In a second phase, the algorithm produces a list of all the shingles and the documents they appear in. This list is sorted by shingle values using a external memory merge-sort algorithm. In the third phase, a list of all the pairs of documents that share any shingles is generated, along with the number of shingles they have in common. This produces a list of triplets $< DOCID, DOCID, count >$. This list is analyzed in the fourth phase to decide if the document pair exceeds a threshold th for resemblance. Documents with high resemblance are joined using a union-find data structure. The connected components form the final clusters. Shingle-clustering is a powerful algorithmic methodology currently adopted by many search engines to find mirrored web pages. Thanks to shingle clustering a notion of similarity between Web pages can be defined, instead of depending on the exact equality of checksums such as MD5. This similarity can be tuned using the parameters w, m, th. An introduction to similarity detection over web pages has been given in [16, 37].

2.1.5 Probabilistic approaches and other methods

We have decided to omit the discussion of probabilistic approaches (in particular expectation maximization [21], and probabilistic LSI [107]), spectral clustering [61, 120], graph clustering [65], randomized clustering [56], co-clustering [145, 158], relational clustering [165], projected clustering [3, 1], density clustering [68, 110], clustering by compressing [48] and many others. All these methodologies are very important but not highly correlated with the main results of our research on Web snippet clustering.

2.1.6 Clustering requirements and limitations

All the clustering algorithms have some limitations and therefore different requirements should be addressed. We will briefly cite some of them and invite the interested reader to refer to ([78] [37], [71]) for more information.

- *Data with outliers*: A clustering algorithm should be robust enough and should not produce bad grouping results in the presence of outliers (i.e. objects with a loose relationship with the remaining dataset). Outliers are usually seen as a source of noise.

- *Chaining Effects*: A clustering algorithm should create groups which have a compact, spherical shape. This is a general requirement, but it can have different degrees of importance according to the particular dataset to cluster.

- *Overlapping Clusters*: An object can have different attributes and therefore can be grouped according to different criteria. So, it is often a good opportunity to have overlapping clusters.

- *Shape of the hierarchy*: For usability, it is often a good idea to superimpose some proprieties to the shape created by the Hierarchical Algorithm:number of levels, number of descendants for each level, and so on.

Many other requirements can be highlighted according to the particular problem faced by the clustering algorithm. In particular, in the next Section we will focus on requirements faced by algorithms which cluster Web pages. In Chapter 4 we will discuss the use of overlapping clusters, the presence of outliers, and the shape of the hierarchy produced by SnakeT, our clustering engine.

2.2 Web snippet clustering

In this Section we introduce the problem of clustering Web pages. We direct particular attention towards the case of Web snippet clustering. In this case, the original Web pages are not given and the clustering process receives an abstract (called "the snippet" of the page) as its input. Snippet clustering was introduced in a primitive form by Northernlight and then made popular by Vivisimo. The problem consists of clustering the results returned by a (meta-)search engine into a *hierarchy of folders* which are *labeled* with variable-length sentences. The labels should capture the "theme" of the query results contained in their associated folders. This labeled hierarchy of folders offers a complementary view to the ranked list of results returned by current search engines. Users can exploit this view *navigating* the folder hierarchy driven by their search needs, with the goal of extracting information from the folder labels, reformulating another query, or narrowing the set of relevant results. This navigational approach is especially useful for informative [29], polysemous and poor queries.

Here we describe the state-of-art technology both from an industrial and an academic point of view. This description will also be used in Chapter 4 where we will present our Web snippet Clustering technology. It goes without saying that the clustering step might benefit from the huge scientific literature nowadays available on this topic (see e.g.[89, 78, 71]), but here the (hierarchical) clustering formation is complicated by many demanding requirements:

- For obvious computational reasons, the engine must cluster the *Web snippets* returned by the queried search engine, and clearly, these snippets are much less informative than the whole of their corresponding documents because they often consist of about 100 words.
- Every cluster must have a meaningful sentence assigned to it, possibly formed by more than one word, in order to capture the "category" of the documents contained in that cluster in an intelligible way.
- Clusters must be organized in a hierarchy and every super-cluster must be assigned again with an intelligible sentence in order to allow the post-navigation of the user for query refinement.
- Unlike persistent clustering, where quality control can be applied to the result (e.g., by manually modifying the hierarchy), in ephemeral clustering the resulting hierarchy is to be used as it is (i.e., it is fully automated).
- A general rule of thumb for clustering is even more stressed: precision is preferred over recall. In practice, this means that when in doubt, a document is not taken into account, in order to avoid the possibility of associating it with the wrong cluster. Instead, the document is left as an outlier.

All these issues make this problem not just *as another clustering problem*, but as a challenging variant both from an algorithmic and IR point of view. In Chapter 4 we will present a hierarchical clustering engine which will address the above requirements and introduce some other innovative ideas. Experiments discussed in Chapter 4, will analyze Mooter and Vivisimo because they are the best performing Web snippet clustering engines among the industrial proposals.

2.2.1 The commercial scenario

In recent years there has been an upsurge of commercial interest [49] on the Web snippet clustering problem. This problem was solved in a primitive form by Northernlight, which stopped working in

2002. It exploited a subject taxonomy, built by humans and classifying a pre-defined set of Web documents by subject, type, language and source in 17,000 categories. In [185] was claimed that *"... browsable folders are organized hierarchically and can be based on type (e.g., press releases, product reviews, etc.), source (sites, domains), language or subject. In order to create subject folders, NorthernLight assigns a set of topic tags (potential folders) to each page as it indexes the page. This is done by classifying the page against one or more values from a 20,000 term subject hierarchy manually developed by librarians. When a query is processed, the system searches for the dominant topic tags among the matching pages and selects from these a subset of tags that are presented as folders. Folders are selected based on the number of documents in each folder, their relevance to the query and their variety.* Note that, since NorthernLight uses a static hierarchy, it is not strictly a clustering, but a classification engine.

Since 2002, this technology has improved in its efficiency and efficacy, and has been employed by many commercial meta-search engines like Vivisimo, Mooter, Copernic, iBoogie, Kartoo, QueryServer, Groxis, Dogpile, and Clusty. They do not use a predefined taxonomy but cluster on-the-fly results returned by commodity search engines. This is a form of ephemeral clustering. For instance, iBoogie, QueryServer and Mooter use both linguistic and statistical methods to hierarchically cluster the query answers and label them via single or pairs of terms. Kartoo is a meta-search engine with a novel visual display interface: the retrieved results are clustered in series of interactive maps through a proprietary algorithm. A similar approach is taken by Groxis, which has developed a knowledge mapping and information visualization tool to cluster search results and to present them by means of a visual graph.

The efficacy of the technology has been recognized by SearchEngineWatch.com which conferred the "best meta-search engine award" to Vivisimo during 2001-03. Last year, Dogpile was voted as the most innovative IR-tool by a jury of about 570 Web users.[1] At the beginning of 2005, Aol's portal decided to adopt Vivisimo to cluster the results provided by Google [206]. Unfortunately, very little information is available about Vivisimo. The only information available about its underlying technology is that it employs statistical, linguistic, and subject matter knowledge, as well as stemming and stopwords. Raul Perez-Valdes, president of Vivisimo, claims *"We use a specially developed heuristic algorithm to group - or cluster - textual documents. This algorithm is based on an old artificial intelligence idea: a good cluster - or document grouping - is one, which possesses a good, readable description. So, rather than form clusters and then figure out how to describe them, we only form describable clusters in the first place ... Our innovation relies on a newly discovered heuristic algorithm that does this well"*. The authors stress the superiority of their algorithm over bag-of-words clustering, claiming that their heuristic performs well even when snippets are extremely small. Vivisimo is a very intriguing system which was an academic spin-off of the Carnegie Mellon University.

An intensive test performed by the author of this Thesis pointed out three important features of the product:

1. Clusters are made up of documents which *share sentences* and not only single keywords.

2. These sentences are made up of *non-contiguous terms*, which appear within a proximity window in the original documents.

3. Clusters can *overlap* and they are organized in a *tree hierarchy*, which separates snippets from a generic to a more specific meaning. The tree does not follow a pre-defined shape, but it is organized dynamically according to the classified topics.

2.2.2 The scientific scenario

Scientific literature offers various solutions to the Web snippet clustering problem. In the simplest case, the folder label is a "bag of words" and the folder clustering is flat. In the more sophisticated case, the folder label is a variable-length sentence and the folder clustering is hierarchical. We

[1] Dogpile has been granted licence to Vivisimo's technology.

survey these approaches by classifying them into a *taxonomy* of four classes based on their folder labeling and hierarchy structuring.

- **Single terms and flat clustering.** Scatter/Gather [105] was one of the first Web-clustering software on top of an IR engine. It may be considered to belong to this class even if it was not tested upon a Web-search engine. WebCat [85] uses Transactional K-Means to produce the flat set of clusters. Retriever [112] uses robust relational fuzzy clustering. The system of [180] expands the set of retrieved snippets with all the in-linking and out-linking pages to improve precision. However search engines do not provide cheap access to the Web graph, making link retrieval efficient only if limited to a local (partial) copy available of the Web graph. Amongst this software only WebCat [85] is available on-line.

- **Sentences and flat clustering.** Grouper [186] was the first publicly available software to address the Web snippet clustering problem. It used sentences of variable length to label the folders, but these sentences were drawn as *contiguous* portions of the snippets by means of a Suffix Tree data structure. Lingo [157] uses SVD on a term-document matrix to find meaningful long labels. Unfortunately, SVD is time consuming when applied to a large number of snippets. Recently, Microsoft [187] proposed a system that extracts (contiguous) sentences of a variable length via regression on five different measures. Altought the clustering is flat, regression needs a training phase (hard to adapt on the whole heterogenous Web), and the system is not available for testing due to its recent commercialization [216]. To sum up, the only available software in this class is Carrot2 [181], an open source implementation of Grouper.

- **Single terms and hierarchical clustering.** FIHC [82] uses an analysis based on the Frequent Itemsets problem [4] in order to construct the folder hierarchy. CREDO [36] uses a concept lattice on single terms, and is the only software in this class available on-line.

- **Sentences and hierarchical clustering.** This is the most interesting class including software that tries to mimic Vivisimo. Lexical Affinities Clustering [143] was the first to propose this approach. It improves precision over recall by using a snippet representation made up of *pairs of terms* (not necessarily contiguous) linked by a lexical affinity, i.e. a correlation of their common appearance. In [186] Etzioni proposed a simple extension of Grouper to hierarchical clustering based on the size of folders' overlap. SHOC [189] uses the Suffix Array for extracting *contiguous* sentences from the snippets, and organizes the folders in a hierarchy via an SVD approach. Highlight [184] adopts lexical analysis and a probabilistic framework for hierarchy construction, but the authors do not provide any evaluation. WhatsOnTheWeb [84] uses a topology-driven approach and models the set of results on a graph whose vertices are the single or multiple terms in the snippets, and whose edges denote the relationships between these terms. The folder hierarchy is obtained by variants of classical graph-clustering algorithms, resulting often in-compact. Besides, the visual interface is server side thus necessitating one Web access for each navigational step. CIIRarchies [136] extracts sentences from the snippets by using a pre-computed language model, and builds the hierarchy via a recursive algorithm. The authors admit that their hierarchies are often in-compact, having large depth and containing some non content-bearing terms which tend to repeat. [2] Recently, IBM [129] proposed a system that constructs the folder hierarchy based on the minimization of an objective function similar to the one we will use in our SnakeT in the Chapter 4. However, their labels frequently consist of single terms, in other (few) cases they are contiguous sentences. The authors did not make this system available for testing. Surprisingly enough, the only software in this class available for testing is Highlight, CIIRarchies and WhatsOnWeb.

The system we will present in Chapter 4 is called SnakeT (SNippet Aggregation for Knowledge ExtracTion), which belongs to this last class. SnakeT is highly engineered, available on-line, and it

[2] Pages 91-92 in [135] "A disadvantage present in both snippet and full text hierarchies is that they include uninformative topics. As the hierarchies become further developed, we will look for techniques to identify these uninformative topics."

aims to overcome the limitations of the above systems (i) by using gapped sentences drawn from the snippets as folder labels, (ii) by adopting some special knowledge bases to rank and select the most meaningful labels, and (iii) by building a hierarchy of possibly overlapping folders. We will compare SnakeT to the best systems of the fourth class available on-line: namely CIIRarchies and Highlight. We will also test Carrot2 for historical reasons: it offers the only available implementation of Grouper. We will not test the most recent results of [187, 129] because they did not allow access to their software, and we could not repeat their experiments because their original dataset are missing and querying the same search engines would give different snippets now.

Name	Single terms as Labels Flat Clusters	Sentences as Labels Flat Clusters	Single terms as Labels Hierarchy of Clusters	Sentences as Labels Hierarchy of Clusters	on-line
WebCat	+				+
Retriever	+				
Scatter/Gather	+				
Wang *et al.*	+				
Grouper		+			
Carrot		+			+
Lingo		+			+
Microsoft		+			
FICH			+		+
Credo			+		+
IBM			+		
SHOC				+	
CIIRarchies				+	+
LA				+	
Highlight				+	+
WhatsOnWeb				+	+
SnakeT				+	+
Mooter			+		+
Vivisimo				+	+

Figure 2.4: Taxonomy of current Web snippet clustering solutions.

2.2.3 A deep overview of the most important clustering algorithms

In this Section we discuss in depth those Web snippet clustering algorithms which have some points in common with our line of research (see Chapter 4).

The suffix tree clustering algorithm

The first clustering algorithm to exploit the order of words, not only their frequencies, was Suffix Tree Clustering (STC) developed by Etzioni and Zamir [186, 185]. The key idea is that sentences are usually more informative than an unorganized set of keywords, and they can be directly used to label the discovered clusters, as seen in Section (2.2), is one the most relevant problems for Web snippet clustering. Etzioni achieved these conclusions: *"The most distinctive aspect of the algorithm was its treatment of documents as strings of words, in contrast with the standard vector-based representation"*. This means that STC assumes that topics are expressed by using identical sentences, where a sentence is an ordered sequence of words. This is an improvement compared to the traditional *bag of words* document's representation used in Information Retrieval (see Section 1.1). For instance, two documents sharing the sentence "cars are often used for traveling" can be assumed as mutually relevant and, therefore, grouped together. The above process can be repeated for a set D of n documents, by extracting all the sentences from them and by grouping together those documents which share the largest number of sentences. Before entering in the details of the algorithm, we introduce its indexing data structures. Let us start with some basic definitions. A string $S = \sigma_1\sigma_2\sigma_3 \ldots \sigma_n$ is a sequence of symbols σ_i defined over an alphabet Σ. A suffix of s is a substring $\sigma_k\sigma_{k+1}\sigma_{k+2} \ldots \sigma_n$, where $1 \leq k \leq n$. We denote with $s[i]$ the suffix starting at position i. We suppose that the symbols $\sigma_i \in \Sigma$ can be compared to each other for equality and precedence.

Suffix tree: Trie and Suffix Tree are data structures used for efficiently searching *any* substring of a given string S. A **Trie** is a tree that has one node for every possible suffix of S. Each node has $|\Sigma|$

possible branches. A **Suffix Tree** (ST) is a compact representation of a trie where each node with exactly one child is merged with its parent (path compression). The tree exploits the fact that some suffixes can share a common substring, which can be stored in the tree by creating an intermediate node. The edge (i, j) of the tree is labeled using the substring shared among all the nodes in the subtree rooted at j. In Figure 2.5 we show an example of suffix tree for the string "mississippi". ST is a data structure which allows us to retrieve all the suffixes of S in lexicographical order, simply using a depth-first visit (linear time). Another important characteristic of a suffix tree is that it can be built in O(n) time, where $|S| = n$ in case of $|\Sigma| = O(1)$. Another important property of the Suffix Tree is that given a string S, a pattern P can be found in O($p + occ$), after building the Suffix Tree in O(n). Here we assume $|P| = p$, and occ is the number of occurrences of P in S. Note that a suffix tree needs O($n|\Sigma|$) space that grows with alphabet size $|\Sigma|$. Suffix Tree has been exploited in many different fields ranging from biology [97] to optimal compression [72].

The first linear algorithm for suffix tree construction was proposed by McCreight [149] in 1976. Then, in 1995, Esko Ukkonen proposed an on-line algorithm [178]. For suffix tree the big O notation hides very high constants both in time and space. For this reason the Suffix Array data structure has been introduced. We will discuss it in the next subsection.

Generalized suffix tree: Let S be a set of strings $S = S_1, S_2, \ldots, S_t$ over an alphabet Σ. A generalized Suffix Tree (GST) for S is a compressed trie in which each suffix of any string in S can be found enumerating each any from the root node to each leaf node. Intuitively, a GST is the *"union"* of many suffix trees. Note that we can easily search any sequence of contiguous words in a given collection of texts, by building a GST over an alphabet made of words (sequence of characters space separated) instead of single characters [6].

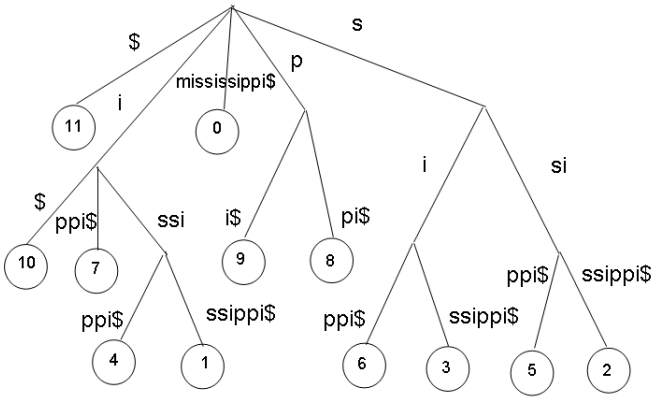


Figure 2.5: Suffix Tree for the string "mississippi$".

STC is an ephemeral clustering algorithm which works into two phases: the first one discovers sets of documents sharing the same sentence (*base clusters*), and the second one combines them into larger entities (*final clusters*). At the end of the two phases we obtain a set of (possibly overlapping) clusters, represented via a flat organization. We must point out that the above two steps can be performed after an optional cleaning phase where stop-words are skipped and the text is stemmed.

The first step of the algorithm discovers the base clusters by creating a Generalized Suffix Tree (GST) over each snippet $d_i, i = 1 \ldots n$. STC builds the GTS using an alphabet Σ made of words, instead of single characters. In Figure 2.6 we show the generalized suffix tree built for three documents d_1, d_2, d_3. In this simple example each document contains just a sentence. Each internal node n denotes the prefix shared by all the suffixes represented in the subtree rooted at n. Each leaf

contains a couple of values. The first one is the index which identifies the document represented. The second value represents the position in the string where a suffix which is not stated with any other document begins. Note that, this step can be realized *"on-line"*. This means that STC incrementally knows all the sentences shared by the documents $d_1, \ldots d_{i-1}$, before inserting the document d_i. The result of this base cluster step is a set of m sentences shared by at least two documents, with links to all documents they occur in. Each sentence s_j is seen as the label that describes the associated base clusters b_j, where $1 \leq j \leq m$. Besides, each base cluster b_j contains a pointer to all the documents $d(b_j)$ it contains, and have associated a variant of TF-IDF score $s(b_j)$ which uses boosting factors for long and more descriptive sentences:

$$s(b_j) = |s_j| * f(s_j) * \sum_{t \in s_j} TFIDF(t) \tag{2.3}$$

where $|s_j|$ is the number of words in the sentence s_j, $f(s_j)$ is a factor which penalizes short-phrases, and $\sum_{t \in s_j} TFIDF(t)$ is a sum of standard $TFIDF$ ranking factors for all terms in phrase p_j. We remark that this step of the algorithm produces a cluster if and only if documents share *exactly* the same sentence. In Chapter 4, we discuss our Web snippet clustering proposal which overcome this need for precise matching.

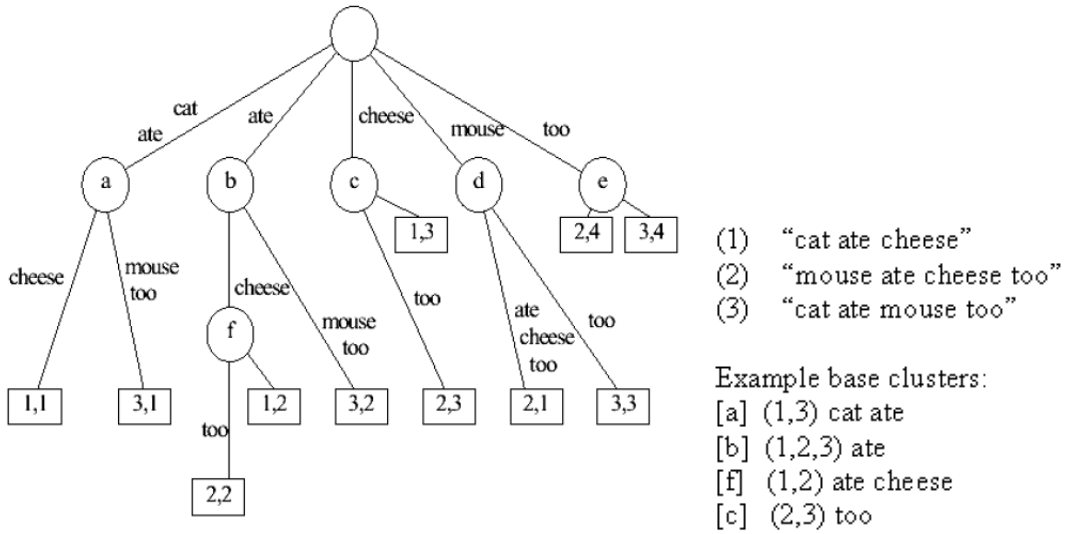


Figure 2.6: An example of a generalized suffix tree. The first document contains just the sentence "cat ate cheese", the second contains the sentence "mouse ate cheese too", and third contains the sentence "cat ate mouse too". Some examples of base clusters are shown. For instance the base cluster [a] is made up of the first and the second document, and they share the substring "cat ate".

The second step of the algorithm takes into account only those base clusters with a score higher than a threshold (minimal base cluster score). Base clusters are considered in order of decreasing score. Two base clusters b_j and b_k are merged whenever their document sets $d(b_j)$ and $d(b_k)$ *"overlap"* for more than a predefined threshold τ. The key idea is that two clusters should be merged if they both share the majority of their documents. The overlap is measured in term of operations on sets as follows:

$$\text{similarity}(b_j, b_k) = \begin{cases} 1 & \text{if } \left(\frac{|d(b_j) \cap d(b_k)|}{|d(b_j)|} > \tau\right) \wedge \left(\frac{|d(b_j) \cap d(b_k)|}{|d(b_k)|} > \tau\right) \\ 0 & otherwise \end{cases} \tag{2.4}$$

The merging finds all connected components of a phrase cluster graph, whose vertices are base clusters and an edge between vertices b_j and b_k exists only when $\text{similarity}(b_i, b_j) = 1$. By selecting

a different threshold τ, one can influence the aggressiveness of the merge phase. A cluster, resulting from a merge operation, takes that particular sentence which has the maximal score among the ones denoting the composing base clusters as a descriptive label. In this way, STC can potentially cluster documents which relate to a topic expressed by more than one sentence, discarding a less descriptive one. The score associated with the merged cluster is eventually recomputed using eq. (2.3). At the end of the two phases, the top-k scored clusters are shown. Note that k is a parameter provided as an input to the algorithm. We can remark that the second step of the algorithm produces a flat organization of clusters. In Chapter 4 we present our proposal which organizes clusters according to a hierarchy. This proposal will overcome the limits of STC clustering.

A few words about the complexity of STC: The base cluster phase requires a time linear in the size of the documents, but hidden constants are high and the known GST implementations require a lot of memory. The merge phase is also linear in the number of base clusters. So, the overall complexity is $O(n)$.

Grouper: STC was used by the academic cluster engine Grouper which processed data obtained directly from HuskySearch meta-search engine. In 2001, Grouper was sold to the search engine MetaCrawler and it is no longer available on-line.

Carrot: Carrot is a recently proposed meta-search engine which uses STC. The original contribution was the introduction of three heuristics to reduce the number of clusters:

1. *Word overlap*: If a phrase has more than 60% of its non-stopped words appearing in another phrase of higher coverage, it will not be displayed. Coverage is the percent of all documents of the collection which are contained in a base cluster.

2. *Sub- and super-sentences*: STC often finds sentences that are sub or super strings of other sentences (for instance "George Bush", "Bush" and "President George Bush"). A sub-phrase always has a higher coverage than its super-phrase, but the super-phrase is usually more specific and therefore more informative. For each phrase, Carrot determines whether it has a sub-phrase or a super-phrase among the other phrases of the cluster. If it has no sub-phrase, it is designated as a more-general phrase; if no super-phrase exists, it is designated as a more-specific phrase. Phrases which are neither more specific, nor more general, will not be displayed.

3. *More general phrase with low coverage*: If a more-general phrase adds little to the coverage of its corresponding more-specific phrase, Carrot prefers more specific phrase as it is usually more informative - the more-general phrase will not be displayed then.

Carrot claims that *"The number of discovered base clusters (phrases) and therefore clusters grow proportionally to the number of retrieved results of a query. Since we do not want to replace browsing of a ranked list with browsing of clusters, some other presentation must be introduced to handle large document sets...A hierarchical presentation, proposed in [185] addresses this issue, but it is rather a patch of the flat STCs merging phase than an innovation to the algorithm. An ideal solution is part of Vivisimo, where the hierarchy is shown as a tree; currently STC has no option of hierarchical clustering."*. We will address this problem in its full generality in Chapter 4 where we will present an algorithm which overcomes the contiguity sentence limitation of STC and has a native hierarchical outcome.

Semantic hierarchical on line clustering (SHOC)

The task of sentence discovery from a text can be accomplished efficiently using the suffix tree data structure. However, the performance (time and space complexity) of the suffix tree is related to the size of the alphabet set.

SHOC is a system developed by Zhang and Dong [189] which uses a sentence discovery algorithm based on Suffix Array. Before entering in the details of the algorithm, we will introduce the indexing data structure it adopts.

Suffix array: Suffix array (SA) is an effective and elegant data structure, proposed by Manber-Myers in [144]. It stores the starting positions of all suffixes of the input string S, sorted lexicographically (symbols are numbered from 0). In practice, a suffix $s[i]$ is typically denoted by a 4-byte integer, i. Figure 2.7 gives the suffix array for the input string $s =$ "mississippi" (we have also listed the suffixes, even if the suffix array does not store them). Note that, a suffix array can just be seen as the list of the labels associated to the leaves of the suffix tree built on a given input string. Leaves are visited using in depth first. Hence, a suffix array can be built in linear time by using the linear time suffix three construction. Recently there were three different proposals for building a suffix array in linear time directly (the smartest one is [121]). Actually the lower bound is $\Omega(n \log |\Sigma|)$ where Σ denotes the alphabet. Manber and Myers reported that suffix arrays are an order of magnitude more efficient in space than suffix trees even in the case of relatively small alphabet size ($|\Sigma| = 96$) (this dimension is easily overcome when the Σ consist of words, instead of single chars, like in the creation of the GST). The longest common prefix (LCP) array is used to accelerate searching operations. An LCP array stores $n+1$ integer elements, where lcp[i], $1 \leq i \leq n-1$, indicates the length of the longest common prefix between $s[i-1]$ and $s[i]$, and lcp[0]=lcp[n]=0. The LCP array can also be constructed with O(n) time complexity [147]. The lcp is used to search for substring in O($p + occ$) time, where occ is the number of occurrences of pattern P in string S. An efficient implementation of the suffix array has been given in [148].

10	7	4	1	0	9	8	6	3	5	2
i	ippi	issippi	ississippi	mississippi	pi	ppi	sippi	sissippi	ssippi	ssissippi

Figure 2.7: Suffix Array for the string "mississippi$".

SHOC algorithm is scalable over the alphabet size and is able to avoid extracting meaningless partial phrases. The key idea of SHOC is to build the suffix array SA and the LCP arrays for document d_i, and to extract key sentences taking advantage of these two data structures. The cluster hypothesis exploited by SHOC is that documents sharing a complete substring are eligible to belong to the same cluster. We will briefly review the algorithm: Given the following definitions, authors proved the below theorem. The theorem implies that every right-complete substring (including complete substring) of the text T ($|T| = n$), can be identified by the position of a pair of adjacent suffixes in the suffix array SA. Note that, there are a maximum of $n-1$ right-complete substrings, even though there are $\frac{n(n+1)}{2}$ substrings of T.

Definition: Suppose a substring S occurs in k distinct positions $p_1, p_2, \ldots, p_k$ in the text T, S is complete if and only if the (p_{i-1})-th token in T is different of the (p_{j-1})-th token for at least one (i, j) pair, $1 = i < j = k$ (called "left-complete"), and the $(p_i + |S|)$-th token is different of the $(p_j + |S|)$-th token for at least one (i, j) pair, $1 = i < j = k$ (called "right-complete").

Definition: Assume RCS is a right-complete substring of T, the identification code of RCS is $ID(RCS) = \min\{w | 1 \leq w < N$, the LCP of $s[w-1]$ and $s[w]$ is RCS$\}$

Theorem: A substring S of T is right-complete if and only if there is a w ($1 \leq w < N$) and S is the LCP of $s[w-1]$ and $s[w]$.

Based on the above theorem, Zhang and Dong proposed a linear time algorithm `discover_rcs` to extract all right-complete substrings of T and meanwhile count their frequencies. A complete substring should be both right-complete and left-complete. To discover all the left-complete substrings, SHOC just applies the `discover_rcs` algorithm to T^R, the inverse document T. If S is a right-complete substring of T^R, then S must be a left-complete substring of T. Another algorithm `intersect_lcs_rcs` is used to find strings which are both right and left complete. It has linear time complexity in the size of the number of the left-complete and right-complete substrings.

Another contribution of SHOC is the application of LSI decomposition to the set of extracted complete-substrings. We remeber that the STC algorithm simply groups documents sharing a common phrase into one cluster. However, such lexical methods can be inaccurate and incomplete. Since there are usually many ways to express a given concept, the literal terms may not match those of a relevant document. In addition, most words have multiple meanings, so the terms will literally match terms in irrelevant documents. According to Zhang and Dong, a better clustering approach should run on a semantic level relying on LSI decomposition. Unfortunately, there is extensive literature which shows that LSI is a rather expensive tool which cannot easily be applied to on-line processing [160]. The final contribution of SHOC is the introduction of a phase which aims to create a hierarchy among the clusters. The proposal relies upon the evaluation of the clusters' cardinalities.

Lexical affinities clustering

Marek, Fagin and others [143] tried to improve precision over recall using a different approach compared to that based on the extraction of sentences (where a sentence is an ordered sequence of contiguous words). In fact, in this case the indexing unit consists of a *pair of words* that are linked by a lexical affinity (LA). An LA between two units of language stands for a correlation of their common appearance. One key advantage of LAs over sentences is that they represent more flexible constructions that link words not necessarily adjacent to each other. A good description of LA is given in [142], where it is shown how LAs can be extracted from a text at a low cost via a 5-word sliding window technique. The clustering phase is realized using a variant of the classical HAC approaches.

Marek and Fagin compared the advantage of using LA versus single words for clustering using a variant of the information-theoretic entropy measure. This variant [62] penalizes both highly heterogeneous clusters and a large number of clusters (note that pure entropy measure will select clusters made by singletons, which is not good). The tests performed over the Reuters21578 dataset [139] clearly show that using LAs improves the quality of the hierarchy by approximately 30%, compared with the same algorithm which uses single words as document features. This experiment was carried out after that the direct children of the root were removed from the resulting hierarchy before the quality measure was applied.

One limitation of LA's approach is that clusters are eventually annotated with non contiguous words (such as "chip;intel", "bush;president") which are less intelligible than longer sentences (such us "Intel produces chips for PC", "Bush, 43-th President"). In Chapter 4 we address this issue, by proposing a unifying approach which integrates the benefits of LA with the use of longer sentences as they are typical of STC.

Frequent itemset hierarchical clustering (FIHC)

Frequent Itemset Hierarchical Clustering (FIHC) [82] is another approach, proposed in 2003, that we consider interesting. The innovative contribution is that FIHC exploits the idea of frequent itemset (a sub algorithm of the apriori ones [4]) to discover terms which are common to documents. The idea behind frequent itemset is that a sequence of items (words) is frequent if all possible sub-sequences are frequent. The assumption made is that documents under the same topics are expected to share more common itemsets than those under different topics.

The algorithm works in four phases: First, it starts with a "bag-of-words" representation of the documents, and reduces this high dimension space by considering only those terms which are

frequent items. Note that, this first step will assign documents to several clusters. In order to control the overlaps, a second phase is adopted to prune the clusters according to a score function which gives preference to those documents which have many global frequent itemsets that appear frequently in the selected cluster. The surviving clusters will adopt k-itemset as their cluster label. The third phase aims to build a hierarchy from the surviving clusters. FIHC uses a bottom-up approach. At level k, The cluster C_k can have as its parent one among the $(k-1)$-clusters. These clusters have the label being a subset of C_k's cluster label. The best parent is chosen using a TDIDF score function. The fourth phase aims to prune the tree if it is broad and deep.

This algorithm has the advantage of being amongst one of the first which introduced some forms of approximation to select indexing units as bases for clustering. But this approximation breaks the power of expression provided by ordered sentences, as they are typical of STC. In Chapter 4 we address this issue, by proposing a unifying approach which will group sentences which are semantically similar but not syntactically the same, since some terms are missing.

IBM Web Snippet Clustering

Recently, IBM [129] proposed a system that constructs the folder hierarchy based on the minimization of an objective function similar to the one we will uses in Chapter 4. However, their labels frequently consist of single terms, in other (few) cases they are contiguous sentences. In fact, during the phase of feature extraction, either noun phrases or single words are extracted. In turn, single words are either nouns or adjectives. Then, the algorithm defines a mathematic objective function which takes into account six intuitive properties:

- **Property 1: Document Coverage**. Ideally, the documents in the collection covered by the taxonomy should be maximized. A document is said to be covered by a taxonomy if it lies in (or is assigned to) at least one of the clusters in the top level of the taxonomy.

- **Property 2: Compactness**. Ideally, the taxonomy should be as compact as possible to effectively summarize the different topics covered in the search results.

- **Property 3: Sibling Node Distinctiveness**. At any level of the hierarchy, the sibling concepts should be as different as possible from each other to minimize ambiguity.

- **Property 4: Node Label Predictiveness**. The node labels should be chosen such that they are good indicators of the documents they contain.

- **Property 5: Reach time**. Ideally, the user should be quickly able to locate search results interest within the hierarchy.

- **Property 6: General to specific**. The most general concepts should be associated with the hierarchy's root, and the node label of any node should be more specific (less general) than that of its parent and at the same time it should be less specific (more general) than those of its children.

The problem of taxonomy generation has been formulated as the problem of finding a permutation that optimizes the defined objective function. Discovery is a heuristic, greedy algorithm which has been presented to optimize the objective function. Complexity is $O(kn)$ where, k are the number of concepts extracted from the search results.

We must point out that the analyzed sentences are just the noun phrases, while in Chapter 4 we provide a unifying model to cluster search results which shares sentences that are "approximately" the same. In addition, we explore a line of research which suggests exploiting the clusters selected by the users in order to personalize the search results.

Microsoft Web Snippet Clustering

Microsoft, is the only major search engine which has clearly stated its effort on Web snippet clustering. In fact, a public demo of a beta system prototype [216], and a beta browser toolbar have been provided. The demos are an evolution of the work presented in [187].

Their system extracts (contiguous) sentences of a variable length via regression on five different measures. However, the clustering does not produce a hierarchy of clusters, but the search results are organized in flat groups. The phase of label extraction is based on five different properties:

- **Property 1: Phrase Frequency / Inverted Document Frequency (TFIDF)**. This property is calculated using a TFIDF measure.
- **Property 2: Phrase length (PL)**. This is simply the count of words in a phrase. Generally, a longer label is preferred for users' browsing.
- **Property 3: Intra-Cluster Similarity (ICS)**. ICS is used to measure the compactness of clusters containing the sentence. First, the snippets are converted into vectors in the vector space model, where each component of the vectors represents a distinct words and is weighted by TFIDF. For each candidate cluster, the centroid is computed, and ICS is calculated as the average cosine similarity between the snippets and the centroid.
- **Property 4: Cluster Entropy (CE)**. Cluster entropy is used to represent the distinctness of a phrase. For a given phrase w, the corresponding document set $D(w)$ might overlap with other $D(w_i)$ where $w_i \neq w$. Cluster entropy is defined as

$$CE(w) = -\sum_t \frac{|D(w) \bigcup D(t)|}{|D(w)|} \log \frac{|D(w) \bigcup D(t)|}{|D(w)|}$$

- **Property 5: Phrase Independence (PI)**. A phrase is independent when the entropy of its context is high (i.e. the average of left and right contexts are random enough).

Once the above proprieties are considered, a random variable $x = (TFIDF, LEN, ICS, CE, IND)$ is mapped to a real-valued y score, by using either linear regression, logistic regression, or support vector regression. y is then used to sort salient keywords in descending order, thus the most salient keywords are shown on top. Regression is based on a training dataset of 30 queries, selected from one day's query log from the MSN search engine, and evaluated by 3 human evaluators. Candidate phrases, ordered alphabetically, have been manually extracted and have been shown to the evaluators. There were one or two hundred candidate phrases for each query. The three evaluators are asked to first browse through all the search results returned by our system, and then select from the candidates 10 "good phrases" (assign score 100 to them) and 10 "medium phrases" (assign score 50 to them). The scores of the other phrases are zero. Experiments are run on 10 queries and show that the IND, LEN and TFIDF are more important than other properties. The coefficient of Intra-Cluster Similarity (ICS) is even negative, which also indicates that the content-based similarity has a small impact on salient phrase ranking. In addition, performance of linear regression, logistic regression and support vector regression with linear kernels are almost the same, showing the linearity of the problem. The complexity of the algorithm is not given, but experimental results show an almost linear behaviour.

We should point out that the system is no longer available for testing due to its commercialization. A key difference with our SnakeT, discussed in Chapter 4, is that it produces flat clustering. The authors claim that, "*We believe a hierarchical structure of search results is necessary for more efficient browsing.*". This is actually what our system does. Another limit of their system is the necessity of an error prone training phase with examples labeled by users. This training phase could not scale with the size of ever changing Web. SnakeT does not need such training and offers many other important features, such as the use of clustering to personalize the search results.

Chapter 3

An overview of Web Ranking

Why is the alphabet in that order? Is it because of that song?, Steven Wright

Abstract

Due to the size of the Web graph, modern search engines have taken a lot of effort to rank Web objects and to provide valuable access to the information contained on the Internet. This Chapter is devoted to studying the ranking algorithms which exploit linkage information among Web pages. We will discuss Hits, Pagerank, Salsa, different form of ranking personalization and state-of-the-art solutions for accelerating Pagerank-like computations. We will conclude our discussion with on overview of recent efforts to learn a ranking function by using already collected click-through data.

Social network theory is concerned with features related to connectivity and distances in graphs, which can be applied to diverse fields such as epidemiology, human relationships and citation indexing, to name just a few. Social network theory is successfully used to study the Web graph, G_{Web}. Due to its size (see Chapter 1), modern search engines have spent a lot of effort to rank Web objects and to provide valuable access to the information contained in the Internet. This Chapter is devoted to the study of ranking algorithms. In Section 3.1 we will give some basic definitions and notations, and review certain features of the Web graph. In Section 3.2 we will discuss several metrics for ranking Web pages. Namely, in Section 3.2.1 we will introduce the Pagerank analysis used by Google, and in Section 3.2.2 we will assess the Hits method which has been extended and used by Ask Jeeves and then in Section 3.2.3 Salsa will be presented which combines aspects of both Pagerank and Hits. In addition, in Section 3.4 we will briefly consider the most recent evolution of the above-mentioned algorithms. Furthermore, in Section 3.5 we will discuss the most recent proposals for computing Pagerank, either in sequential or in parallel environments, over a massive dataset of Web pages or using advanced numerical and algorithmic methodologies. One of the most effective numerical and algorithmic methodology, currently available, is proposed by the author in Chapter 6. We will conclude our discussion in Section 3.6 with on overview of recent efforts to learn a ranking function by using already collected click-through data.

3.1 Basic definition and notation

Let $G_{Web} = (V, E)$ be the Web graph, where V represents the set of Web pages, $|V| = N$, and the direct edge (i, j) belongs to the set E if page i contains a link to page j. Let outdeg(i) be the outdegree of the page i, that is, the number of links which appear in page i, and let indeg(j) be the number of links which point to j. The graph can be written in matrix form using an adjacency matrix E_{ij} such that: $E_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$

A scalar λ and a vector $\mathbf{x}$, $\mathbf{x} \neq \mathbf{0}$, are an eigenvalue and a corresponding (right) eigenvector of a matrix M if $M\mathbf{x} = \lambda\mathbf{x}$. The multiplicity of λ is the number of linear independent eigenvectors associated with the same eigenvalue λ. In the same way, if $\mathbf{x}^T M = \lambda\mathbf{x}$, $\mathbf{x}$ is called the left eigenvector corresponding to the eigenvalue λ. Note that, a left eigenvector is a (right) eigenvector of the transpose matrix.

A matrix is *reducible* if a permutation matrix Π exists, such that $\Pi^T M \Pi = \begin{bmatrix} M_{11} & M_{12} \\ 0 & M_{13} \end{bmatrix}$. One method to test irreducibility is to consider the graph G as induced by the pattern of M, therefore, M is irreducible if, and only if, the underlying graph G exhibits strong connectivity. A matrix is *aperiodic* if, for all possible nodes $u, v \in G$, there are paths containing every possible number of links, with the exception of a finite set of path lengths which are missing. A matrix is *row-stochastic* if its rows are non-negative and the sum of elements in each row is equal to one. In this case, it is easy to show that there is a dominant eigenvalue equal to 1 and a corresponding eigenvector $\mathbf{x} = (c, c, \ldots, c)^T$, for any constant c. A very simple way of computing the dominant eigenpairs is the Power method [88] which is convergent for any choice of the starting vector $\mathbf{x}_0$ with non-negative entries for stochastic irreducible matrices,. A stochastic matrix M can be viewed as a transition matrix associated to a family of Markov chains, where each entry m_{ij} represents the probability of a transition from state i to state j. A finite $n \times n$ transition probability matrix $\mathrm{M} = [m_{ij}]$ is a stochastic matrix where m_{ij} is the transition probability of going from state i to state j. The period of a state i is defined as $p(i) = gcd\{n >= 1 : m_{ii}^n > 0\}$. A state i is aperiodic if its period is 1. A Markov chain is aperiodic if every state is aperiodic. According to the Perron-Frobenius Theorem for Markov chains [171] an irreducible, aperiodic, stochastic matrix M has a unique steady state distribution, that is, a vector π such that $\pi^T M = \pi$. This means that the stationary distribution of a Markov chain can be determined by computing the left eigenvector of the stochastic matrix M. The Perron-Frobenius theorem is used to prove the convergence of the Pagerank algorithm.

Some structural properties of the Web graph

In the last four years, the Web graph has been intensively studied, and one crucial result [32] is that it has a *bowtie* shape, as illustrated in Figure 3.1. The picture shows a core made by a large strongly connected component (*SCC*) and four sets of vertices which can be distinguished from their relationship with the core: upstream nodes (*IN*) can reach *SCC* but cannot be reached from *SCC*; viceversa, downstream nodes (*OUT*) can reach the core but the core cannot be reach them; the *Tendrils* nodes neither can reach nor be reached from the core (those tendrils leaving IN to enter in OUT are called *Tubes*); the last set is made up of many *disconnected components*, i.e., small groups of vertices not connected to the bow-tie.

It has been demonstrated [32, 93] that many features of the Web graph follow a power law distribution (i.e. a function such as $\frac{1}{k^\alpha}$), like in-degree and out-degree shown in Figure 3.2.

Haveliwala [117] has shown that each Web page p has a dominant number of outlinks which point to the same DNS domain of p. Therefore, a permutation of the link matrix where pages belonging to the same DNS domain are grouped together, will result in a block organization, as shown in Figure 3.3. This property has been used for accelerating the computation of Web ranking, as discussed in Section 3.5.

3.2 The mathematic behind traditional Web ranking

Here we describe the most important ranking algorithms given in literature: Pagerank, Hits and Salsa. We will adopt the notation of [37]. Nowadays, these Web metrics are considered traditional since there are many public studies about them. Anyway, it should be pointed out that modern search engines are supposed to use not just a single ranking algorithm but a combination of many algorithms, and, nevertheless, many of them still remain a deep industrial secret.

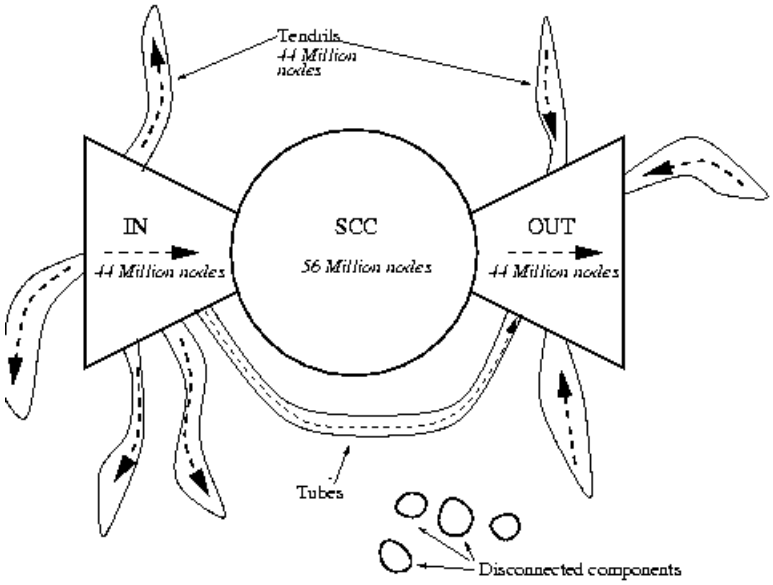


Figure 3.1: The bowtie structure of the Web graph.

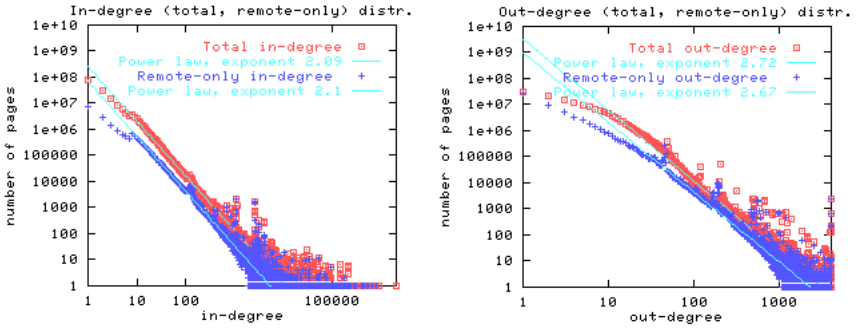


Figure 3.2: The power law features of the Web graph.

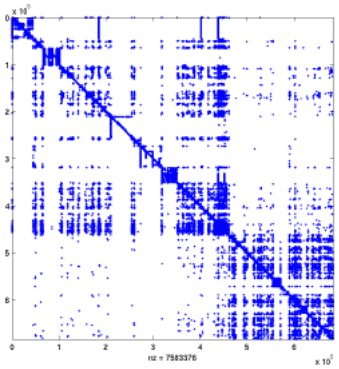


Figure 3.3: The block structure of the Web graph.

3.2.1 Pagerank

Pagerank is the ranking algorithm used by Google. Here we give the formal definition as reported in Brin and Page's seminal paper [27]. Consider the adjacency matrix $E[u, v]$ associated with G_{Web}. Let $z[v]$ a positive real number associated with the notion of *prestige* of v. Then the vector $\mathbf{z}$ denotes the prestige scores of all nodes and can be computed as the stationary vector associated with E. From the equation $\mathbf{z} = E^T\mathbf{z}$, we derive that the prestige of each node is equal to the sum of the its parents prestige. Indeed, the matrix notation can be re-written as $z_{i+1}[v] = \sum_u E^T[v, u]z_i[u] = \sum_u E[u, v]z_i[v]$. We must point out to the reader that we are not interested in the exact values of prestige assigned to each node, but rather, in the relative order assumed by them. From a mathematical perspective, this means that the values can be normalized using any convenient norm.

In order to guarantee the existence, and uniqueness of the fixed point for the equation $\mathbf{z} = E^T\mathbf{z}$, the matrix E must be modified to satisfy the hypothesis of the Perron-Frobenius Theorem. These conditions are not usually satisfied by the adjacency matrix induced by the Web graph, which is clearly *not* strongly connected and aperiodic. In fact, it contains several pages with no outlinks (the so called dangling links) and several cycles where there is a risk that the surfer may become trapped. The solution to this was proposed in 1998 by Larry Page and Sergei Brin [159]. Let us assume that the Web is strongly connected (i.e. the matrix is irreducible) and aperiodic, therefore, in order to force the stochastic property we can divide each row of the adjacency matrix by the outdegree of the corresponding node. Formally, we have $\mathbf{z} = P^T\mathbf{z}$ where $P[u, v] = \frac{E[u,v]}{\text{outdeg}(u)}$. This matrix displays an interesting stochastic interpretation in terms of a surfer who crawls the Web graph, uniformly picking a link at random on each page in order to move on to the next page.

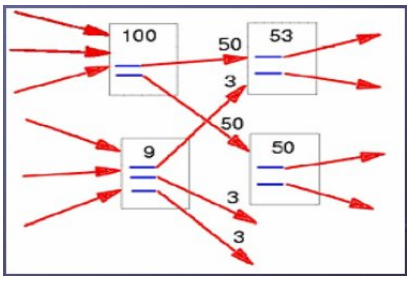


Figure 3.4: Pagerank: An example of link ranking.

The pages with no outlinks can be handled in two alternative ways. One solution is to remove them, thus reducing the size of computation. After computing the score of the surviving nodes, the rank is propagated to the nodes eliminated through the aforementioned pre-processing step. Note that this kind of pruning can result in the elimination of a huge amount of DAGs connected to the Web graph. A more detailed overview on how to handle the dangling nodes is proposed in Chapter 6.

The second solution to manage dangling nodes is to impose a random jump on every page, whenever a dangling node is reached. This solution leads to a new matrix $\bar{P} = P + D$, where D is a rank-one matrix such that:

$$D_{ij} = \begin{cases} \frac{1}{N} & \text{if outdeg}(i) = 0 \\ 0 & \text{otherwise,} \end{cases}$$

In order to solve the problem of the Web graph's reducibility and aperiodicity, Page and others proposed surfing the graph following hyper-links with probability α and, occasionally, *"to bother and to jump"* to a different place with probability $1 - \alpha$. α is called the "teleportation factor". Mathematically, this corresponds to the definition of a stochastic, irreducible, and aperiodic matrix $\tilde{P}$:

$$\tilde{P} = \alpha\bar{P} + (1-\alpha)\frac{1}{N}\,\mathbf{e}\mathbf{e}^T, \tag{3.1}$$

where N is the total number of pages in the Web graph, and $\mathbf{e}$ is the unit vector and so, the "random jump" is not biased by any particular node. Therefore, Pagerank assigns the ranks to Web pages by solving:

$$\mathbf{z}^T\tilde{P} = \mathbf{z}^T. \tag{3.2}$$

Page and others suggest setting $\alpha = 0.85$ (note that different values of α will result in different results for $\mathbf{z}$). Pagerank has its own interpretation in terms of Markov Chains, where a single random walk is made over the whole Web. According to the Perron-Frobenius Theorem, the principal eigenvector of an irreducible, aperiodic and stochastic matrix is actually the stationary distribution of the underlying Markov chain. In addition, the rank assigned to Web pages corresponds to the steady state distribution of the random walker visiting the Web graph.

We would like to underline that Pagerank assigns a vote to a page regardless of its content and, as a consequence, Pagerank is query-independent. For these reasons, Google uses Pagerank in association with other content-based criteria, such as the presence of query terms in sensitive parts of Web pages (title, anchors, etc), or the proximity among queried terms in the retrieved Web pages.

3.2.2 HITS

HITS [124] (Hyper-link Induced Topic Search) was developed by Jon Kleinberg at the IBM Almaden Research Center and is based on a clever intuition. We point out that an algorithm derived from HITS is currently used by Ask Jeeves, but there is no public information about the internal specifications. The Web is made up of two different types of pages: *authorities*, which contain a high-quality information, and *hubs*, which are pages with a long list of links pointing to the authorities. Hubs and Authorities are linked by a mutual reinforcement property: a good hub points to many good authorities, and a good authority points to many good hubs. Any page in the Web is associated with two different scores: an authority score and a hub score (note the difference to Pagerank, which assigns a single score to each page). The mutual reinforcement property can be mathematically expressed as:

$$\begin{cases} \mathbf{a} = E^T\ \mathbf{h} \\ \mathbf{h} = E\ \ \mathbf{a} \end{cases} \tag{3.3}$$

where E is the adjacency matrix of a graph built as follows: Given a query q, a set of n pages is retrieved by a search engine. This is the so-called root set R. The root set is expanded to the base set B considering all those pages which are pointed to by the pages in R, and all those pages which point to the pages in R. E is the adjacency matrix of the graph associated to B. Note that HITS is query-dependent (the opposite of Pagerank, which is query-independent), and that, by simple substitution, we obtain $\mathbf{a} = E^TE\mathbf{a}$ and $\mathbf{h} = EE^T\mathbf{h}$. Therefore, if $\mathbf{a}$ converges to a fixed point, this will be the eigenvector of the matrix E^TE. In the same way, the fixed point for $\mathbf{h}$ is the eigenvector of EE^T. Since $(E^TE)[v,w] = \sum_u E^T[v,u]E[u,w] = \sum_u E[u,v]E[u,w] = |\{u : (u,v) \in E, (u,w) \in E\}|$, the entry (v,w) in EE^T is the *co-citation* index of v and w. In the same way, we can show that the matrix E^TE is the *co-cited* matrix. An example of root set expansion and computation is shown in Figure 3.5.

Kleinberg has demonstrated two important features of HITS: If at each step the results are normalized, then equations (3.3) are guaranteed to converge using the *power iteration* method. Furthermore, in the presence of ambiguous queries such as "jaguar" or "python", and polarized queries such as "abortion" or "cold fusion", the highest-order eigenvectors of (3.3) will identify a bipartite sub-graph of R, which focuses on the weakest semantic meaning of the queried term. The latter feature is particularly useful for *"Topic Distillation"*, a clustering technique based on the structure of the links, as opposed to one based on the content of the Web pages. Note that HITS

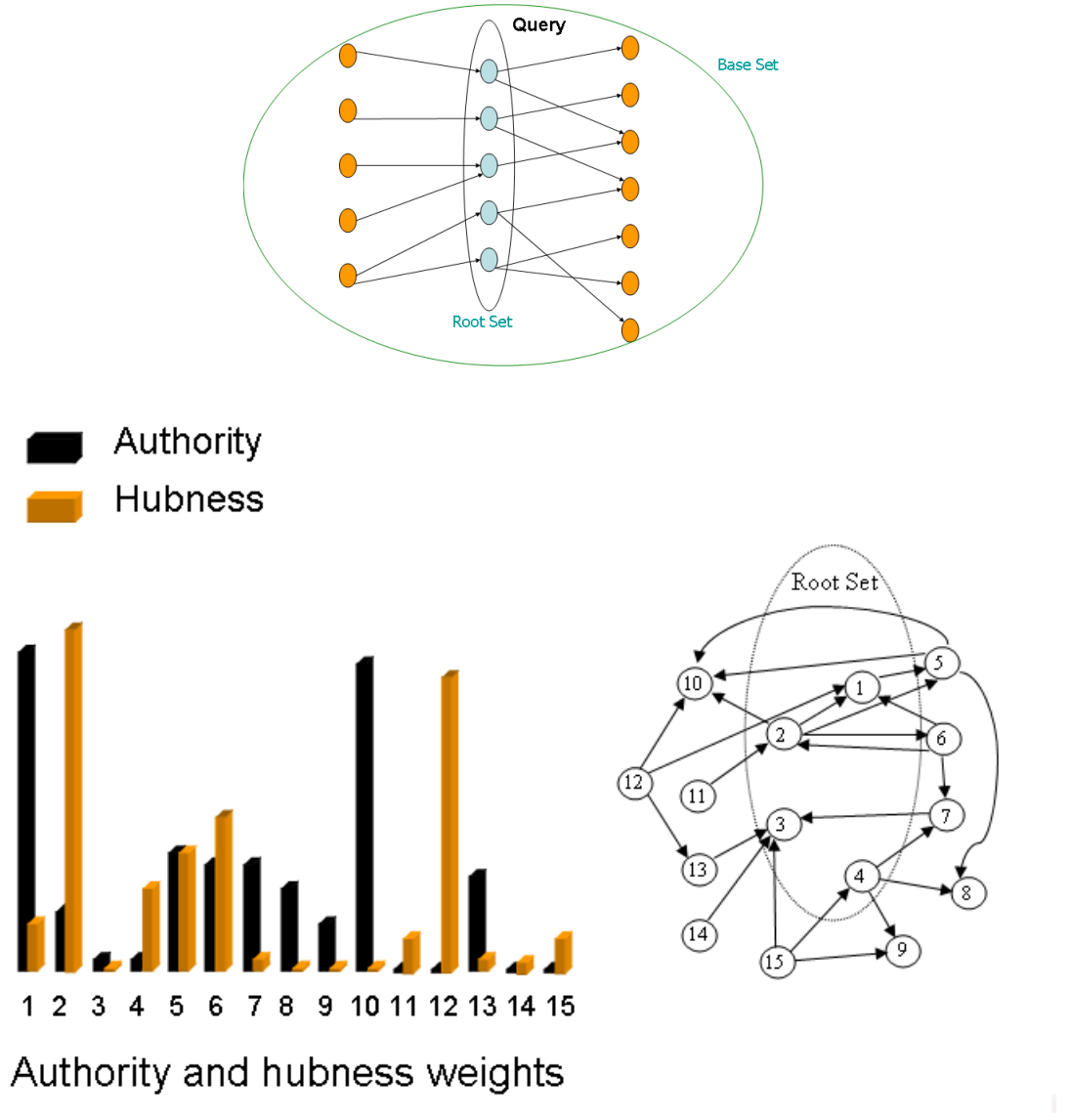


Figure 3.5: HITS: examples of base set, root set and computation.

can be seen as a particular kind of SVD decomposition (see subsection 2.1.3). In fact, the SVD decomposition can be applied to the HITS matrix EE^T, such that $EE^T = U\Sigma^2U^T$. This implies $EE^TU = U\Sigma^2$. This can be rewritten as $EE^TU(j) = \sigma^2U(j)$, where $U(j)$ is the j-th column of U. So $U(j)$ is the eigenvector of EE^T corresponding to the eigenvalue σ^2. Thus, HITS is an SVD decomposition of the adjacency matrix (source, destination), in the same way in which LSI is the SVD decomposition of the matrix (document, term).

The main problem associated with HITS is its lack of stability. Small alterations to the graphs associated with R will result in large changes to the hub and authority ranks assigned to each node. HITS needs to identify bipartite subgraphs of R in order to propagate scores for hubs and authorities until its convergence. It is sufficient to insert a single node, and therefore, two edges, to break the mutual reinforcement properties and significantly alter the final results. This phenomenon is frequent on the Internet. For example, let us consider a page redirection imposed

after moving a part of a Web site to another server. A further problem is that the results can be easily spammed. Indeed, let us imagine maliciously creating a hub page h which links to several good authority pages, then this h page will suddenly acquire an artificially high hub score which can be redistributed to other pages (forcing them to become artificial authorities). It was demonstrated in [155] that Pagerank is less sensitive to (small) changes in the Web graph, essentially in relation to the random jump introduced for forcing irreducibility. Another bad property of HITS is the so-called Tightly-Knit Community (TKC) effect, where a tightly-knit community is a small but highly interconnected set of pages. The TKC effect occurs when such a community scores high in link analyzing algorithms, even though the pages in the TKC are not authoritative on the topic, or pertain to just one aspect of the topic. HITS sometimes ranks the pages of a TKC in unfeasibly high positions.

3.2.3 Salsa

Salsa (Stochastic Algorithm for Link Structure Analysis) [138] is an approach proposed by Moran and Lempel to combine certain aspects of Pagerank with certain aspects of HITS, where the aim is to cast the bipartite reinforcement in a random surfer framework. The main goal is to reduce the bad TKC effect. Salsa works in two steps:

1. At each node v, the random surfer moves back to a node u which points to v. The choice of v is uniformly made at random in the set $\{v : (u, v) \in G_{Web}\}$.
2. From u the surfer uniformly takes a node w at random, such that the edge (u, w) exists.

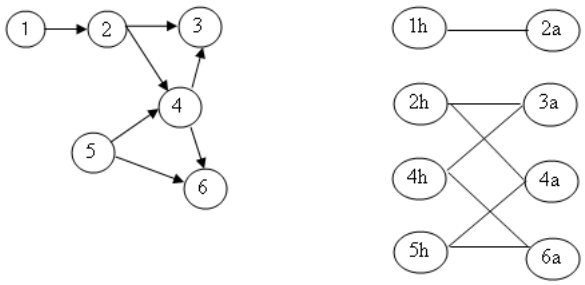


Figure 3.6: SALSA: construction of the bipartite graph.

The graph G, in Figure 3.6(a), is transformed into a bipartite undirected graph G_{bip} in figure 3.6(b). $G_{bip} = (V_h; V_a; E)$ is defined as follows:

$$\begin{cases} V_h = \{s_h : s \in G \text{ and outdegree}(s) > 0\} & \text{(the hub side of } G_{bip}) \\ V_a = \{s_a : s \in G \text{ and indegree}(s) > 0\} & \text{(the authority side of } G_{bip}) \\ E = \{(s_h; r_a) : s \to r \in G\} \end{cases} \tag{3.4}$$

where h denotes the hubs and a the authorities. Each non-isolated page $s \in G$ is represented in G_{bip} by one or both of the nodes s_h and s_a. Each link $s \to r$ is represented by an undirected edge connecting s_h and r_a. On this bipartite graph, Salsa performs two distinct random walks and each walk will only visit nodes from one of the two sides of the graph by traversing paths consisting of two edges in G_{bip} at each step. Note also, that every path of length two in G_{bip} represents a traversal of one hyper-link (when passing from the hub-side of G_{bip} to the authority-side), and a retreat along a hyper-link (when crossing in the other direction). This movement recalls the salsa dance and is the reason for the name given to the algorithm. Since the hubs and authorities of topic t should be highly visible in G_{bip} (reachable from many nodes by either a direct edge or by short paths), we may expect that the t-authorities will be amongst the nodes most frequently

visited by the random walk on V_a, and that the t-hubs will be amongst the nodes most frequently visited by the random walk on V_h.

Salsa examines the two different Markov chains which correspond to these random walks: the chain on the authority–side of G_{bip} (the authority chain), and the chain on the hub side of G_{bip}. It defines two stochastic matrices, which are the transition matrices of the two Markov chains. The matrices are defined as follows: let W be the adjacency matrix of the directed graph defined by G. Denote by W_r the matrix obtained by dividing each non–zero entry of W by the sum of the entries in its row, and by W_c, the matrix obtained by dividing each non–zero element of W by the sum of the entries in its column. Then, the hub-matrix H consists of the non-zero rows and columns of $W_r W_c^T$, and the authority-matrix A consists of the non-zero rows and columns of $W_c^T W_r$. SALSA ignores the rows and columns of A and H which consist entirely of zeros, since by definition, all the nodes of G_{bip} have at least one incident edge. The matrices A and H are irreducible, if G_{bip} is connected. Then, Salsa uses A and H to compute the ranks in a way similar to HITS.

Lempel and Moran have demonstrated that, according to a certain hypothesis, the Ergodic Theorem can be applied, guaranteeing convergence. The steady-state node probabilities of the authority-to-authority transition have a very simple form:

$$\pi_v = c_1 \text{ Indegre(v)}$$

And the steady-state node probabilities of the hub-to-hub transition have an analogous form:

$$\pi_v = c_2 \text{ Outdegre(v)}$$

where c_1 and c_2 are two constants. Therefore, SALSA converges on hub values proportional to the outdegree and authority values proportional to the indegree. Lempel and Moran have also demonstrated that SALSA is less sensitive to the TKC effect. Provided that cgi-link, ad-link and nepotistic links were removed manually, the experiments carried out by the authors produced good results. This requirement has made the use of SALSA rather difficult and, as far as we know, no commercial search engine is using SALSA as its ranking algorithm.

3.3 Personalized Web ranking

Using a personalized Web ranking algorithm, the Web pages returned as search results for a user U_1 are not the same set of pages returned to the user U_2, even when U_1 and U_2 submit the same query. The goal is to provide search results which vary according to different behaviour, interest or tastes implicitly or explicitly expressed by the users. As an example, one user can be interested to the helicopter "apache", another in the native Americans "apache" population, and yet another in the "apache" web server.

The industrial scenario: This consists of on a beta version by Google [192] based on [111]. In addition, Google allows [224] the users to search also their own Web-search history by offering some additional information about the frequency and the last visit of each search result. A similar service is offered by Yahoo [193], Ask Jeeves [194], and A9.com [208]. Another interesting proposal is Eurekster [221] which relies on patented learning search technology and patent pending processes that link search algorithms to social networks. However, all of these approaches either need to maintain up-to-date user profiles, possibly defined on a restricted (tiny) set of alternatives, or they require an explicit `login` which allows the underlying search engine to track the user behaviour.

The scientific scenario: The need of search personalization is due to the different types of queries submitted to the search engines. In fact, literature offers many studies on different types of queries submitted to search engines by the users. A seminal study has been presented in [29]. The author creates a taxonomy of intents that express different search goals. A query can either be informational, as in traditional IR, or transactional, to express an "intent to perform some

web-mediated activity", or navigational, to express the need to reach a web site. A related work is [174] which points out that users often reach their search goal via Web pages identified by using less precise queries but more easily to specify. An interesting work is [175] which investigates the different goals of people when they use search engines. It appears that people express different opinions on how relevant they considered search results to be. The author states that, while current Web search engines do a good job in result retrieval, they do not do a very good job in discerning the search goals of every individual.

Different search goals are a good reason to study personalized ranking algorithms. Available literature offers different solutions to this problem. The first approach to personalization was discussed by Brin and Page [27]. In this work, it has been suggested to start the computation by considering that some Web pages are "more important" than others, thanks to factors which are not directly related to the pure link analysis. In fact, Page claims that *"rather than distributing the random linking probability equally among all nodes, it can be divided in various ways over every site .... For example, it could be distributed so that a random jump only takes the surfer to one of a few nodes with a high importance, and to any of the other nodes"*. Personalized Pagerank can easily be described by modifying the equation (3.1) as follows:

$$\tilde{P} = \alpha(P + \mathbf{d}\mathbf{v}^T) + (1 - \alpha)\ \mathbf{e}\mathbf{v}^T, \tag{3.5}$$

where $\mathbf{v}$ is a non-uniform probability vector.

Other proposals have been given in the literature. For instance, [101] uses the pages contained in the Web-directory Dmoz to modify the random jump of Pagerank towards a specific topic. The limitation of this approach is that it is necessary to compute a different Pagerank value, for each Web page, and for each possible topic. A partial solution to this "scaling problem" is given in [111], where the dependence on the number of topics was reduced to be sub-linear but still growing together with them. This approach focuses on user profiles. One Personalized Pagerank vector (PPV) is computed for each user. A set of pages H with high Pagerank is created. Then a subset of H is chosen by the users to create profiles. These subsets are called the preference sets or the base vectors. PPVs are expressed as a linear combination of base vectors. The advantage of this approach is that for a set of N pages with high page rank, one can compute $2N$ Personalized Pagerank vectors without having to run the algorithm again. This is an improvement in comparison with [101], where the whole computation must be performed for each biasing set. The disadvantages are that the users are forced to select their preference set only from within a given group of pages, which are common to all users. Another disadvantage is the relatively high computation time for large Web graphs.

[44] suggests gathering user profiles by exploiting the user navigation sessions captured by a proxy, and then [111] is applied. Another approach is [127] which suggests filtering the search engine results to a query, exploiting an analysis of the frequency in which the user asking the query has visited or selected the (neighborhoods of the) various Web communities in the corpus in the past. Recently [173] proposed personalizing the Web-search results by exploiting *click-through data*. This data records who submits the queries and which users have selected specific pages. Data was analyzed using an high-order SVD decomposition that automatically captures the latent relations between users, queries and Web pages. The limitation of this approach is that high-order SVD is very slow and it is not clear how to automatically determine the number of singular values used in the computation. In addition, we note that several methodologies have been proposed to implicitly track user activities, and to create information for enhanced Web searching, including the users query history [168], browsing history [152, 172], Web communities [127, 172, 79], and rich client-side interactions [14, 152].

These solutions have just partially achieved the features of adaptability, privacy protection and scalability, required by any good personalization service. In Chapter 4 we will present SnakeT a valid solution when (a) a flat list of search results in not enough, or when (b) link analysis methodologies tend to boost those Web pages related to the most popular meanings and penalize those pages related to less popular meanings, or (c) when users have difficulties in expressing their search needs. SnakeT is a WebIR tool that creates labeled hierarchies of search results. These

search results are then personalized according to the clusters selected locally by the users.

3.4 Extensions to the traditional Web metrics

The seminal works of Kleinberg [125], and Brin and Page [27], were followed by a number of extensions and modifications. These modifications try to increase the stability of ranking methods, and to introduce some temporal information in them.

Extensions to HITS: Bharat and Henzinger [19] considered an improvement to the HITS algorithm, called Topic Distillation, by using textual information to weigh the importance of nodes and links. [163] presented a variant of the HITS algorithm that uses random jumps, similar to SALSA. Another similar algorithm named Randomized HITS has been proposed by Ng et al. [155].

A different line of research exploits the application of probabilistic and statistical techniques for computing rankings. The PHITS algorithm [50] assumes a probabilistic model in which a link is caused by latent "factors" or "topics". They use the Expectation Maximization (EM) algorithm [60] to compute the authority weights of the pages. Hofmann [108] proposed an algorithm similar to PHITS which also takes into account the text of the documents.

Extensions to Pagerank: Tomlin [177] generalized the Pagerank algorithm to compute flow values for the edges of the Web graph, and a TrafficRank value for each page. An interesting line of research aims to combine Pagerank with temporal information [13]. On the Web, the temporal information for outgoing links is under the control of source pages and is, therefore, susceptible to "cheating". On the other hand, the incoming links reflect the attention a Web page has attracted and seem to be more democratic in their nature, they are also less susceptible to cheating. Among those incoming links, the link emanating from the random surfer's current position can be picked out and treated in a special way. These observations suggest that the probability of the random surfer choosing y when leaving his current page x is a combination of many factors: the freshness $f(y)$ of the target page y, the freshness $f(x, y)$ of the link from x to y, and the average freshness of all incoming links of y. In a similar way, the random jump probability of a target page y is a (weighted) combination of the freshness of y, the activity of y, the average freshness of the incoming links of y, and the average activity of the pages that link to y. All these considerations lead to a modified version of Pagerank which takes in account temporal information.

In addition, there are many Pagerank modifications which consider graphs with different levels of granularity (HostRank, Pagerank on host instead of Web pages) [45], or with different link weight assignments (internal, external, etc.).

Another interesting line of research has tried to avoid the use of an arbitrary choice for the teleportation factor in Pagerank. In particular, Boldi and Vigna [23] provide a closed formula for computing Pagerank using a Maclaurin polynomial of degree k. Therefore, the Pagerank of a page can be seen as a rational function of α, and this function can be approximated quite efficiently: this fact can be used to define a new form of ranking, TotalRank [22], that averages Pageranks over all possible αs: a form of ranking without damping.

3.5 Accelerating the computation of Pagerank

Recently, the research community has devoted increasing attention to reduce the computational time needed by Web ranking algorithms. In particular, many techniques have been proposed to speed up the Pagerank algorithm. This interest is motivated by three dominant factors: (1) the Web graph has huge dimensions and it is subject to dramatic updates in terms of nodes and links [156] - therefore the Pagerank assignment tends to become obsolete very soon; (2) many Pagerank vectors need to be computed when adopting strategies for collusion detection [190] (3) many different Pagerank values could be computed for addressing the need of personalized ranking. State-of-the-art approaches for accelerating Pagerank have gone in at least six different directions.

In addition, there is a more general need, since Pagerank has also become a useful paradigm of computation in many Web search algorithms, such as spam detection or trust networks, where the input graphs have different levels of granularity (HostRank) or different link weight assignments (internal, external, etc.). For each algorithm, the critical computation is a Pagerank-like vector of interest. Thus, methods to accelerate and parallelize these kinds of algorithms are very important.

At least six different methodologies have been discussed in literature for accelerating the computation of Pagerank. They are (see Figure 3.7):

1. The first approach tries to efficiently compute the Pagerank in the external memory, by using a sequence of scan operations on the Web graph [100, 40].

2. The second tries to fit the graph structure into the main memory by using compression techniques for the Web graph [24, 164].

3. The third approach is to exploit efficient numerical methods in order to reduce computation time, and such numerical techniques may be considered the most promising, having produced some intriguing results in the last few years [7, 32, 117, 119, 18, 118, 137, 52].

4. The fourth approach is to compute the Pagerank in parallel by taking care of problems such as load balancing, efficiency, and scalability [87, 86, 10].

5. The fifth approach is to update the Pagerank periodically [42], by re-using the results of the computation over the previous snapshot of the Web graph.

6. The sixth approach tries to approximate Pagerank [41, 131, 33].

In the next Section we will discuss a naive method for computing Pagerank, which works quite well when there is enough internal memory. In addition, we will review the aforementioned methodologies.

Study	External Memory	Compression	Numerical	Parallel	Update	Aggregation
[100]	+					
[40]	+					
[164]		+				
[7]		+				
[32]			+			
[117]			+			
[119]			+			
[18]			+			
[118]			+			
[137]			+			
[52]			+			
[87]			+	+		
[86]			+	+		
[10]				+		
[42]					+	
[133]					+	
[132]					+	
[41]						+
[133]						+
[33]						+

Figure 3.7: Taxonomy of current methodologies for accelerating Pagerank computation

Computing with a naive sequential algorithm. Hereafter, we will follow the notation of [40]. Assume that the Web graph exists in the form of two files: a url file where the URLs are stored in lexicographical order, and a link file containing every edge in arbitrary order, where each edge is a pair of integers referring to line numbers in the url file. The Pagerank algorithm does not have access to the url file at all. In fact, each iteration for the computation of Pagerank can be described (see Equation. (3.2)) as a vector-matrix multiplication between a vector containing the rank values of the nodes before the iteration (also called source S) and the adjacency matrix, resulting in a

vector of new rank values after the iteration (also called destination D). The link file in Figure 3.8 contains one record for each node, identifying the node, its out-degree, and every destination node which is linked to it; the file is sorted by the line number of the source node. In addition, there are two vectors of 32-bit floating point numbers, a source vector on disk containing the rank values before the iteration, and a destination vector stored in the memory which contains the end values.

At each iteration, the total amount of rank assigned to D parents in the previous step, is propagated to the vector destination D. Since both source and link file are sorted by the line numbers of the source nodes, each iteration can scan over both files in a synchronized fashion, adding additional rank value for each source and linking to the appropriate destination in the memory-resident destination vector. Finally, the destination vector is written out to be used as the source in the next iteration. The algorithm is stopped when the norm of the difference between the source and destination vectors is under a given threshold. Note that, the naive algorithm is very fast if the data files fit into the main memory. If this memory is not large enough to contain the destination vector, then the process of disk swapping will require additional time. We will discuss different methodologies for addressing this problem in the following Sections.

Source Node (4 bytes)	Outdegree (2 bytes)	Destination Nodes (series of 4 bytes)
0	4	12,17,23,42
1	3	18,70,267
2	6	3,100,330,450,555
....		

Figure 3.8: Link File

Computing with a compressed Web graph. In [24] Boldi and al. presented the state-of-the-art for compressing the Web graph. They discussed many techniques for reducing the space need in terms of both the URLs and the link file itself. [40] estimated that 8Gb of memory are enough to contain the entire compressed Web graph. Their evaluation was based on the assumption that each link requires 3.5 bits. Today, this estimation is obsolete. In Chapter 1, we estimated that the size of the Web is 11.5 billion of pages and there are estimations which claim an average of 7.2 outgoing links for a single page [128]. Therefore we need $11.5 \times 10^9 \times 7.2 \times 3.5$ bit, which is around 37 Gbyte of main memory just to store the Web graph.

Computing in external memory with more sophisticated algorithms. When internal memory is not enough to contain the destination vector, it is necessary to store it in the external memory. Haveliwala [100] proposed partitioning the destination vector into d blocks D_i, each fitting into the main memory, and to compute one block at a time. It would incur considerable further cost to repeatedly scan the large link file in this process, and so to avoid this, he suggested preprocessing the link file and partitioning it into d files L_i, where L_i only contains the links pointing to the nodes in block D_i of the destination vector, sorted by the source node as before. In each iteration, Haveliwala runs the naive algorithm on S and L_i to compute block D_i of the destination vector, for $0 < i < d$. Note that the source file has to be read d times in this scheme; this limits its scalability to massive datasets as the number d of partitions increases.

Chen et al [40] proposed a sort-merge algorithm which aims to partition the computation into repeated sorting steps. For each iteration of Pagerank, they transmit a rank value from the source to the destination of each link. This can be achieved by creating an 8-byte packet for each link that contains the line number of the destination and a rank value to be transmitted to that destination. These packets are then routed by sorting them according to the destination, and combining the ranks into the destination node. Owing to the large degree of locality in the Web graph [119], it is convenient to merge packets with the same destination, after they have been sorted and written

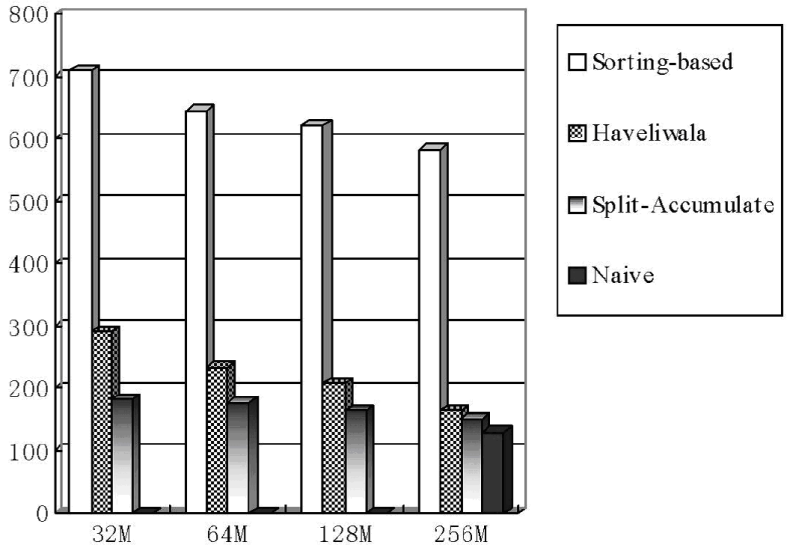


Figure 3.9: Comparing different algorithms in the external memory. The dataset is composed by 44.792.052 nodes and 652.901.912 links. The Naive Algorithm is the best choice when applicable (vector does fit).

out.

Another algorithm proposed by Chen et al [40], combines some of the properties of Haveliwala's algorithm and the Sort-Merge algorithm. In particular, it uses a set of link structure files L_i very similar to the Haveliwala's L_i. Instead of performing a local sort followed by a merge as in the Sort-Merge algorithm, it performs a split into buckets followed by an aggregation in a table (which could also be interpreted as a counting-based sort). The algorithm splits the source vector into d blocks S_i, so that the 4-byte rank values of all nodes in a block fit into the memory. The blocks S_i only exist in the main-memory, and the algorithm uses three sets of files L_i, P_i, and O_i, $0 < i < d$. File L_i contains information about all links with source node in block S_i, sorted by destination. Note that this is essentially the reverse of the files L_i in Haveliwala's Algorithm except that the out-degrees of all sources in S_i are not stored in L_i but in a separate file O_i that is a vector made up of 2-byte integers. File P_i is defined as containing all the packets of rank values with its destination in block D_i, in arbitrary order. In each iteration, the algorithm proceeds in d phases from $i = 0 \ldots d-1$. In the i-th phase, it first initializes all values in block S_i in the memory to $\frac{(1-\alpha)R(0)}{n}$, where $R(0)$ is the total amount of rank initially inserted in the Web graph. Then it scans the file P_i of packets with destinations in D_i, and adds the rank values in each packet to the appropriate entry in D_i. After P_i has been completely processed, it scans the file O_i of out-degrees, and divides each rank value into S_i by its out-degree. Then, the algorithm starts reading L_i, and for each record in L_i, consisting of several sources in S_i, and a destination in D_j, it writes one packet. This packet includes the destination node and the total amount of rank to be transmitted to it from these sources into output file P_j. The output file will become file P_j in the next iteration. The performance of the three external memory algorithms is compared in figure 3.9.

Computing by exploiting advanced numerical techniques. Many different numerical techniques are given in literature to accelerate the computation of Pagerank. Arasu et al. [7] have observed that computing Pagerank is equivalent to solving a linear system and have, therefore, proposed exploiting iterative methods such as the Gauss-Seidel algorithm. They also suggested reducing computation time by exploiting the "bow-tie" structure of the Web [32]. Kamvar et al. [117] have proposed accelerating computation by using extrapolation methods, which periodically subtract estimates of non-principal eigenvectors from the current iterate of the Power method. In [119]

the authors note that the Web graph's matrix, permuted by lexicographically sorting the URLs, assumes an approximate block structure since most of its links are intra-domains [18]. For this reason they suggest separately computing several "local" Pagerank vectors: one for each block, using the concatenation of these vectors as a starting point for the computation of a global Web rank. In [118], the author has proposed avoiding the re-computation of the (sub)-components of the Pagerank vector as soon as they arrive at a fixed point, and this results in an approximately 17% increase in speed. Lee et al [137] have obtained another significant result: They suggested splitting the Pagerank into two sub-problems. One is obtained by collapsing all Web pages with no out-links into a single (super)node, the other by combining all pages with outgoing hyper-links into a single node. The solutions to these two sub-problems are then combined to produce the Pagerank vector.

Computing by exploiting parallelism. Recently, a study performed by Yahoo! [87] suggested, independently by [52], that Pagerank can be successfully computed using linear system iterative solvers. They have developed a scalable parallel implementation and studied Jacobi and other subspace iterative methods. They showed that GMRES are the best choice of solution methods. Significantly different from [52], the main focus of Yahoo!'s study is on the parallel computation and on load balance. The parallel Pagerank codes use the Portable, Extensible Toolkit for Scientific Computation (PETSc) [207] to implement basic linear algebra operations and basic iterative procedures on sparse matrices. Another parallel implementation, which uses mpi and runs over 1.5 billion links, is given in [10]. They adopted a simple pairwise communication model to synchronize local Pagerank scores between processors. [26] claims that hypergraph-based partitioning substantially reduces communication volume over conventional partitioning schemes (by up to three orders of magnitude), while still maintaining computational load balance. A solution for parallel Pagerank computation on a gigabit PC cluster has been proposed in [10]. The work is more orientated towards the developing of an infrastructure of parallel solvers.

Computing by exploiting approximation. The first method to compute Pagerank with approximation has been discussed in [41]. The authors suggest expanding a small subgraph around a starting subset of the Web graph, placing an estimated Pagerank at the boundary nodes, and then re-using the standard algorithm. Lumpable Markov chains for approximating the solution of Pagerank have been discussed in [133, 122]. A graph aggregation method in which pages are partitioned into hosts is discussed in [33]. The authors suggest computing the stationary distribution in two steps, by combining the stationary distribution intra-host with the stationary distribution inter-host. In this model, departing from a page consists of a random transition to another page of the same host, followed by a Pagerank-like step from that page. Experiments conducted on a Web graph containing over 1.4 billion pages and 6.6 billion links showed that the resulting rank has a high correlation with the original Pagerank formulation. The advantage of this method is that it requires less than half the running time of a highly optimized implementation of Pagerank. Another approximation strategy has been given in [92].

Computing by exploiting updates. Futhermore, there is another set of techniques which address fast Pagerank computation, namely, given that the Web graph changes continuously, the Pagerank can be updated just for those nodes which are influenced by such a change. The updating problem is well-known for Markov Chains [133] and a first result is [42], which only works for hyper-link, and not for Web page modifications. Other interesting results are [133, 132] which use state-lumping technique for the stationary analysis of large Markov chains. It should be noted that even if the lumpable techniques are very interesting from a theoretical point of view, it is still an open research issue to find an easy way to partition a Markov chain and guarantee fast convergence. Besides, although there are some effective results [155] pertaining to Pagerank's stability in the presence of a slight perturbation of the Web graph, the results owing to Fetterly [77] and Cho [45] show that on the contrary, the Web graph is subject to rapid and *large* changes in terms of both nodes and links. State-of-the-art techniques for updating the Pagerank can be found in [20].

In Chapter 6 we will present our most recent results [52] based on numeric techniques. We will show how Pagerank computation in the original random surfer model can be transformed into the problem of computing the solution of a sparse linear system. The sparsity of this linear system has made it possible to exploit the effectiveness of Markov chain index reordering to speed up the Pagerank computation. In particular, we will propose rearranging the system matrix according to several permutations and we will apply different scalar and block iterative methods in order to solve smaller linear systems in a very time effective manner. We have tested our methodologies on real Web graphs crawled from the net. The largest one takes into account about 24 million nodes and more than 100 million links. Upon this Web graph, the cost of computing the Pagerank is reduced by 65% in terms of Mflops and of 92% in terms of time compared to the most common Power method.

3.6 Ranking algorithms based on query learning

There are many rumors about different ranking algorithms used by commercial search engines, but they have not been deeply studied from an academic point of view as they are not publicly available. The only remarkable exception is the algorithm RankNet, which was adopted in August 2005 by Microsoft for its MSN search engine. RankNet follows a line of research completely different from Pagerank, HITS, Salsa or other similar ranking algorithms which are based on link analysis. The key idea of RankNet is to have a training set of queries and to investigate them using gradient descent methods for learning ranking function. RankNet uses a probabilistic cost function which models the probability that a search result, in the training set, is judged "better" than another, by a group of human experts. The approach extends the use of back-propagation in neural networks to handle ordered pairs. This methodology is used to minimize the above cost function. Experiments have been run on a toy-dataset of about 17,000 queries where query-dependent features have been extracted from the query combined with four different sources: the anchor text, the URL, the document title and the body of the text. Some additional query-independent features are also used. For each query, a set of 1000 search results have been retrieved and up to 569 features have been extracted, many of which are counts. In addition, one or more of the returned search results had a manually generated rating, from 1 (meaning "poor match") to 5 (meaning "excellent match"), for each query. Unlabeled documents were given rating 0. The ranking accuracy is then computed as the function of the labeled search results, which are ranked in the top positions. Experiments have shown that the ranking accuracy is high and the time needed to learn this is acceptable, when a one or two layer neural network is adopted. The fundamental problem of RankNet is the construction of a large training dataset; this process requires intensive human effort, is error prone, and can yield inconsistency due to the fact that decisions and the quality of training items are personal and express time-varying needs.

The problem of optimizing search engines using click-through data, namely querylog of search engine in connection with the log of links the users clicked on, has been studied in [113, 162] where the learning retrieval function is computed using an approach based on support vector machine [37].

Chapter 4

Snaket: SNippet Aggregation for KnowlEdge exTraction

Where do I find the time for not reading so many books?, Karl Kraus

Abstract

In this Chapter we propose a (meta-)search engine, called SnakeT (SNippet Aggregation for Knowledge ExtracTion), that queries more than 18 commodity search engines and offers two complementary views on their returned flat results. One is the classical ranked list and the other consists of a hierarchical organization of results into labeled folders created on-the-fly at the query time. Users can browse the hierarchy with various goals: knowledge extraction, query refinement, and personalization of search results. The key idea is that a user issues a query using SnakeT, gets back a labeled folder hierarchy built automatically on the top of hundreds of search results. Then, (s)he selects a set of folders whose labels (themes) best fit his/her query needs. Given this selection, SnakeT personalizes the original ranked list by *filtering out* those results which do not belong to the selected folders on-the-fly. In this way, the personalization is then carried out by the people themselves

SnakeT is a complete system that offers both hierarchical clustering and folder labeling with variable-length sentences. SnakeT also offers a peculiar form of personalization that is fully adaptive, preserves privacy, scalable, and non intrusive for the underlying search engines. We have extensively tested all of SnakeT's features, and compared SnakeT against all other available Web snippet clustering engines. The final result is that SnakeT achieves performance close to the best known clustering engine, namely Vivisimo.com, and its form of personalization turns out to be much more effective and promising. SnakeT shows that a mutual reinforcement relationship between ranking and Web snippet clustering exists. In fact the better the ranking of the underlying search engines is, the more relevant the results are from which SnakeT distills the hierarchy of labeled folders. Viceversa, the more intelligible the folder hierarchy is, the more effective the personalized (re)ranking executed by SnakeT is over the original ranked list of query results.

The snippet hierarchical clustering idea has been independently considered by Vivisimo and more recently by Microsoft for commercial use. Nevertheless, they did not suggest using it for personalizing the Web search results. From a commercial point of view, we should note that it is still an open question to understand if the most users find clustering useful or if they prefer simply re-issuing a narrower query. The methodologies discussed in this Chapter have been published in [73, 74, 75, 76, 95].

4.1 Our contribution

The purpose of a search engine is to retrieve, from a given collection, the documents deemed *relevant* to a user query. This is known to be a very difficult task because of many reasons: *relevance* is a subjective and time-varying concept; the Web is heterogeneous, highly dynamic [156] and it is growing at a staggering rate [96, 70]; users have different expectations and goals— informative, navigational or transactional [29]— and they are lazy because they look mainly at the top-10 results, and moreover they often compose short and ill-defined queries. Although by now search engines have become very accurate with respect to navigational queries, for informational queries the situation is murkier: quite often the responses do not meet the users needs, especially for ambiguous queries. Third-generation search engines try to address this issue by focusing on the *user need* rather than the user query, and by offering various *post-search tools* that help the user in dealing with large sets of somewhat imprecise results. Examples include, but are not limited to, query suggestions or refinements (e.g., Yahoo and Teoma), result clustering and the naming of clusters (e.g., Vivisimo and Wisenut), and mapping of results against a predetermined taxonomy, such as ODP (the *Open Directory Project* used by Google and many others), Yahoo, and LookSmart.

In this Chapter we focus on "result clustering" which is one of the most promising IR-tools [153]. Result clustering was introduced in a primitive form by Northernlight and was then made popular by Vivisimo. The problem solved by these tools consists of clustering the results returned by a (meta-)search engine into a *hierarchy of folders* which are *labeled* with variable-length sentences. The labels should capture the "theme" of the query results contained in their associated folders. This labeled hierarchy of folders offers a complementary view to the ranked list of results returned by current search engines. Users can exploit this view by *navigating* the folder hierarchy driven by their search needs, with the goal of extracting information from the folder labels, or reformulating another query, or narrowing the set of relevant results. This navigational approach is especially useful for informative [29], polysemous and poor queries.

Result clustering is a challenging variant of classical clustering because of two demanding requirements:

- The folder hierarchy and its labels must be computed *on-the-fly* in order to adapt themselves to the different themes of the results returned by the queried search engine(s).
  Canonical clustering is instead persistent since "hierarchical structure is generated only once, and folder maintenance can be carried out at relatively infrequent intervals" [166].
- The construction of the labeled folder hierarchy must deploy the heterogeneous, uncontrolled, and poorly composed information returned by commodity search engine(s) for each query result. Namely, the URL, the title and the fragment of the result page, called *(Web-)snippet*, which summarizes the context of the searched keywords in that page. This stringent requirement is due to computational reasons, because it would be unacceptably costly to download the whole Web pages and make the labeled clustering from them. Given these features, the result clustering problem is also called *Web snippet clustering*, which is the terminology we adopt in our study.
  It is important to note that Web snippet clustering might be considered as a sort of document organization into topical contexts. But the documents here are the poorly composed snippets, without any associated *structural information* [161] or any available *static classification* [9, 39]. In this context any "fixed set of category labels" [105] would not be flexible enough to capture all the variegated snippet themes.

Nowadays, many industrial systems implement Web snippet clustering technology in their (meta)-search engines: Vivisimo, Mooter, Copernic, iBoogie, Kartoo, Groxis, Dogpile and Clusty. Their effectiveness has been recognized by SearchEngineWatch.com which conferred "the best meta-search engine award" to Vivisimo and Dogpile during 2001-2004.[1] In January 2005, the AOL portal adopted Vivisimo on top of the search results provided by Google. Google and Microsoft also seem to be interested in this IR-tool [234, 216] because *"clustering technology is the Pagerank of the*

[1] Dogpile has been granted licence to Vivisimo's technology.

future". Unfortunately, very little information is available about such industrial software. On the other hand, scientific literature offers several detailed solutions to the Web snippet clustering problem, but the performance of these solutions is far from the one achieved by Vivisimo.

Another approach to help users in searching the Web is the *personalization* of the ranked lists of query results. Personalized ranking combines Web-graph link information with some contextual/profiled information with the goal of achieving *adaptivity* and *scalability* to the variegated user needs. Examples of industrial personalized services are offered by Google [192], Yahoo [193], AskJeeves [194] and Eurekster. These approaches offer however a partial solution to the personalized ranking problem because they either allow profiles over a restricted (tiny) set of choices, or need to maintain up-to-date profiles which are a critical and private resource. In scientific literature the personalized ranking problem has also been investigated by proposing efficient scaling techniques to classical link-based ranking approaches. Yet these solutions are unappealing in practice because they ultimately need to compute a number of ranking values that are related to the number of user profiles, for each Web page (see e.g. [44, 111]).

The contribution of this Chapter is twofold: (i) we propose the first academic system for Web snippet clustering, called SnakeT (SNippet Aggregation for Knowledge ExtracTion), that achieves efficiency and efficacy performance close to Vivisimo; (ii) we exploit the labeled folder hierarchy of SnakeT to design an innovative form of personalized ranking that achieves full-adaptivity, privacy protection, and scalability. Overall this shows that a mutual reinforcement relationship between ranking and Web snippet clustering exists, from which both of them may benefit. In fact the better the ranking is, the more relevant the results (and thus their snippets) are from which SnakeT distills the hierarchy of labeled folders. Vice versa, the more intelligible the labeled folders produced by SnakeT are, the more effective the personalization of the ranked list of query results is.
In detail:

- We describe the anatomy of SnakeT, the first complete system in literature that offers both hierarchical clustering and folder labeling with variable-length sentences drawn on-the-fly from snippets. One specialty of SnakeT is that we use *gapped sentences* as labels, namely sequences of terms occurring *not contiguously* in the snippets. The second speciality is that we use a DMOZ ranker engine built off-line to dynamically rank the sentences extracted on-line from the snippets. The third speciality is that we propose a new hierarchical clustering algorithm based on a greedy function which takes into account the expressiveness and the rank of the labels and the compactness of the hierarchy. The known solutions and disclosed software do not address all of these features together, as we discussed in Section 2.4.
- We suggest using Web snippet clustering as a tool for personalized ranking. We will show how to plug SnakeT on top of any (*un*personalized) search engine in order to obtain a form of personalization that is fully-adaptive, privacy preserving, and scalable to an unbounded number of user needs. The key idea is that a user issues a query using SnakeT, gets back a labeled folder hierarchy, and then selects a set of folders whose labels (themes) best fit his/her query needs. Given this selection, SnakeT personalizes the original ranked list by *filtering out on-the-fly* the results which do not belong to the selected folders. We think that this is an innovative feature offered by SnakeT's interface because *it is able to dynamically adapt the ranked list of (about 200 or more) results to the local choices made by any user*. This approach of course does not require an explicit `login` by the user, a pre-compilation of a constrained user profile, a tracking of the user's past search behaviour, or a modification of the underlying search engines.
- In Chapter 2 we have provided a complete survey of all current literature and industrial systems, and in this Chapter we will compare SnakeT to the best systems available on-line by executing a large set of experiments. As a further contribution we have built, and offer [199] to the community, a benchmark dataset for evaluating Web snippet clustering engines. This dataset contains the results resulting from more than 18 search engines over 77 queries (i.e., about 200 results per query) which were selected from the top searched ones on Lycos and Google during 2004.

- Finally, we have implemented and engineered a public prototype of SnakeT that includes all the features above. It runs on top of more than 18 search engines working on Web, Blog, News and Books sources. It has an interface similar to Vivisimo (see Figure 4.2), available at `http://snaket.di.unipi.it/` (`http://snaket.com`).

Key Contributions
1. Gapped Sentences
2. DMOZ KB Ranker
3. Greedy Hierarchical Clustering
3. Evaluation functions for Clustering results
5. Exploiting the clusters selected by the users to Personalize the search results

Figure 4.1: Key contributions of this Chapter

In Section(2.2) a complete survey of all current literature and industrial systems related to Web snippet clustering has been provided. The system we will present in this Chapter is called SnakeT which belongs to the class of hierarchical clustering engines which uses sentences as an indexing unit. SnakeT differentiates itself from other engines for using approximate sentences, which will be introduced in the rest of the Chapter. SnakeT is highly engineered, available on-line, and it aims to overcome the limitations of the other systems given in literature (i) by using gapped sentences drawn from snippets as folder labels, (ii) by adopting special knowledge bases to rank and select the most meaningful labels, and (iii) by building a hierarchy of possibly overlapping folders. We will compare SnakeT to the best systems of the fourth class (see Figure 2.4) available on-line: namely CIIRarchies and Highlight. We have also tested Carrot2 for historical reasons: it offers the only available implementation of Grouper. We did not test the most recent results of [187, 129] because they didn't provide us with access to their software, and we could not repeat their experiments because their original datasets are missing and querying the same search engines would give different snippets now .

We extensively tested SnakeT by following three main methodologies proposed in literature: anecdotal evidence on the quality of the results, user surveys conducted on a set of queries, and some mathematical evaluations (see Figure 4.3). It is important to point out that there is not a general consensus about a mathematical measure for evaluating a Web snippet clustering engine. There are in fact many proposals for evaluating flat clustering [99, 151], but the definition of a function which is able to cleverly take into account the *expressiveness* of the labels within a folder hierarchy it is still an open problem [151]. In Section 4.4 we will evaluate SnakeT according to the first two methodologies above, and we will propose an extension to the methodology of [187] with the goal of addressing the issue of "label expressiveness". SnakeT will also be evaluated according to this last mathematical measure.

The rest of this Chapter is organized as follows. In Section 4.2 we will describe SnakeT's architecture and comment on algorithmic specialties underlying its design and implementation. In Section 4.3 we will discuss our proposal for personalizing the search results returned by a commodity search engine based on SnakeT's labeled folder hierarchy. Finally, in Section 4.4 we will present several experimental results, validate our design choices for SnakeT's architecture, and compare the performance of SnakeT against state-of-the-art solutions. The work is also described in the following papers [74, 75, 73, 76, 95].

4.2 The anatomy of SnakeT, the clustering engine

The architecture of SnakeT is detailed in Figure 4.4 where all of its modules and their interrelations are illustrated. We can actually identify the three main algorithmic phases in SnakeT's functioning:

Figure 4.2: Vivisimo (left) and SnakeT (right) on the query "*asthma*". Notice on top of SnakeT's window the buttons for invoking the personalized ranking.

	Anecdotal evidence	User Survey Number of Users	User Survey Number of Queries	Mathematical Measures
WebCat			10	F1
Retriever	+			
S/G	+	13	3	
Wang *et al.*				Accuracy
Grouper		7	3	Precision
Carrot	+	14	3	Click sequence length
LA				Entropy
Microsoft		3	30	P@N
FICH				F1
Lingo		7	8	
Credo	+			
IBM	+	20	10	Coverage of results
SHOC	+			
CIIRarchies		8	49	Expected mutual information
Highlight	+			Click sequence length
WhatsOnWeb	+			Cluster coherence
SnakeT	+	20	77	P@N

Figure 4.3: A summary of experiments conducted on Web snippet clustering engines.

sentence selection and ranking, hierarchical clustering and labeling, and personalized ranking. The first two phases will be detailed in this Section, personalization will be discussed in the next Section.

4.2.1 Two knowledge bases

SnakeT exploits two knowledge bases (hereafter KBs) which are built off-line, and are then used at query time for label selection and ranking. In the rest of this Section we will describe the construction and the properties of these two KBs, whereas their use is postponed to Section 4.2.3.

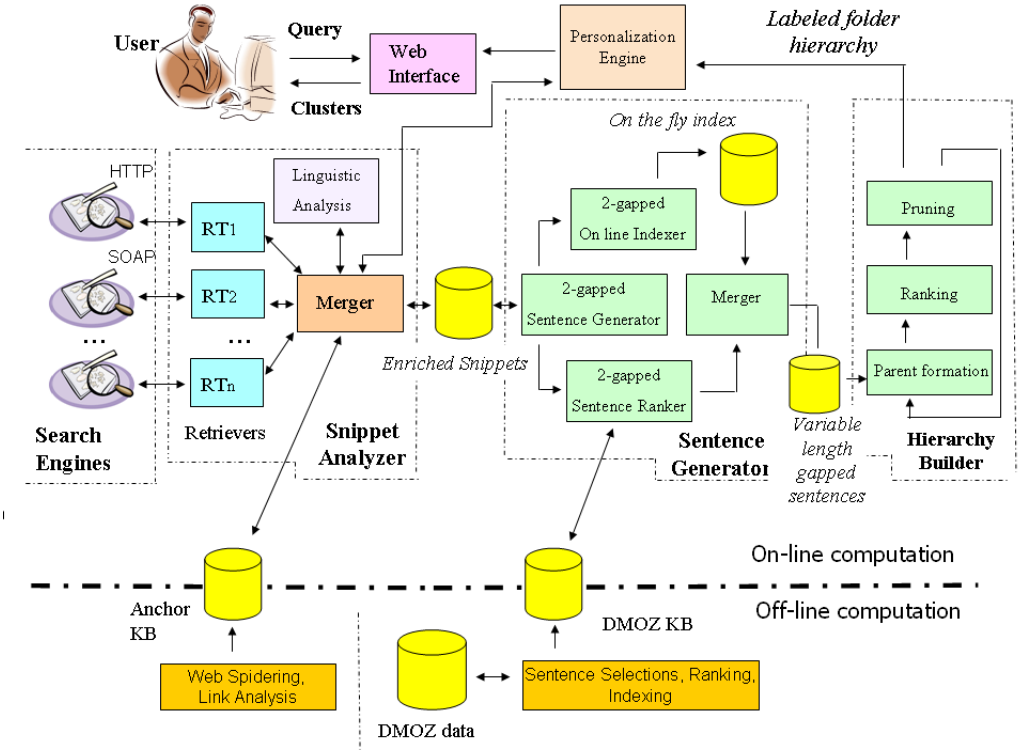


Figure 4.4: The architecture of SnakeT.

The Anchor Text Database

Several search engines exploit the hyper-links among Web pages as a source of information for ranking and retrieval. In this Chapter, we use this information for the *clustering and labeling of snippets*. To achieve this goal, we model each Web page p as a *virtual document* consisting of two sets of terms: The set $A(p)$ formed by the terms contained in p, and the set $B(p)$ formed by the terms contained in the (anchor) text surrounding each hyper-link that points to p (about 20 terms). We refer to $A(p)$ as the *content* of p, and to $B(p)$ as the *context* of p. The virtual document for p is then given by $A(p) \cup B(p)$.

The motivation underlying the use of $B(p)$ is that anchor texts often provide a precise, yet concise, description of the Web page p. In search-engine literature, the use of anchor texts is well established [102]. In fact, when the query "Madonna" is submitted to Google the first hit is the home page `http://www.madonna.com`, even if this page does not contain any textual information but just a shockwave presentation. Google claims that "These terms only appear in links pointing to this page". Unfortunately, deriving on-the-fly $B(p)$ from commodity search engines is computationally expensive since these terms do not occur in the returned snippets. Nevertheless, we wish to exploit the meaningful information offered by $B(p)$, and thus we have deployed the *Nutch's open source spider* [230] to collect more than 50 million Web pages selected amongst those which were top-cited during the crawling process. After the crawling phase, anchor texts referring to each page p are grouped together and indexed via a B-tree, called $\mathcal{A}_{\texttt{link}}$. We used the URL of p as the key for later retrievals. $\mathcal{A}_{\texttt{link}}$ is accessed through a memory cache and takes about 4,5Gb of disk space. At the query time, the content of the (poor) snippets referring to URLs indexed in $\mathcal{A}_{\texttt{link}}$ is enriched by exploiting the anchor texts contained in this index. In Figure 4.5 some examples of the anchor texts referring to well known URLs are shown: `www.sigir2003.org` and `www.celera.com`.

Anchor texts for the URL: `www.sigir2003.org`	# occurrences
SIGIR 2003 Home Page	4
SIGIR 2003	3
SIGIR	2
26th Annual International ACM SIGIR Conference	2
26th Annual International	1
Twenty-Sixth Annual International ACM SIGIR Conference Research and Development	1
SIGIR 2003 Conference Information	1
Anchor texts for the URL: `www.celera.com`	# occurrences
Celera	21
Celera Genomics	21
www.celera.com	14
Celera Genomics Applied Biosystems	1
CeleraScience	1
Celera Genomics group	1
Celera Corporation	1
Celera Genomics	1
Celera Genome Research	1

Figure 4.5: Two examples of queries to $\mathcal{A}_{\texttt{link}}$. For each queried URL we show the anchor texts referring to that URL, and the number times these anchor texts occur in the crawled pages.

The Semantic Knowledge Base

One of the key ingredients in effectively extracting meaningful folder labels is a method that *ranks* and *selects* a good set of candidate sentences drawn from the snippets. There are many proposals in IR literature for ranking individual terms, either based on precomputed archives (e.g. ontologies, dictionaries and lexical databases, like the ones provided by WordNet or AppliedSemantics) or on frequency information [183]. In the former approach, terms are weighted and related to each other by means of intensive human effort. The resulting ranking is of high quality but it does not scale and it is mainly static. In the latter approach, frequency information is automatically computed on top of an available index. This provides the so called `TFIDF` ranking usually adopted by modern search engines (see also [106]). The main drawback of frequency information is that it is computed over uncontrolled data containing several non meaningful terms, which is especially true on the Web.

We did not want to renounce the positive features of both rankings, and thus we designed a *mixed* ranking strategy. We indexed Dmoz, a directory freely available on the Web, and calculated an ad-hoc `TFIDF` measure for its controlled and high-quality lexicon. Actually Dmoz classifies more than 3,500,000 sites which are organized in more than 460,000 categories. The Dmoz project claims to *"provide the means for the Internet to organize itself. As the Internet grows, so do the number of net-citizens. These citizens can each organize a small portion of the Web and present it back to the rest of the population, culling out the bad and useless and keeping only the best content"*. It is therefore not surprising that many search engines, like Google, use Dmoz for ranking and retrieval. We are the first, to the best of our knowledge, to use the whole Dmoz for Web snippet clustering.[2] Specifically, we indexed this valuable archive by developing a *ranking engine*, hereafter denoted by $\mathcal{R}_{\texttt{dmoz}}$, based on inverted lists. The lexicon of $\mathcal{R}_{\texttt{dmoz}}$ consists of more than three million single terms. Upon $\mathcal{R}_{\texttt{dmoz}}$ we implemented a `TFIDF` measure that takes into account single terms, pairs of terms, and longer sentences. Moreover, unlike the standard `TFIDF` measure that is document centered, our measure is centered on the Dmoz-*categories*. Details are as follows:

Let $\#(w)$ be the total number of occurrences of the term w into Dmoz, $\#_C(w)$ be the number of Dmoz-categories in which w appears, and let $\#_C$ be the total number of Dmoz-categories. Moreover

[2]Unlike [39], we are using Dmoz only for the *ranking* of the sentences which are extracted on-the-fly from the snippets.

let $ns(C_i)$ be a boosting factor for the category C_i that takes into account its depth in the Dmoz-hierarchy— we are indeed postulating an increased importance for deeper categories since they are more specific. Let $b(w, C_i)$ be a boosting factor for the term w if it is placed in a *relevant* part of C_i, such as in its description or title— we are indeed postulating an increased importance of terms in crucial parts of the category's description. We then define

$$\mathrm{TF}(w) = 1 + \log \#(w), \qquad \mathrm{IDF}(w) = \log \frac{\#_C}{\#_C(w)}$$

and compute the rank of a term w, with reference to a category C_i, as:

$$\texttt{rank}(w, C_i) = b(w, C_i) * \mathrm{TF}(w) * \mathrm{IDF}(w) * ns(C_i)$$

The rank for a pair of terms (w_h, w_k) is then defined as:

$$\texttt{rank}(w_h, w_k) = \max_{C_i}\{ \prod_{r=h,k} b(w_r, C_i) * \mathrm{TF}(w_r) * \mathrm{IDF}(w_r) * ns(C_i)\} \tag{4.1}$$

Finally, the rank of a longer sentence s is defined by first identifying the set $P(s)$ of pairs of *contiguous* terms within s, and then by taking the sum of their ranks

$$\texttt{rank}(s) = \sum_{w_h, w_k \in P(s)} \texttt{rank}(w_h, w_k). \tag{4.2}$$

acm invitation to the 2003 conference
  acm invitation = 46
  invitation conference = 140
  [*total weight* = 186]
  [*avg response time* =0.16]

acm journal of experimental algorithmics
  acm journal = 412
  journal experimental = 268
  experimental algorithmics = 34
  [*total weight* = 714]
  [*avg response time* = 0.08]

Figure 4.6: Two examples of sentence ranking using $\mathcal{R}_{\texttt{dmoz}}$. We show the rank of every pair of adjacent terms in these sentences, the total rank of each sentence, and the time required to compute these ranks.

In order to speed up the on-line computations of SnakeT at query time, we used $\mathcal{R}_{\texttt{dmoz}}$ as follows. We pre-computed *off-line* the TFIDF rank of ten million *term pairs* lying within a proximity of four adjacent terms in some Dmoz-category description (skipping the stop-words). These ranks have been added to the index $\mathcal{R}_{\texttt{dmoz}}$, requiring about 500Mb of disk space. Any longer sentence s will be then ranked on-the-fly at query time by searching $\mathcal{R}_{\texttt{dmoz}}$ on the adjacent term pairs of s, and by adding their ranks. An example is given in Figure 4.6. In future we plain to integrate other sources in our semantic KB, such as Wikipedia.

4.2.2 The first module: the snippet analyzer

The task of this module is to forward the user query to a set of selected, remote search engines and to collect the returned snippets. SnakeT exploits Helios, an open-source meta search engine [95] also developed for this scope. Snippets referring to the same page (i.e. URL) are merged, then enriched with information retrieved from the $\mathcal{A}_{\texttt{link}}$ index, and finally analyzed syntactically and

linguistically. The collected information provides the so called *Enriched Snippets Collection*, and constitutes the input for the next *Sentence Generator* module. Let us detail the operations executed in this module.

Helios, the Meta-Search Engine: SnakeT fetches the snippets from more than 18 search engines, thus offering a large coverage of the Web [96]. They are A9.com, About, AllTheWeb, Altavista, E-spotting FindWhat, Gigablast, Google, Looksmart, MSN, Overture, Teoma, and Yahoo for the Web. Besides, we have Google News for news, a9 for the Amazon's book collection, and BlogLine for a collection of em blogs. All of these search engines are queried by the so called *Retrievers*, that return a ranked list of results (with their snippets). The retrievers exploit Helios, a full working open-source meta-search engine available at *http://www.cs.uiowa.edu/~asignori/helios/*. Helios is currently used by a number of academic research projects such as a personalized Web snippet clustering engine [76], a rank comparison engine [198], and an experiment for measuring the size of the Web [96]. Using Helios it is easy to plug-in a new engine. The meta search engine was intensively engineered to be efficient, light-weight, and thus usable on low cost platforms. Currently, the Retrievers request an average of 40 results from each search engine by using async I/O, for a total of 200 results. The average retrieval time is of about 2-3 secs.

Pre-Processing the Search Results: The index $\mathcal{A}_{\texttt{link}}$ is queried to retrieve the anchor texts pointing to each result page p. These anchor texts are used to enrich the description of p's snippets (see Section 4.2.1). The merged snippets and the anchor texts are segmented into sentences by choosing the punctuation symbols and HTML tags (such as title, description and text anchors) as separators. Subsequently, these sentences are processed by a *Linguistic Analyzer* which (i) filters a *stop list* of about 2000 words belonging to 12 different languages, (ii) stems the resulting sentences by using a variant of the Snowball stemmers [205], and finally (iii) extracts Part of Speech (PoS) and Named Entities [217] (see Figure 4.7).

The output of these phases is the so-called *Enriched Snippets Collection* that constitutes the input for the next *Sentence Generator* module. This module will deploy the enriched snippets in order to extract the candidate sentences for the labeling of the hierarchy folders.

URL: http://www.acm.org/sigir/

TITLE: information server ... sigir information server

SNIPPET: ... sigir-announce the official electronic mailing list of sigir. sig-irlist an electronic newsletter on ir-related topics. acm sigir Web site ...

TEXT ANCHORS: http://www.acm.org/sigir/ . ACM SIGIR . ACM SIGIR Home Page . ACM Special Interest Group on Information Retrieval (SIGIR) . ACM SIGIR Forum (Special Interest Group on Information Retrieval) . ACM . ACM Interest Group on Information Retrieval. ACM:SIGIR. Special . ACM SIGIR . Special Interest Group on Information Retrieval. ACM SIGIR Information . Special Interest . ACM's SIGIR ...

Figure 4.7: An example of enriched snippet referring to the Web page of `SIGIR2003`. A single "dot" separates anchor texts or sentences. Multiple dots denote something omitted for conciseness.

4.2.3 The second module: sentence selection and ranking

We are interested in folder labels that are long and intelligible sentences rather than single terms. Sentences are of course more useful for identifying the snippet themes. Grouper and other tools [186, 189, 181, 82] recognized the importance of sentences as folder labels, but their sentences were extracted from the snippets as *sequences of contiguous terms*. However, it is well known [186, 143] that the extraction of sentences formed by *non contiguous terms*, hereafter called *gapped sentences*, might boost the precision of the folder labels and, in turn, the usefulness of the folder hierarchy to

humans. For example, if a snippet contains the phrase "George W. Bush, the President of USA", its gapped sentences of length three are: "George Bush USA", "George Bush President", "Bush President USA" and all of their word-based permutations. To control the number of generated sentences, we have introduced some pruning strategies based on sentence rankings (detailed below). This fuzziness in sentence formation is useful to extract the same sentences from snippets which are semantically similar but not syntactically the same, since some terms are missing. It is not by chance that Vivisimo makes use of gapped sentences, as we checked from its functioning.

Therefore, we decided to develop a sentence generator module that constructs long gapped sentences, which are then used for folder formation and labeling. In this way, SnakeT overcomes the "contiguity limitation" of Grouper's labels and generalizes the notion of Lexical Affinities [143] to more than two terms. To the best of our knowledge, we are the first to design a Web snippet clustering engine that tries to group together snippets containing *almost the same sentences*. For efficiency reasons, our approach is syntactic and works in three phases.

First phase: formation of 2-gapped sentences. From each sentence s of an enriched snippet, we build the set $\texttt{AS}^2(s,d)$ formed by all pairs of terms that occur in s within a proximity d (like [143]). If a pair of terms (w_h, w_k) occurs in s, we insert it in $\texttt{AS}^2(s,d)$ together with (w_k, w_h). Some of the term pairs in $\texttt{AS}^2(s,d)$ could introduce artificial meanings. For instance, the sentence "Alexei Abrikosov, Vitaly Ginzburg and Anthony Legget received the nobel price ..." derives the 2-gapped sentence "Vitaly Legget" which is an artificial name. In order to circumvent this problem, we *discard* any pair whose terms appear contiguously neither in a snippet nor in the knowledge base from $\texttt{AS}^2(s,d)$.

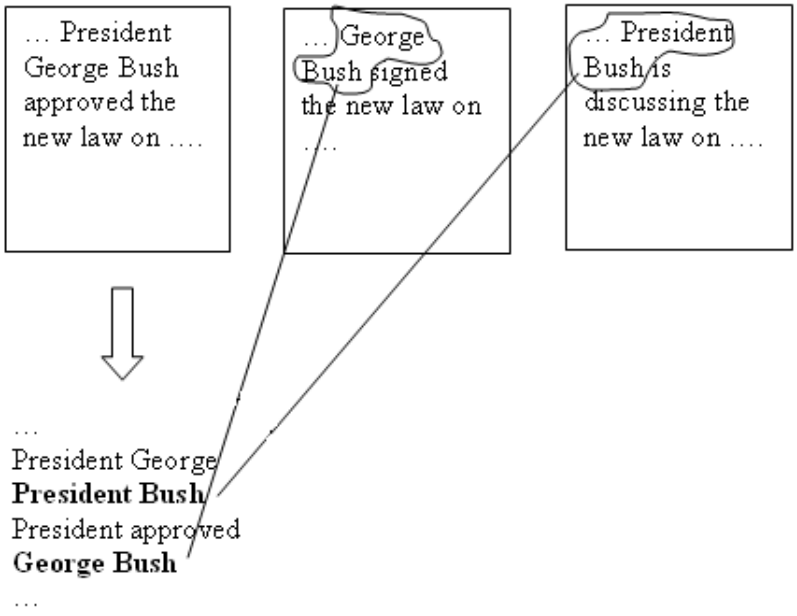


Figure 4.8: Generation of 2-gapped sentences. In this figure each box represents a snippet retrieved by search engines as an answer to a query. Considering the first box on the left, we have marked in bold the valid couples of words, among all those at distance $d = 2$. A couple of words is considered valid if it appears contiguously either in at least one snippet or in the knowledge base (KB DMOZ). The key idea is that we want to minimize the risk of introducing false positive, i.e. couples of words that do not exist in the search results provided by the search engines.

Second phase: selection and ranking of 2-gapped sentences. The 2-gapped sentences are ranked using *local* information and *background* information. Local information is a set of statistics

collected from the enriched snippets about a 2-gapped sentence, such as the number of occurrences of a 2-gapped sentence within the snippets, its HTML contexts and the rank attributed to the snippets by the queried search engines. Background information is a judgement on the relevance of a 2-gapped sentence derived from $\mathcal{R}_{\texttt{dmoz}}$ (see Section 4.2.1).

Technically, given a term pair $(w_h, w_k) \in \texttt{AS}^2(s, d)$ for some s, let $\#(w_h, w_k)$ be the number of times (w_h, w_k) occurs in the enriched snippets collection, and let $ws(w_h, w_k)$ be a boosting factor that assigns more weight to 2-gapped sentences occurring in top-ranked snippets. We compute the rank of (w_h, w_k) as a function of $\#(w_h, w_k)$, $ws(w_h, w_k)$ and $\texttt{rank}(\texttt{w}_\texttt{h}, \texttt{w}_\texttt{k})$. Since both (w_h, w_k) and (w_k, w_h) exist in $\texttt{AS}^2(s, d)$, we discard the one with lowest rank. In general, the 2-gapped sentences in $\texttt{AS}^2(s, d)$ having rank below a given threshold m are discarded; the remaining 2-gapped sentences form the set $\texttt{AS}^2_\texttt{m}(s, d)$.[3]

It is important to point out that the above process is a little bit more complicated when we use *stemmed terms*. In this case, since we still wish to build *unstemmed sentences* for labeling the folders, some more information must be kept. The top ranked, stemmed term pairs are associated to each unstemmed (w_h, w_k). This association is then exploited to reconstruct long unstemmed sentences for folder labeling.

2-gapped sentences	$\#(w_h, w_k)$	$\texttt{rank}(w_h, w_k)$ from Eqn. 4.1
data mine	172	532
knowledg discoveri	33	164
intern confer	15	304
confer data	15	192
intern data	15	144
mine softwar	9	426
siam confer	9	216
knowledg journal	6	212
...		

Figure 4.9: An example of top-ranked and stemmed 2-gapped sentences generated for the query "data mining".

Third phase: building longer gapped sentences. The 2-gapped sentences in $\texttt{AS}^2_\texttt{m}(s, d)$ participate in a merging process which aims to form longer gapped sentences. In order to efficiently perform the merging, we need to build *on-the-fly* an indexing data structure based on $\cup_s \texttt{AS}^2_\texttt{m}(s, d)$. We use neither suffix arrays nor suffix trees but just inverted lists indexed by the 2-gapped sentences. They are much more efficient to store and allow the fast construction of longer and intelligible gapped sentences. Specifically, any inverted list keeps detailed information for a 2-gapped sentence: the `snipID`, the subsection (e.g. title, description, link, category), the exact sentence number where a term pair (w_h, w_k) occurs, and the exact positions where the terms w_h and w_k occur. These inverted lists are sorted to exploit compression and to perform faster merging.

To build longer 4-gapped sentences we proceed as follows. We pick two 2-gapped sentences, say (w_h, w_k) and (w_x, w_y), and merge them if they have same `snipID` and occur in the same sentence within a fixed proximity window. The merging operates via a linear scan of the two sorted inverted lists associated with (w_h, w_k) and (w_x, w_y). For each created 4-gapped sentence, we build the corresponding inverted list which is similar in structure to that of a 2-gapped sentence.

Note that the 4-gapped sentences are indeed sentences composed by either three terms (merging (w_h, w_k) with (w_k, w_j)) or four terms (merging (w_h, w_k) with (w_i, w_j)). During the merging process we preserve the relative order of the terms as they occur in the snippets. All the 2-gapped sentences which do not participate in any merging are in some sense *maximal* and thus they are inserted in a candidate list of gapped sentences, hereafter denoted by CL. Finally, we rank any 4-gapped sentence by exploiting $\mathcal{R}_{\texttt{dmoz}}$ (see Section 4.2.1). Those sentences which have a rank below the threshold m are discarded.

[3]For memory reasons, SnakeT does not generate all the term pairs and then discards the low ranking ones, but executes both phases simultaneously. The discussion above is done only for the sake of presentation.

The process can obviously be generalized to obtain a 2^k-gapped sentence occurring in some snippets, for any given k. The idea is to create the 2^k-gapped sentences by merging pairs of 2^{k-1}-gapped sentences. The 2^{k-1}-gapped sentences that participate into any merging are in some sense *maximal*, and thus they are inserted in the candidate list CL. We finally rank any 2^k-gapped sentence by exploiting $\mathcal{R}_{\texttt{dmoz}}$ (see Section 4.2.1), and discard those sentences which have a rank below the threshold m. We stop this inductive process as soon as no pairs of gapped sentences may be merged further, or when k reaches a fixed value (currently we have $k = 3$).

At the end of the merging process, the set CL will contain sentences of variable length (at most 8 terms), sorted by ranks. These sentences will not contain stop-words and will be formed by (possibly) stemmed terms. To reconstruct meaningful unstemmed sentences we exploit the association maintained between the stemmed 2-gapped sentences and their originating top-ranked unstemmed sentences, as indicated above. These unstemmed gapped sentences will provide the *candidate labels* for the leaves of the folder hierarchy built by SnakeT.

4.2.4 The third module: the hierarchical organizer

SnakeT uses an innovative bottom-up hierarchical clustering algorithm whose aim is to construct a folder hierarchy which is *compact* in terms of the total number of folders, *balanced* in terms of subtree sizes, and *overlapping* in that a snippet may cover multiple themes and thus belong to many leaf folders. SnakeT also aims to assign folder labels that are *accurate* with respect to snippets' themes, *distinct* to avoid an overwhelming repetition in their constituting terms, and *intelligible* by means of variable-length sentences. The overall process is called *hierarchy builder* in Figure 4.4.

Initially, snippets are grouped into folders according to the (candidate) gapped sentences of CL they share. These folders provide the leaves of our hierarchy, and their shared sentences provide their labels (called *primary labels*). We are postulating that *snippets sharing the same gapped sentence deal with the same theme*, and thus must be clustered into the same (leaf) folder.

In order to agglomerate (leaf) folders for hierarchy construction, SnakeT enriches each folder C with a *set of secondary labels*, defined as gapped sentences that occur in a fraction c of the snippets contained in C (currently $c = 80\%$). The key idea is that primary labels provide a finer description of folders, whereas secondary labels provide a coarser description of folders. To manage primary and secondary labels efficiently, we concatenate them into a unique string and use a special character as a separator. This string is called the *signature* of the folder C, hereafter denoted by $\texttt{sig}(C)$, and the primary label $\ell(C)$ provides the annotating sentence of C.

The inductive step of the bottom-up hierarchy construction consists of three main phases: parent formation, ranking and pruning.

Parent Formation. A *parent* folder P is created for each group $C_1, C_2, \ldots, C_j$ of folders that share a substring among their signatures (hence, both primary and secondary labels are deployed). In this way a parent node captures a theme which generalizes the children nodes' specific themes in the hierarchy. The shared substring provides the primary label of P, and thus its annotating sentence $\ell(P)$. The set of secondary labels of P is formed by taking the secondary labels of the C_is that occur in at least $c\%$ of P's snippets. $\texttt{sig}(P)$ is finally obtained by concatenating the primary label of P with all of its secondary labels. We note that the efficient computation of the shared substrings could be carried out via a Suffix Array built over all of the folder signatures. Since $\ell(P)$ is computed among (primary and secondary) gapped sentences, it is a gapped sentence itself and it is not necessarily a substring of $\ell(C_1), \ldots, \ell(C_j)$.

Ranking. After the formation of all (candidate) parent folders and their labels, SnakeT ranks them by exploiting the rank of the labels of their children folders. Indeed, the rank of folder P is computed from the rank of the labels of the folders C_is by using $\mathcal{R}_{\texttt{dmoz}}$ (see Section 4.2.1). Given these ranks, SnakeT builds a *weighted bipartite* graph G in which the two sets of vertices are given by the (candidate) parent folders and their children folders, the edges of G denote the parent-child relationship between these folders, and the weight of a vertex is the rank of the corresponding folder.

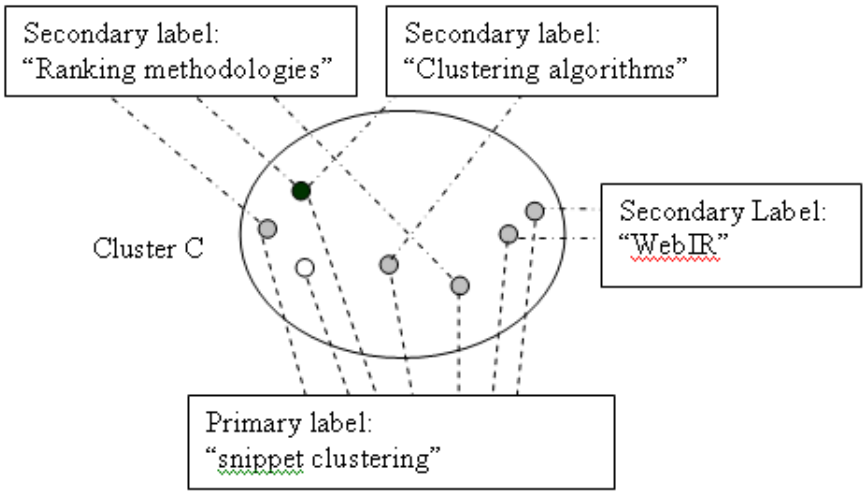


Figure 4.10: Primary and secondary labels. By definition, *all* the snippets in the cluster C share the label "snippet clustering". This is the so-called primary label, and it is a fine grain description of C itself. Different *subsets* of C may eventually share the so-called secondary labels. By definition, the secondary labels are all those which appear in at least the c% of the C's snippets. The intuition behind our algorithm is that the secondary labels provide a coarse-grain description of the cluster itself. In this example, the snippet represented by the white dot has just the primary label. Instead, the snippet represented by the black dot contains two labels, "Ranking methodologies" and "Clustering Algorithms", which can be considered as a secondary label for C. The snippets represented by the grey dots have just one other label seen as secondary ("Ranking methodologies", "WebIR" and "Clustering Algorithm" respectively). We used the informal expression "this cluster contains those label" to mean that the labels have been extracted from the snippets and ranked using the methodologies described in the Chapter.

Pruning. The graph G is exploited to clean up the (candidate) parent folders that do not match the goals stated at the beginning of this Section. The number of *redundant* parent folders is not negligible and their deletion contributes to make the overall hierarchy more intelligible. We formalize this problem as a *covering* problem and solve it via a greedy approach which takes into account the rank of the labels.

SnakeT adopts two rules to detect and discard the redundant (candidate) parent folders:

- **Context pruning rules.** This aims to discard parent folders which are redundant compared to a *graph-covering relation*: if two parent folders share at least sc% of their children folders, then SnakeT keeps the parent folder with the highest rank. sc% is a parameter currently set to 95%. Indeed, we want to minimize the impact of the parent folders which give access to a redundant amount information below in the hierarchy. In this way, it is not necessary for the user to take many alternative decisions which can lead to the same underlying information. Moreover, a parent folder is discarded if it has *more than* a fixed number of children (currently 20). The intuition is that we want to reduce the number of very large clusters which are not discriminative enough and do not help users in their choices. These rules take into account the maximum overlap between folders, the maximum size of the folders and the balancing of the hierarchy. They are similar to the rules proposed in [129], but here we use gapped sentences as labels, and apply the Content-pruning rule below to obtain better folder labels (see e.g. Figure 4.12).

- **Content pruning rule.** This aims to discard parent folders which are redundant with respect to a *syntactic-similarity relation* among their labels: if two parent folders are annotated with (almost) the same labeling terms, then SnakeT keeps the parent folder with the highest rank. This rule takes into account hierarchy compaction and overlapping folders.

After the redundant parent folders have been discarded, some of their children folders may turn out to have no link to any surviving parent folders. We do not want to loose the snippets belonging to those folders, so we proceed like Vivisimo putting these *uncovered* folders into a special folder "Other Topic".

After the pruning phase, the remaining parent folders provide the next level upon which the bottom-up process is repeated. This process is stopped after three levels have been built (a deeper hierarchy would be not user friendly). We remind the reader that our approach guarantees that a snippet occurs in many folders and this is consistent with the observation that a Web page can cover *multiple themes*. Moreover, we observe that the use of gapped sentences allows SnakeT to cluster together two query results even if some terms are missing in their snippets, or even if the terms in these snippets occur according to a different order.

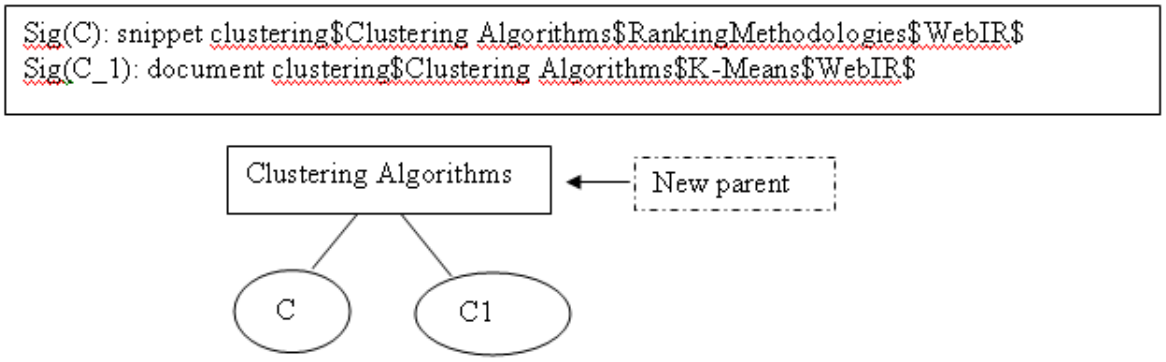


Figure 4.11: Hierarchical clustering. The clusters C and $C1$ share the secondary label "Clustering Algorithm", as you can see by analyzing their signature. We should remember that the secondary labels are a coarse grain description of the cluster itself, while the signatures SIG(C) are just a juxtaposition of primary and secondary labels where the labels' rank determines the position in the signature. The reason for this algorithmic choice is that the secondary label ("Clustering Algorithm", in this example) can be a good candidate – under certain conditions described in this Chapter – in order to create a father node P which has C and $C1$ as children. P's signature is built by combining C and $C1$'s signatures. We should remind that the set of secondary labels of P is formed by taking the secondary labels of C and C_1 that occur in at least c% of Ps snippets. SIG(P) is finally obtained by concatenating the primary label of P with all of its secondary labels. In this way, a hierarchy of clusters can be built with a bottom-up aggregation process. Therefore, we point out that the creation of the hierarchy is inherently driven by the (primary and secondary) labels dynamically extracted from the snippets.

4.2.5 Time and space complexity

Figure 4.13 is an overview of the asymptotic time complexity of SnakeT and the other known engines. Let n be the number of snippets retrieved from the pool of remote search engines, t be the number of formed 2-gapped sentences, m be the number of signatures extracted by SnakeT, ℓ_{max} be the maximum length of an inverted list of any 2-gapped sentences, and p be the maximum length of a signature. The Snippet Analyzer takes $O(n)$ time because each snippet consists of few terms and thus may be assumed to have a constant size (see Section 4.2.2). The Sentence Generator

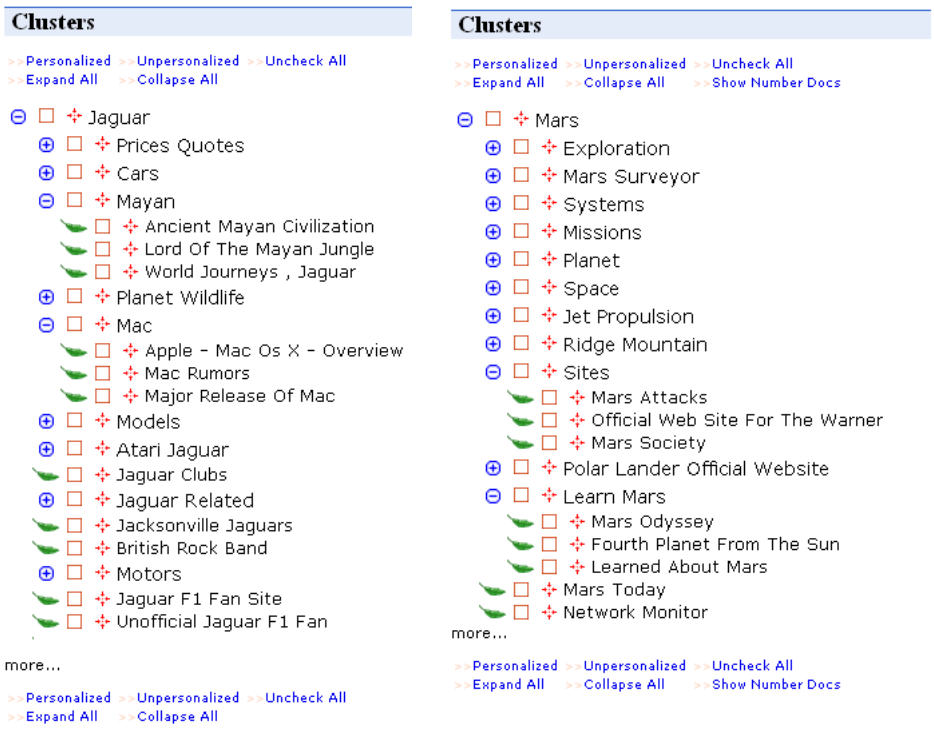


Figure 4.12: Examples of labeled folder hierarchies for the queries "jaguar" and "mars".

module takes $O(t \log t)$ time to sort the 2-gapped sentences according to their rank. Forming the 4-gapped sentences takes $O(t^2 \ell_{max})$ time in the worst case, because of the possibly merging of the $O(t^2)$ pairs of 2-gapped sentences, and the $O(\ell_{max})$ worst-case length of an inverted list. Forming 8-gapped sentences has cost bounded by that one (see Section 4.2.3). The Hierarchy Builder takes $O(m^2 p^2)$ time to look for the shared substrings between the m signatures of maximum length p. (Note that this cost can be reduced to $O(mp)$ by using a Suffix Array.) Moreover the greedy strategy used for pruning the redundant parent folders is linear in their number, which is then bounded by $O(m^2 p^2)$. Finally, by recalling that the hierarchy has at maximum depth of three and that the leaf formation is the most costly step because of the high number of folders involved, we conclude that $O(m^2 p^2)$ is the overall time cost of the Hierarchy Builder module (see Section 4.2.4).

Consequently, SnakeT takes time complexity bounded above by $O(n + t^2 \ell_{max} + m^2 p^2)$. In practice the parameters are small (now $n = 200$, $m = 200$, $p = 20$) and the operations involved are mainly based on sequential processing of lists, that makes them very fast. In Section 4.4 we comment on many experimental tests carried out on SnakeT over a large selection of queries. These tests showed that SnakeT achieves excellent results in clustering and labeling by requiring just few seconds (see e.g. Figure 4.14).

As far as the space occupancy is concerned, we note that SnakeT uses $\mathcal{R}_{\texttt{dmoz}}$ and $\mathcal{A}_{\texttt{link}}$ for sentence ranking and selection, thus requiring about 5Gb of disk space (see Section 4.2.1). At query time, SnakeT also needs to store the bitmaps and inverted indexes built on-the-fly over the Enriched Snippets Collection in internal memory. However, these data structures use negligible space given the short length of the snippets and the few other information introduced by Lexical Analyzer, Stemmer and PoS modules.

Flat	Time Complexity	Hierarchical	Time Complexity
Retriever	$O(nk)$	LA	$O(n^2 \log n^2)$
Wang	$O(kn \log n)$	IBM	$O(kn)$
Grouper	$O(n)$	SHOC	$O(n)$
Carrot	$O(n)$		
Microsoft	$O(n)$	SnakeT	$O(n + t^2\ell_{max} + m^2p^2)$

Figure 4.13: Time complexity of known Web snippet clustering engines. Here k denotes the number of generated folders.

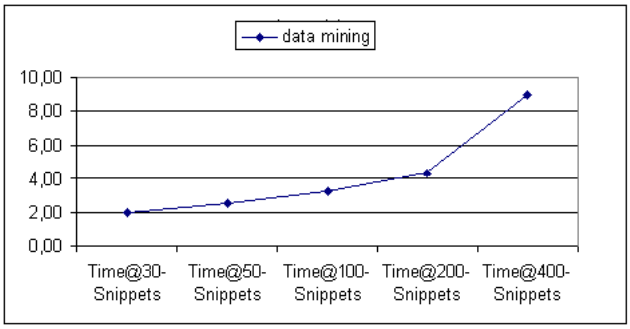


Figure 4.14: Time (in seconds) taken by SnakeT to retrieve and cluster a growing number of snippets on the query "data mining".

4.3 Personalizing search results

Link-based ranking algorithms tend to produce results which are biased towards the most popular meaning of an ambiguous query. At the time of this study, "Jaguar" on Google does not get answers related to the mayan civilization in the first ten results. Conversely, SnakeT is able to distill from the snippets *few key concepts* (possibly some of low rank, see Figure 4.12) that may be subsequently deployed by the user for many activities such as query refinement, disambiguation, knowledge extraction, and even *personalization* of the ranked list of results produced by the underlying search engines.

Nowadays, many search engines are putting a lot of effort into personalizing search results according to user behaviour. The traditional paradigm is to observe users web surfing and other habits and to infer a user profile. The main problem with the traditional approach to personalization is that user interests are not static, and evolve over the time, moving often from one topic to another. It has been noted [236] that this model's limitation in personalization is the need to maintain updated profiles, a problem which appears at least as difficult as the problem of ranking the Web itself. We believe that, in order to solve the problem, the key idea is to analyze the search results, show users the variety of themes therein, and let them explore their interests at that moment, which only they know. In other words, *the best personalization is carried out on-the-fly by people themselves*. By adopting the above idea, the traditional search paradigm is enriched and evolves into a more sophisticated "personalized navigational search".

We will now describe some benefits in adopting SnakeT.

Hierarchy browsing for knowledge extraction. Users can navigate through the hierarchy by expanding or collapsing on-the-fly its folders. The expansion is cheap since it occurs at the client side. This navigation can be seen as a type of *knowledge extraction* process that allows the user to acquire several points of view on the 200 or more query results fetched by SnakeT, without the effort of scanning all of them. This is useful because users frequently look at just the first top-ten

Different kinds of Search Personalization	Different Approach Issues
Explicit profiling [192] - Geographical localization - User selection of interests - ...	Cost of keeping the information updated
Mapping queries to predefined categories [39]	Classification scheme is fairly arbitrary and cannot scale with the size of Web
Monitoring search clients behaviour [176]	Not fully adaptive, privacy issues, scalability issues
Hierarchical Web Snippet Clustering and on-the-fly [76] Personalization of search results according to the selected clusters	Does not allow the exploitation of past preferences expressed by other users

Figure 4.15: The intuition behind search personalization is that people tend to be interested in topic areas. This observation provides a context for interpreting the search activities and can be used to improve the relevance of the results returned by the search engines. This can be achieved following different strategies. In a first scenario, an explicit profile can be built (for instance by associating an individual with a geographical area, or allowing users to explicitly select their interests). This scenario demands many efforts to keep the profile information updated. In another scenario, the search context can be built up mapping users's queries to a predefined taxonomy of topic but the choice of classification scheme is fairly arbitrary, and manual classification cannot be applied to the large and heterogeneous nature of the Web. In a more sophisticated scenario, the search clients behaviour is monitored over time and this provides the context for future queries. This approach is not fully adaptive and poses privacy and scalability issues. Our proposal follows a different approach which does not require the maintainance of users' profiles. In our approach search results are clustered and the personalization is achieved on-the-fly by exploiting the context provided by the clusters selected by the users. The only limitation of this approach is that it does not allow the exploitation of past preferences expressed by other users.

results of the ranked list, and they issue poor queries mainly consisting of at most two terms. See Figure 4.12 where a user learns from the folder labels created for the query "jaguar" that this term refers to: an animal, a car, the mayan civilization, a British rock-band, and Mac OS X. Another example detailed in the same figure is for the query "mars": SnakeT points out that it is the *fourth planet from the sun* and distinguishes between the film "Marks attacks" and the "Exploration" of Mars.

Hierarchy browsing for results selection. Users can narrow the ranked list of results to those ones which generate a label l, by just clicking on l. This is pretty similar to what Vivisimo does, with the speciality that SnakeT carries out the narrowing at the client side. In Fig 4.16 we show two examples for the queries "apache" and "soap". It is interesting to point out that, at the time of this study, the first ten results on Google do not contain links referring to "Indian Histories — Apache Tribe" or "Liquid Soap".

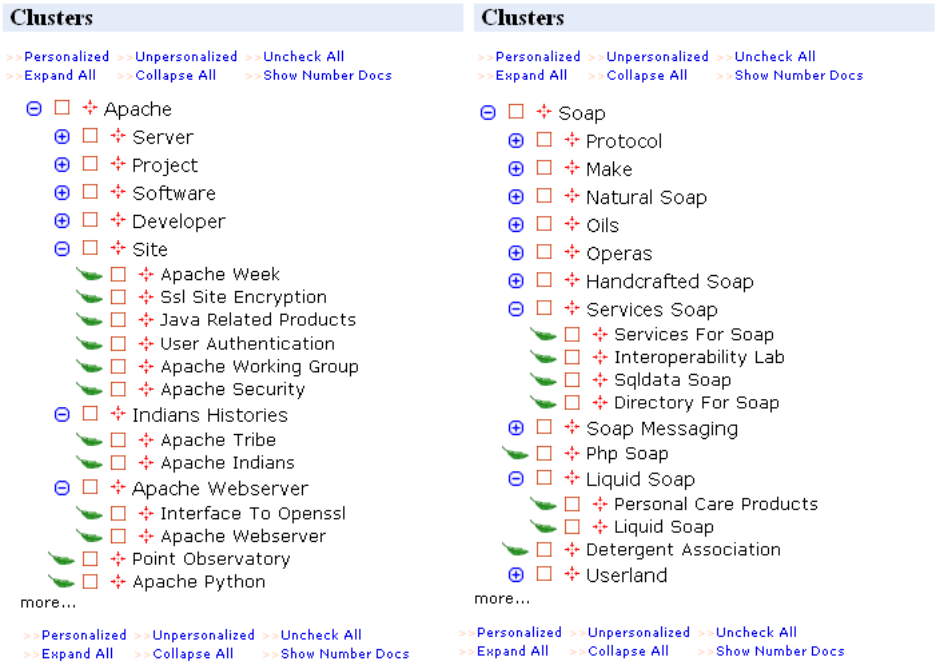


Figure 4.16: Examples of hierarchy browsing for queries “apache” and “soap”.

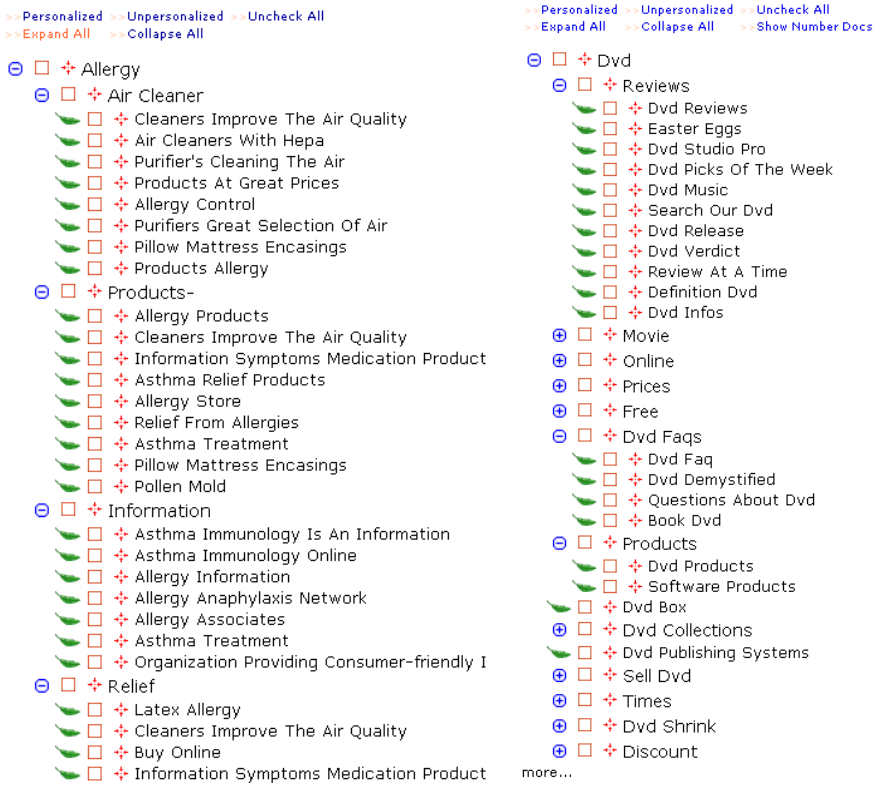


Figure 4.17: Examples of labeled folder hierarchy for the queries “Allergy” and “DVD”.

Query Refinement. Once the user looks at the folder hierarchy, (s)he can decide to refine the query Q in two different ways. Either (s)he can take inspiration from the folder labels to compose a *new* query and then submit it to SnakeT, or (s)he can click on a folder label l and then submit the *refined query* $Q \cup l$. See Figure 4.17 where the query "allergy" may be refined as "latex allergy" by clicking onto the label "Relief/Latex Allergy". Another example is "DVD" where the user can expand the query towards "Reviews/Easter Eggs" or "Dvd Faqs/Dvd Demystified". This is a form of *query expansion/suggestion* used by many commercial search engines, here re-interpreted in the Web snippet hierarchical clustering framework.

Personalized Ranking. When a user issues a query, SnakeT generates the labeled folder hierarchy and the ranked list of results. The former is computed as we largely commented on in the previous sections. The latter is produced by aggregating the query results provided by more than 18 search engines according to a *static rule* which weights these engines differently.[4]

Users can select a set of labels $L = \{l_1, \ldots, l_f\}$ by clicking on the checkbox associated to each of them. Then, they may ask SnakeT to *filter out* from the ranked list of results, returned by the queried search engines, the snippets which do not belong to the folders labeled with L. We think that this is the most innovative feature offered by SnakeT's interface, because it allows us to dynamically adapt the ranked list of (about 200 or more) results to the local choices made by any user. As far as we know, we are the first to suggest the use of Web snippet clustering technology as a tool for personalizing the ranked list of results returned by a (meta-)search engine. This feature turns out to be particularly effective when the users submit informative [29], polysemous, and poor queries. See Figure 4.19 for an example of personalized ranking in which a user aims to obtain introductory material about the programming language "java". First, the user formulates the query "java", and then selects the labels "Tutorials" and "Training" in order to obtain *personalized results*.

Figure 4.18: SnakeT on the query "java".

We refer the reader to Section 3.3 for a discussion on literature concerning personalization. We note here that SnakeT's personalization is fully adaptive, scalable, and non intrusive for the user

[4]We note that the design of a *good* aggregating rule is not the aim of this study, but the interested reader can refer to [69] and references therein.

Figure 4.19: Personalized SnakeT: the user selects the two labels "Tutorial" and "Training" and obtains his *personalized* ranked list.

and the underlying search engine. It is fully adaptive and scalable because it is not based on a user profile, and users can adapt the choice of their selected labels according to their subjective and time-varying interests. SnakeT also protects user privacy because it does not require an explicit `login`, a pre-compilation of a user profile, or the tracking of the user's past searches. Furthermore, a user can change the selected labels multiple times and thus modify on-the-fly his/her set of personalized results. The filtering is always done on all the (200 or more) snippets returned by the queried search engines. Everything occurs at the client side, thus being computationally cheap. To sum up, SnakeT is a plug-in that turns any un-personalized (meta-)search engine into a personalized one, in an non-intrusive way.

Personalized Web interface. We underline that SnakeT offers a lightweight Web-client interface that does not require maintaining any state on the server. For each query, the server performs the hierarchical clustering of the snippets returned by the queried engines and then sends the Web client all the information needed to perform the above tasks via a single communication in XML format. Folder expansion, browsing and personalization is scalable since they occur at the client side. Conversely, Vivisimo and WhatsOnTheWeb require communication between the client and the server for each navigational operation.

4.4 Experimental results

SnakeT runs currently on a commodity PC with Linux, P4 CPU and RAM 1.5Gb. Its Web interface is accessible at `http://snaket.di.unipi.it`.

The literature of Web snippet clustering offers three different methodologies to compare systems: anecdotal evidence for the quality of the results, user surveys conducted on a set of queries, and some mathematical evaluation measures. However, it must be said that there is not a general consensus for these latter mathematical measures. Although many proposals are known for evaluating clustering engines that produce flat hierarchies [99, 151], the definition of a measure which takes into account the *expressiveness* of the labels within a multi-level folder hierarchy is still an open problem. In the following, we will evaluate SnakeT by executing some user surveys, by drawing anecdotal evidence of the overall efficacy of its modules, and by extending the methodology of [187] with the goal of addressing the "label expressiveness" issue. These last two evaluations deploy a unique (in literature) dataset of snippets, enriched with clustering results, that we have

collected from our 18 search engines using 77 queries, selected from the top searched ones on Lycos and Google during 2004. This dataset is available on-line [199] and can be used freely by the research community either to reproduce our experiments or to test any new Web snippet clustering engine.

4.4.1 Users surveys

First Study: Is Web snippet clustering beneficial? This study was aimed at understanding whether a Web snippet clustering engine is a useful complement to the flat, ranked list of results offered by classical search engines. We asked 45 people, of intermediate Web ability, to use Vivisimo during their day-by-day search activities. After a test period of 20 days, 85% of them reported that using the tool provided "[..] a good range of alternatives with their meaningful labels", and 72% said that the most useful feature is "[..] the ability to produce on-the-fly clusters in response to a query, with labels extracted from the text". This study confirms the evaluation reported by SearchEngineWatch.com.

Second Study: SnakeT vs available clustering engines. We selected 18 queries from our dataset belonging to many different topics: iraq, bush, data mining, bill gates, last minute, car rental, mp3, divx, sony, final fantasy, ipo, equity, google ipo, warterloo, second war, aids, allergy, nasa. According to our discussion in Section 4.1, we asked three users to compare SnakeT's results against those provided by Mooter, CIIRarchies, Highlight, Carrot2. For a large part of the queries the users did not like Mooter, since it provides folders labeled with single terms. Carrot2 often tends to create a number of folders which exceeds the number of snippets, thus giving a negative impact on the usability of such software. Carrot2 also fails to cluster together similar labels such as "knowledge, knowledge discovery" and "mining and knowledge". Furthermore, it labels the hierarchy paths with sentences which are the substrings of the others, thus introducing little additional knowledge during user browsing. Highlight obtains its top-level labels by using classification, so that they are few and of little use. Moreover, its clustering frequently produces identical subtrees under different top-level categories and a number of folders which exceeds the number of snippets themselves (e.g 160 folders for the query "iraq"!). CIIRarchies provides good hierarchies but, as the authors admit, they are often not compact, have large depth and contain some non content-bearing terms which tend to be repetitive (See Section 2.4). From the point of view of the performance we remark that the two best available tools, CIIRarchies and Highlight, are significantly slower than SnakeT. We also remind the reader that no other system is available on-line for comparison, as discussed in Section 2.4, and that our three-user survey is fair because of the objectivity of their negative comments.

Third Study: SnakeT vs Vivisimo. We selected 20 students from the University of Pisa and asked them to execute the above 18 queries on these two engines. 75% of them were satisfied of the quality of SnakeT's folder hierarchy and of its labels. Figure 4.20 provides more details on this comparison. Hence we can state that SnakeT achieves performance close to Vivisimo.

4.4.2 SnakeT's dataset and anecdotal evidence

We have built a dataset made up of the snippets collected by our 18 search engines in response to 77 queries. This dataset is available on-line [199]. Queries selected were among the top-searched ones on Lycos and Google during 2004. For each query we retrieved between 180 to 220 snippets. The dataset has been manually annotated by humans who judged the relevance of the results with respect to the 77 queries. Moreover, the dataset has been enriched with the labels of the folders produced by SnakeT. We use this dataset to infer *anecdotal evidence* to the quality of the folder hierarchy produced by SnakeT, and to tune its software modules (Figs. 4.12, 4.17, 4.18, 4.19 are extracted from the dataset). As far we know, this dataset is the largest available on-line, and the only one built over queries retrieved by many search engines. It can be freely used by the research community either to reproduce our experimental results or to test any new Web snippet clustering engine. This dataset is crucial because Web snippet clustering is a form of ephemeral

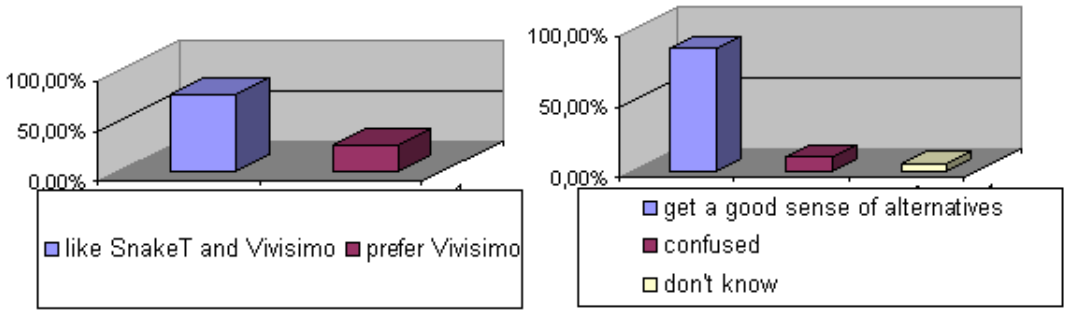


Figure 4.20: On the left, the judgement of SnakeT's results. On the right, we detail the user preferences.

clustering [143], and thus its results may change over time as a consequence of a change in the ranked list of results (snippets) returned by the queried search engines.

4.4.3 Evaluation of SnakeT

We ran extensive testing to tune up the different modules which SnakeT is made of. For the sake of presentation we report our main achievements here, which mostly agree with the whole set of results reported in [200].

We evaluated SnakeT using our dataset and a mathematical evaluation measure that extends the one adopted in [187] by taking into account the *expressiveness* of the labeled folder hierarchy. For each one of the 77 queries, we evaluated the precision at the first N labels associated to the top-level folders generated by SnakeT. Precision at top N is defined as:

$$P@N = \frac{M@N}{N}$$

where $M@N$ is the number of labels which have been *manually tagged* relevant among the N top-level labels computed by SnakeT. If a label l has been manually tagged as "ambiguous", we judge l relevant if the majority of its children labels are relevant. P@N is a classical information retrieval measure, and it represents the ratio of the number of relevant records retrieved by the first N results. In fact, we are more interested in achieving high precision for small values of N (e.g. in the first N results). Typically choices are $N = 3, N = 5, N = 7, N = 10$. We believe that P@N, specialized in the top-level folder labels, reflects the natural user behaviour of considering these top-level labels as the most important for hierarchy navigation. We use P@3, P@5, P@7 and P@10 since lazy users do not like browsing a wider folder hierarchy.

Benefits of using Dmoz. The index $\mathcal{R}_{\texttt{dmoz}}$ acts as a ranking engine for driving the selection of the best gapped sentences as folder labels. This engine produces a significant boost of P@N since it increases the number of relevant top-level labels. In our experiments we noticed this phenomenon on a very large set of queries (see Figure 4.21 for queries "Travel" and "Movie").

Benefits of using the anchor-text index. The index $\mathcal{A}_{\texttt{link}}$ was introduced to enrich Web pages that do not have a good textual description, or belong to the Web frontier reached by the search-engine spiders. In this set we also have the top authorities which are Web pages described well by many other pages [124]. In Figure 4.22 we report different values of P@N for the queries "Yahoo" and "Sherlock Holmes", using or not using the anchor-text index. An interesting phenomenon common to many queries may be observed: *The use of anchor texts to enrich the snippets increases P@N for the lower values of N.*

Benefits of using multiple search engines. Meta-search engines offer better coverage of the Web because of the low overlap currently experienced by individual search engines (like Ask Jeeves,

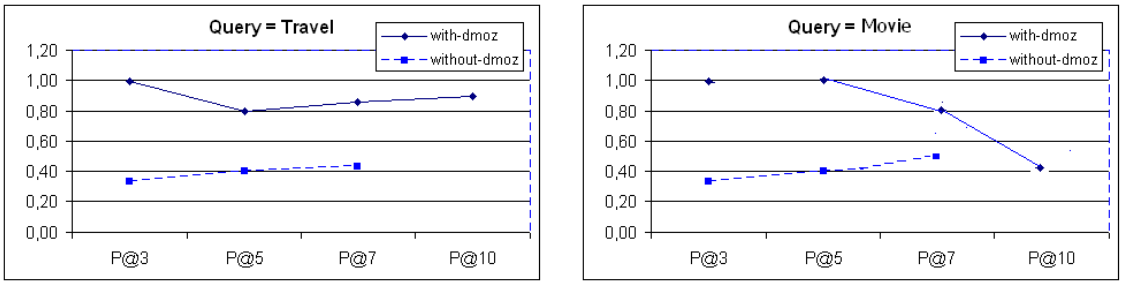


Figure 4.21: P@N using the Dmoz index on the queries "travel" and "movies".

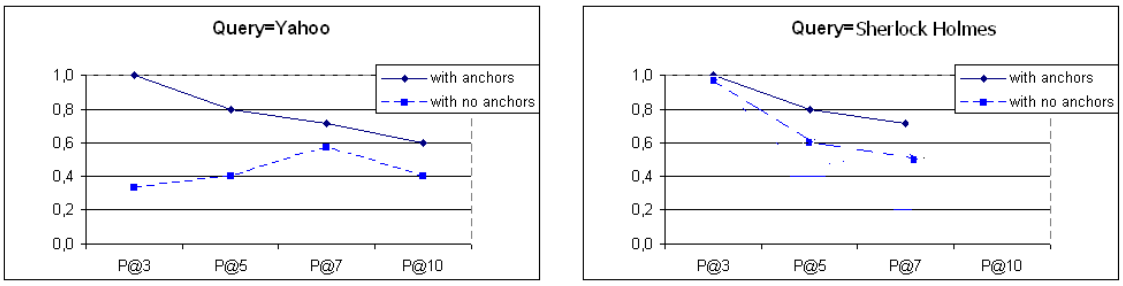


Figure 4.22: P@N using the anchor-text index over the queries "Yahoo" and "Sherlock Holmes".

Google, Yahoo, MSN, Teoma [15, 96]). This is usually seen as a limitation, rather than a resource, when designing a meta-search engine because of the difficulty in combining the list of results returned by multiple search engines. Figure 4.23 reports a different view of this issue: the use of query results coming from many search engines produces a more detailed and meaningful labeled folder hierarchy. In this case, SnakeT takes advantage of the *heterogeneous* results returned by its multiple underlying search engines to draw a more variegated set of themes. Notice that these themes might be assigned with low ranks in commodity search engines because of either their low popularity over the Web or their freshness of publication in the Web sphere [45]. Conversely, the user can then take advantage of SnakeT's personalization to filter out only the ones of interest to him/her from the 200 results.

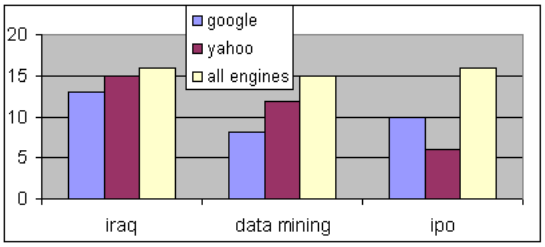


Figure 4.23: Number of top-level folders on three queries: "iraq", "data mining", "ipo".

Benefits of using gapped sentences as folder labels. Since some clustering engines use contiguous sentences as folder labels (see Carrot2 and [187]), we tried to evaluate the impact of SnakeT's gapped sentences concerning the meaningfulness of the selected labels. We studied the distribution of the gaps for the most relevant pairs of terms within the snippets of our dataset. Figure 4.24 reports the distribution of four relevant term pairs for the query "car rental" within different text gaps, in logarithmic scale. All other experiments [200] confirm these figures. Due to these observations, SnakeT adopts a maximum window of length four for generating the gapped sentences (stop words are not taken in account).

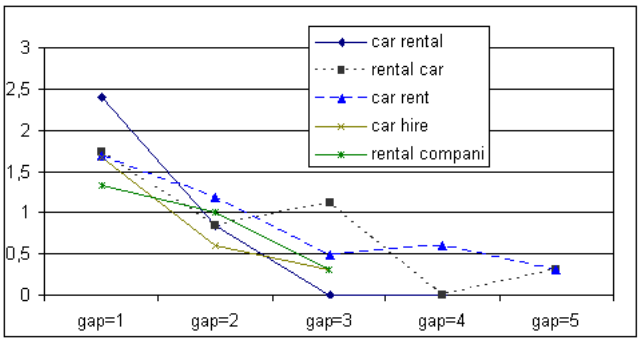


Figure 4.24: Log-distribution of relevant term pairs as a function of the gap between their occurrence.

We are left with the study on the overall performance of SnakeT. Here, we comment on the most salient aspects and refer to Table 4.29 for details.

Number of top-level labels and their precision. SnakeT was tuned to produce only meaningful top-level labels and to discard of those reporting a low rank. Therefore not all the queries produce ten top-level labels, and hence ten top-level folders. In Figure 4.25 we report the exact number of top-level labels (folders) generated for all the 77 queries in our dataset. Notice that all queries produce at least three, and many of them produce up to ten, top-level labels. As far as the precision of SnakeT's top-level labels is concerned, we observe that on all queries of our dataset, SnakeT achieved an average precision of P@3=91%, P@5=83%, P@7=78% and P@10=79%, see Figure 4.26.

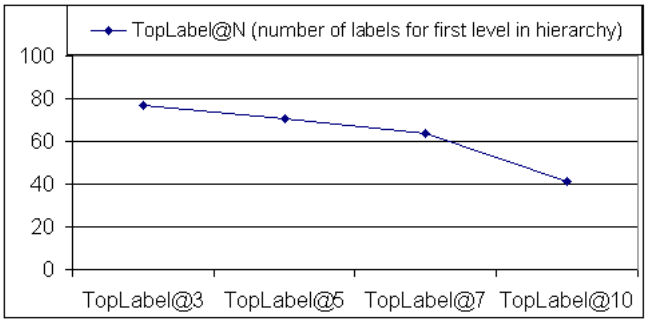


Figure 4.25: Number of queries in our dataset generating N top-level labels (TopLabels@N).

Precision over the personalized results. In Section 4.2 we described the ranking adopted by SnakeT to aggregate the results of the 18 search engines, and in Section 4.3 we described how SnakeT personalizes the search results. We studied how precision changes when SnakeT's personalization

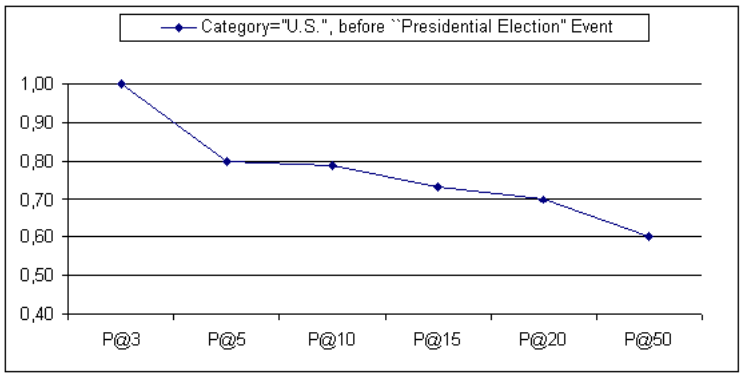


Figure 4.26: P@N on our dataset.

is applied, by using the following measure *over the snippets*: $P@N_{snippets} = \frac{MS@N}{N}$, where $MS@N$ is the number of snippets which have been *manually tagged* as relevant among the N top-ranked snippets returned by SnakeT's personalization. We experienced an increase in $P@N_{snippet}$ of about 22% averaged over N and over all 77 queries of our dataset. In Figure 4.27 we compute $P@N_{snippets}$ on personalized vs. unpersonalized results for the queries "divx" and "guppy".

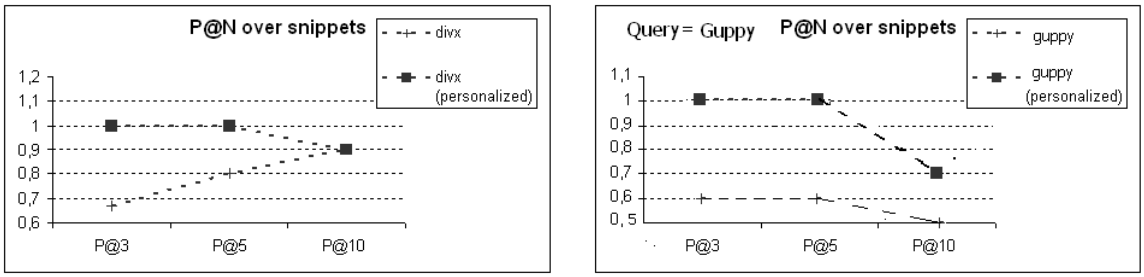


Figure 4.27: P@N for the queries "divx" and "guppy".

Number of Web snippets contained into the top-level folders. The *weight of a folder* is defined by the number of snippets it contains. A hierarchy is defined as *weight balanced* if nodes at the same level have comparable weights. In Figure 4.28 we report the distribution of the weights for the top-level folders generated for the query "data mining". Only the top-level folder "Software" has been expanded for the sake of exposition. Notice that SnakeT's hierarchy is balanced, and this phenomenon occurs on most of the queries of our dataset. We remark that a good weight-balance is crucial for our form of personalization since it enforces the folder hierarchy to equally detail all the themes behind the user query.

4.5 Conclusions

We have presented different algorithms in [73, 74, 75, 76, 95] which are exploited in SnakeT, a publicly-available meta-search engine for Web snippet clustering (http://snaket.com). SnakeT achieves efficiency and efficacy performance close to the industrial state-of-the-art, namely VIVISIMO [206] and MICROSOFT [216] but our proposal improves those solutions. The key idea is that a user issues a query using SnakeT, gets back a labeled folder hierarchy built automatically on the top of hundreds of search results. Then, (s)he selects a set of folders whose labels (themes) best fit his/her query needs. Given this selection, SnakeT personalizes the original ranked list by *filtering out* those

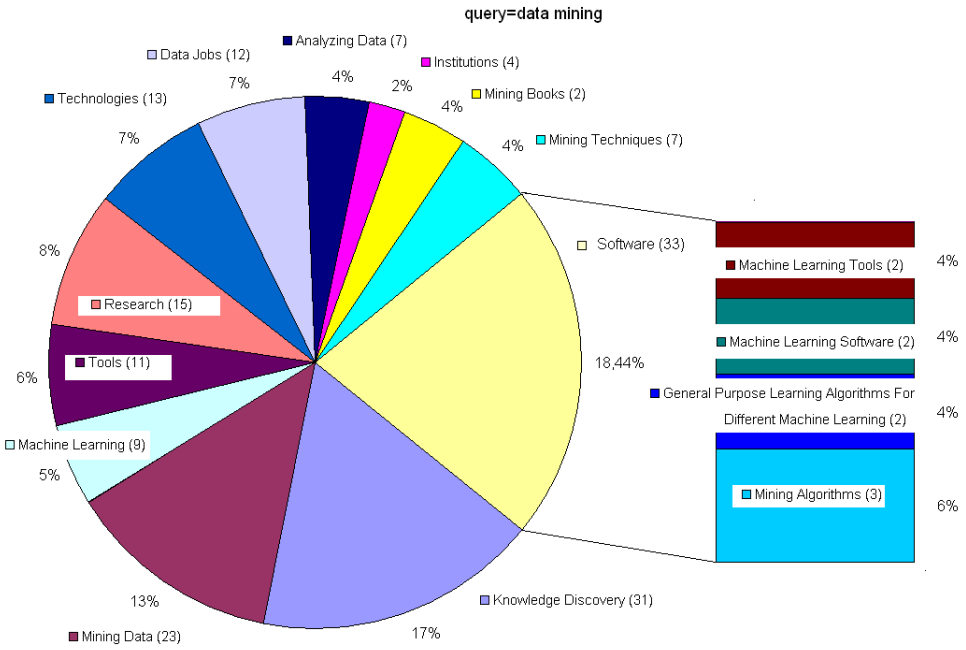


Figure 4.28: Folder weights for the query "data mining".

results which do not belong to the selected folders on-the-fly. In this way, the personalization is then carried out by the people themselves. We point out that SnakeT does not use predefined ontology and the topics and their organization in a hierarchy are inferred on-the-fly analyzing the list of answers given by Web search engines. This approach of course does not require an explicit `login` by the user, a pre-compilation of a constrained user profile, a tracking of the user's past search behaviour, or a modification of the underlying search engines. We believe that SnakeT is a valid solution in all situations where a flat list of search results in not enough. This limit is particularly evident when a query has many different meanings on the Web. In this case, link analysis methodologies tend to boost those Web pages related to the most popular meanings and to penalize those pages related to less popular meanings. Another situation where the limit is evident, is when users have difficulties in expressing their search needs. For all these cases, SnakeT is the WebIR tool that hierarchically groups search results by similar topics. Therefore, it is useful to orientate the query towards a more specific search need, and to help users understand the meaning of less popular query's. We conclude the Chapter with a final tought. We should point out that so far, despite many efforts, no large commercial engine has been successful in integrating clustering in the general UI in a manner that the users found compelling. The only notable exceptions are Vivisimo with its Clusty.com and AOL, which was granted a licence by Vivisimo. It is still an open question to understand if most users find clustering useful or if they prefer to simply re-issue a narrower query.

The research line proposed in this Chapter has been later extended in the papers [176, 43, 188], which evaluate different methodologies for re-ranking and not filtering the snippets returned by a remote search engine. Dumais in [176] suggests personalizing the top search results locally for privacy reasons by using BM25 [114], a well known probabilistic weighting scheme. User profiles have been implicitly derived either by exploiting a desktop index, such as that offered by Ask Jeeves or Google, by exploiting the past submitted queries, or by exploiting the similarities between Web pages visited in the past and the snippets returned by search engines. This work shows that personalization with the user's entire desktop index provides the best results, followed by representations based on subsets of the index (recently indexed content, and Web pages only). It also showed that using only the user's query history (query) or non user-specific representation

(no model) did not perform as well. The work confirmed that the best corpus representation is not document based but should exploit just the title and snippets, which are inherently query focused. This result was also confirmed by our research. Another significant contribution of [176] is the suggestion of using query expansion for search personalization. A good strategy for query expansion should exploit the terms which are used in the snippets matched near to those of the query itself. We note that we can consider our form of clustering labels as a powerful methodology for query expansion and therefore we consider the two results as similar. As discussed above, we believe that the best personalization is carried out on-the-fly by people themselves, and we do not believe that it is possible to maintain updated profiles for each single user by analyzing the query and click logs. Nevertheless, query and click logs can be useful in detecting similarities amongst Web pages, which cannot be inferred by just analyzing the text. Other types of non-textual similarities could be inferred by exploiting hyper-textual links between the pages. Unfortunately, query and click logs and Web graph crawling are subject to data privacy and therefore they are not easily available for academic research.

The recent work in [43] uses a form of explicit personalization where profiles are explicitly created by selecting preferred categories in DMOZ. Personalization is achieved by re-ranking snippets according to many distance metric with the preferred DMOZ categories. We note that this explicit profiling approach suffers from the problem of maintaining the profiles over time, and from the problem of imposing the user to the time-consuming and non flexible task of choosing among a predefined and static list of categories. On the contrary, our profiles are implicitly derived from the users' selection over the hierarchy of clusters built on-the-fly; and these implicit profiles do not need to be maintained.

An interesting future research theme is the study of metrics for evaluating both the hierarchy and the quality of the annotating labels. The first step in this direction has been given in [47] where the authors propose a goal-oriented evaluation measure, Hierarchy Quality, for hierarchical clustering algorithms. Their metric considers the content of the clusters, their hierarchical arrangement, and the effort required in finding relevant information by traversing the hierarchy starting from the top node. It compares the effort required to browse documents in a baseline ranked list with the minimum effort required to find the same amount of relevant information by browsing the hierarchy (which involves examining both documents and node descriptors). Nevertheless, the metric does not take into account the quality of the labels used to annotate the nodes in the hierarchy.

Another interesting work is [188]. The authors suggest re-ranking search results by optimizing two metrics: diversity – which indicates the variance of topics in a group of documents; and information richness – which measures the coverage of a single document of its topic. Both of the two metrics are calculated from a directed link graph named Affinity Graph (AG). AG models the structure of a group of documents based on the asymmetric content similarities between each pair of documents.

An interesting research theme to explore is the use of methodologies for Web snippet Clustering, like those used in SnakeT, upon a small random sample instead of the full set of search results. This idea appears intriguing after the publication of [5], where a technique for efficiently sampling Web search engine query results has been discussed. In fact, efficient sampling could mitigate the ranking bias of query results and consequently could allow a larger variety of the topics covered by the clustering hierarchies.

Another interesting line of research is to explore the use of hierarchical clustering for the personalization of those search results extracted from domains where no explicit hyper-links are available. In particular, for desktop searching a kind of search service which is nowadays provided by all the major search engines.

In the next Chapter we will face another situation where link analysis is not a good ranking solution. If the user is interested in fresh information, such as news articles, in our case study there is not enough time to have links pointing to it. For this reason, different ranking methodologies should be developed.

QUERIES	M@3	M@5	M@7	M@10
adsl	3	4	6	
aids	3	5		
airline flight tracking	3			
allergy	3	5	7	10
asthma	3	5	7	10
athens	2	2	3	
avril lavigne	3	5	6	8
bats	3	4	6	6
bible	3	5	6	8
britney spears	3	5	7	10
bush	3	4		
christina aguillera	3			
data mining	3	5	7	10
david beckham	2	4	6	8
divx	3	5		
dragon ball	3	3	5	8
dvd	3	5	7	10
dylan dog	3	3	3	5
eminem	3	5	7	
fbi	3	5	7	
final fantasy	3	5	7	10
final ipo	1	1	3	
firewall	3	5	7	
google	3	5		
grande fratello	3	5	7	8
guppy	3	3	4	
halloween	3	4	5	6
harry potter	3	5	7	10
hurricane	3	5	7	
ikea	2	2	2	
iraq	3	4	6	9
jaguar	3	4	6	
janet jackson	3	4	4	7
java	3	5	7	10
jennifer lopez	3	5	6	
kazaa	3	4	4	
las vegas	3	5	6	6
madonna	3	5	7	9
marjuiana	2	4	6	9
matrix	2	2	4	7
michelle vieth	3	4	4	
morpheus	3	5	7	9
mousetrap	3	3	5	6
movie	3	5	6	9
mp3	3			
music	3	5	7	10
napster	3	4		
nasa	2			
news	2	3	4	4
new york	3	5	5	6
nokia suonerie	2			
nostradamus	3	4	4	
pagerank	3			
perl	3	4	6	9
pink	2	3	3	
pisa	2	3	3	
pokemon	2	3	5	
retrovirus	2	3	3	5
sailor moon	3	5	5	5
samba	3	5	6	7
search	2	4	6	
seti	2	3	5	
shakira	3	4	6	
silvio berlusconi	2	4	4	
simpson	2	4	4	7
skateboard	3	5	6	9
sony	3	5	7	
spiderman	2	3	4	7
tattos	3	5	7	9
terrorism	3	5	7	10
time	3	5	6	8
travel	3	5	7	
vasco rossi	3	4	4	6
waterloo	3	5	6	7
winmx	2	2		
wtc	3	3		
wwf	3	4	6	7
	P@3	P@5	P@7	P@10
	91%	83%	78%	79%

Figure 4.29: M@N and P@N over the SnakeT's dataset.

Chapter 5

Ranking a Stream of News

Liars when they speak the truth are not believed, Aristotele

Abstract

In this Chapter we will face another situation where link analysis is not a good ranking solution. If the user is interested in obtaining fresh information, like news articles, there is not enough time to have links pointing to it. For this reason, different ranking methodologies should be developed. This need is particularly urgent. In fact, it has been noted [231] that, on an average day, about 94 million American adults use the Internet. Among them about 46% are trying to get news information. Surprisingly enough, this Internet activity is the third one, just after emailing and traditional Web searching. In response to this growing trend, the major Web search engines have established solutions for indexing news feeds, however, despite significant commercial interest, no academic research has as yet focused on the question of ranking a stream of news articles and a set of news sources as of yet.

This Chapter addresses this anomaly by proposing a framework for news ranking which models: (1) the generation of a stream of news articles, (2) news article clustering by topic, and (3) the evolution of a news story over time. The proposed ranking algorithm rates news information, seeking out the most authoritative sources and identifying the most current events in the particular category to which the news article belongs. Each of the aforementioned measures takes into account time parameters and do not require a predefined sliding observation window over the news stream. The complexity of our algorithm is linear in terms of the number of news articles under consideration at the time of a new posting, allowing for continuous on-line ranking. Our framework is validated by a collection of more than 300,000 news articles, generated over two months by more then 2000 news sources and belonging to thirteen different categories (World, US, Europe, Sports, Business, etc.). The collection is extracted from the index of comeToMyHead, an academic news search engine also developed for this purpose and available on-line. The index itself is freely provided to the research community for any non-commercial use. This line of research is also exploring the mutual benefit relationship between Clustering and Ranking, which is pervasive to our study. In this case, the better the rank provided by the news sources is, the better the clustering of news articles by similar topics is. Besides, the better the clustering built on-the-fly by our system is, the better will result the results of the ranking algorithms.

Ideas similar to the one discussed here, have been independently considered by Google, Ask Jeeves, Microsoft, and Yahoo! for commercial use. Google obtained a patent [227] after the publication of the results discussed in this Chapter [53, 94]. There is no public information about the methodologies used by Ask Jeeves, Microsoft and Yahoo!.

In the last year, there has been a surge of interest regarding news engines, i.e. software tools for gathering, indexing, searching, clustering and delivering personalized news information to Web users. According to a recent survey carried out by Nielsen NetRatings [219, 229], news browsing and searching is one of the most important Internet activities, with more than 28 million active users in the US recorded in October 2004 (see Figure 5.1). Yahoo! News had an audience roughly half that of Yahoo! Web Search, a third that of Google Web Search and slightly larger than AOL Web Search, which is surprising given that Yahoo News for example, had an audience of about 13 million users in 2002 [20]. To quote Peter Steyn of Nielsen/Netraing: "The Internet complements television for news coverage as it provides a different perspective and greater depth of information

- statistics, pictures, interactive maps, streaming video, and analyst comments". It is clear that recent world events such as the Tsunami, the SARS and Bird Flu epidemics, the war in Iraq and the constant threat of terrorism have contributed to the growth of on-line news search engines. Another study [231] pointed out that, on an average day, about 94 million American adults use the Internet. Among them about 46% are trying to get news information. Surprisingly enough, this Internet activity is the third one, just after emailing and traditional Web searching. It has been observed that the staggering amount of news articles available on-line reflects the users' need for a variety of information and opinions, and it could therefore, be said that news engines comprise a direct link to fresh and unfiltered information.

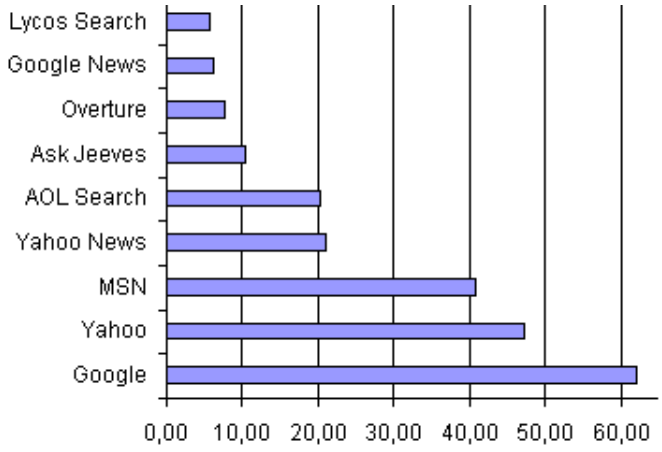


Figure 5.1: Comparing News and Web Search Engines (July 2004, Nielsen/Netratings).

The commercial scenario. Many commercial news engines can already be found on-line, such as, Ask News [213], Google News [196], Yahoo News [197], MSNBot [195], Findory [222] and NewsInEssence [228]. Aside from clustering similar news items, Google News retrieves up-to-the-minute information from more than 4,000 sources, categorizing it and automatically building a page that displays the most important articles in each category, while Yahoo news runs a similar service, accessing more than 5,000 sources. Microsoft recently announced the creation of NewsBot, a news engine that provides personalized news browsing according to the individual profile of each user, while Findory proposes a similar engine which relies on patent pending algorithms. NewsInEssence is another important engine, which clusters and summarizes similar news articles. [202] provides a complete listing of commercial news engines. Although there is no information available for public consumption regarding the way in which commercial search engines rank news articles, extensive tests performed on these systems by the author suggest that several criteria, such as the freshness and authoritativeness of articles, as well as replication/aggregation of news items are taken into account. In response to this, a framework is introduced in this Chapter which also exploits these criteria.

The scientific scenario. Despite the great variety of commercial solutions for news search engines, we found just a handful of papers relating to the topic [8, 38, 46, 57, 58, 59, 66, 83, 106, 146]. As mentioned above, NewsInEssence [57, 58] is a system for finding and summarizing clusters of related news articles from multiple sources on the Web and the system aims to automatically generate a news rundown by using a centroid based summarization technique. It works by considering salient terms forming clusters of related documents, and using these terms to construct a cluster summary. QCS [66] is a software tool and development framework for streamlined IR which matches a query to relevant documents, clusters the resulting subset of documents by topic, and produces a single summary for each topic, the main goal of the above systems being to create

summaries of clustered news articles. In [46], a topic mining framework for a news data stream is proposed. In [106], the authors study the difficulty of finding news articles on the Web that are relevant for the ongoing stream of broadcast TV news. In [59], a tool to automatically extract news from Web sites is proposed. In [83] NewsJunkie, a system which personalizes news articles for users by identifying the novelty of stories in the context of those already viewed, is analyzed. In [146], Mannilla et al. introduce the problem of finding frequent episodes in event sequences, subject to observation window constraints, where an episode is defined as a partially ordered collections of events, and can be represented by a directed acyclic graph. In [8] Atallah et al. propose an extension of [146] to rank a collection of episodes according to their significance. We note that, in reference to these episodes, both the entities which generated them, and the way in which they self-aggregate are not taken into account and, therefore, in this Chapter we demonstrate that these are crucial features in ranking a news story. [38] presents many different methodologies to correlate named entities extracted from news articles.

5.1 Our contribution

This Chapter discusses the problem of ranking news sources and a stream of news information generated over time. To the best of our knowledge, this is the first academic work written on this subject and, therefore, there is no possibility of comparing our results with other ranking methods. For this reason, we had to formalize the problem by laying out a number of potential requirements which would be desirable for our ranking scheme (Section 5.2) and by introducing a suitable model for describing interactions between articles and news sources (Section 5.3). The ranking algorithm is obtained by gradually introducing a number of constraints in order to match the requested requirements and is validated by two intuitive limit cases, which allow us to better define such approaches (Section 5.4). The final algorithm is described in Section 5.5. In detail:

- It allows the ranking of both the news articles and the news sources which produce information. The algorithm exploits a mutual relationship between news information and news sources: The value of a news article is proportional to how many times it has been replicated by other sources, and the value of a news source is proportional to the output of weighted news items.

- It takes into account the time when the articles have been produced and it dynamically models the importance of the articles just at the moment of time when they are provided as answer to a user query. This condition is modeled using a decay function which is translationally invariant, such as the exponential decay function. Note that other more complicated decay functions can be plugged into our schema, provided that they are translationally invariant.

- It exploits the similarities between articles for creating "virtual links" between pieces of news which share common topics. The intuition is that the more the articles discuss a topic areas, the more important the topic is.

- It acknowledges the importance of news sources that produce many "breaking news" items by giving them a-posteriori rank bonus. The intuition is that a source should be privileged if it is either breaking a story or it is following the story quickly. Moreover, it could potentially privilege news articles produced in the past which have many similar pieces of news produced after them. It should be pointed out that this last feature has not been exploited, since we have decided to privilege the fresher news articles. In a future work, we plan to plug this feature into a model that can exploit it in order to counter-balance the decay of the article importance.

The complexity of our method is parallel to the number of news articles still of interest at any particular time. This study is also described in the following paper [53].

Our ranking scheme is dependent on two parameters, ρ accounts for the decay rate of the freshness of news articles, and β, which controls the source's rank to be transferred to each news

Key Contributions
1. Ranking of News Articles
2. Ranking of News Sources
3. Exploiting News Clustering for Ranking
4. Time Aware Algorithms

Figure 5.2: Key contributions of this Chapter

article posted. Studying the sensitivity of the given rankings obtained by varying these parameters, we observed that our algorithm is robust, insofar as the correlation between rankings remains high, which changes the decay rule and the parameter β.

Subsequently, a comprehensive test was carried out, the results of which are presented in Section 5.7. The results obtained by ranking news articles and news sources for each category confirm our method's ability to recognize the most authoritative sources and to assign a high rank to important news items. The algorithms proposed in this Chapter aim to establish a general ranking schema based on unbiased factors rather than personal considerations, such as a user's interests or ideology. As with the Web search ranking scheme, it is possible to extend our approach by introducing a personalization parameter which accounts for the users's personal taste.

Our research pointed out that the relationship between Clustering and Ranking produces mutual benefits even for news ranking. In this case, the better the rank provided by the news sources is, the better the clustering of news articles by similar topics. Besides, the better the clustering built on-the-fly by our system is, the better the results of the ranking algorithms.

5.2 Some desiderata

Ranking news articles is an altogether different task from that of ranking Web pages, but we can at least expect a reduced amount of spam, since the stories themselves are derived from controlled sources. There are two different scenarios pertaining to the moment a news item is generated: (1) the news article is completely independent from previously published stories, or (2) it can be aggregated by a set of previously posted news articles. Furthermore, we stress that by definition, a news article is a *fresh piece of information* and, for this reason, when a news article is posted there is almost never a hyper-link pointing to it, therefore hyper-link based analysis techniques such as Pagerank [159] or HITS [125], can have limited benefits for news ranking. In Section 5.3 we outline a proposed model which exploits the *virtual linking* relationship between news items and news sources, based on both the posting process and the natural aggregation by topic between different stories. Now we will discuss certain requirements desired when ranking algorithms for news articles and sources, before presenting the algorithms which have been designed to match such requests.

Requirement R1: *Ranking for News posting and News sources.* The algorithms should assign a separate rank to news articles and news sources.

Requirement R2: *Important News articles are Clustered.* It is probable that an important news article n will be (partially) replicated by many sources, considering for example, a news article n originated from a press agency. A measure of its importance can also be gauged by the number of different on-line newspapers which either replicate or extract sections of text from n. The phenomenon of using stories released by other sources is common in the context of Web journals and, as far as the news engine is concerned, this means that the (weighted) size of the cluster formed around n can be deemed a measure of its importance.

Requirement R3: *Mutual Reinforcement between News Articles and News Sources.* We can assign particular importance to different sources according to the significance of the news articles they produce, so that for example, a news item from "The Washington Post" can

be deemed to be more authoritative than a similar article derived from the "ACME press", since the former has a reputation for reporting good news.

Requirement R4: *Time awareness.* The importance of a news item changes over time for the reason that we are dealing with a stream of information in which a fresh news story can be considered more important than an old one.

Requirement R5: *On-line processing.* It is necessary that the time and space complexity of the ranking algorithm permits on-line processing, that is to say, that at certain times their complexity can depend on the mean amount of news articles received rather than the time elapsed since the observation commenced.

In Section 5.5 we define an algorithm for ranking news articles and news sources which matches the above requirements. The algorithm is progressively designed, ruling out more simplistic algorithms which do not satisfy some of the above requirements.

5.3 A model for news articles

News posting can be considered a continuous stream process and to approach it we can exploit a certain window of observation. One way to analyze the stream is to maintain a window of a fixed size, in this way, the maximum amount of observed data remains constant, which can also result in being unable to determine the temporal relationship between news articles posted at a time not covered by the current window. A second method is to use a window of observation with unbounded time constraints and in adopting this method, the size of the data considered increases with time. This brings to light a typical data streaming problem in which the flow of information is so overwhelming that it is not feasible even to store the data or to perform single (or more) scan operations over the data itself (see [154] and references therein). This can be said to be particularly true for information flows, since different sources can independently post many streams of news articles. In Section 5.4.2 we propose a solution for this problem which handles the data stream of news information with no predefined time window for observation. This solution takes into account a particular decay function associated with any given news item.

In the following, we introduce the model which characterizes news articles and news sources. Given a news stream, a set of news sources, and fixing a time window ω, the news creation process can be represented by means of an undirected graph $G_\omega = (V, E)$ where $V = S \cup N$, S are the nodes representing the news sources, while N are the nodes representing the news stream seen in the time window ω. Analogously, the set of edges E is partitioned into two disjoint sets E_1 and E_2. E_1 is the set of undirected edges between S and N. It represents the news creation process, E_2 is the set of undirected edges with both endpoints in N and it represents the results of the clustering process which allows similar pieces of news to be connected. The edges in E_2 can be annotated with weights which represent the similarity between two pieces of news. The nodes in S "cover" those in N, i.e., $\forall n \in N, \exists\ s \in S$ such that $(s, n) \in E_1$.

We define a similarity measure among the news articles, in order to satisfy the Requirement (R2). This depends on the clustering algorithm chosen and accounts for the similarity among the news stories. Given two nodes n_i and n_j we define the *continuous similarity* measure as a real value $\sigma_{ij} \in [0, 1]$, signifying that σ_{ij} is close to 1 if n_i is similar to n_j. A simplified version provides a *discrete similarity* measure, which is 1 if the two news postings are exactly the same (in other words, they are mirrored) and 0 if they are different.

Let A be the (weighted) adjacency matrix associated with G_ω. We can attribute an identifier to the nodes in G_ω so that any source precedes the pieces of news. The matrix can be defined as

$$A = \begin{bmatrix} O & B \\ B^T & \Sigma \end{bmatrix},$$

where B refers to edges from sources to news articles, and $b_{ij} = 1$ iff the source s_i emitted article n_j and Σ is the similarity matrix between similar news articles. Assuming that one can learn

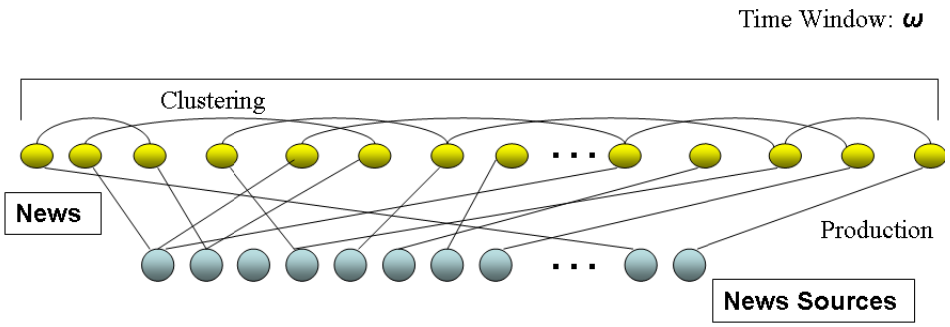


Figure 5.3: News Ranking Graph.

the similarity of sources, the matrix A can be modified in the upper-left corner incorporating a sub-matrix taking into account source-source information.

The amount of articles emitted in a short period of time from all the sources in a given category is an important parameter for the news engine. This quantity, denoted by $\mathsf{newsflow}(\mathsf{t}, \mathsf{c})$ for time t and category c, is subject to drastic variation over time due to great resonance events (for instance, during the first days of November 2004 we had a peak in newsflow for category "U.S." due to the Presidential Election).

We must remark that this model describes a framework where different data stream clustering algorithms (see [2, 91] and the references therein) can be plugged in, to create and weight the set of edges E_2. Starting from the above model, in Section 5.4 we propose some ranking algorithms which progressively satisfy the requirements described in Section 5.2, and fit the general model for representing news articles and news sources described here.

Before starting to discuss our contribution in detail, a consideration about data stream model is needed. In the data stream model [154], data arrives sequentially and is processed by an algorithm whose workspace is insufficient to store all the data, so the algorithm must process and then discard each data item. A data stream algorithm maintains a summary whose storage requirement is considerably smaller than that of the stream. Using this compact summary, the algorithm is able to answer queries about the data stream. We should note that our algorithms do not need to access the whole data stream in order to compute the rank of a news article. Nevertheless, it could be noted that storing all the news articles produced world wide is a feasible task with a medium sized network of workstations. In fact, MOREOVER.COM, one of the largest news article aggregators available on-line is producing about 250K articles per day, but it has a limited coverage of news articles produced in geographical areas different from U.S. If we suppose that about 2M of articles are produced daily world wide, we have about 720M articles per year – an amount of information that can be stored with no problem in a distributed network of workstations.

5.4 Algorithms for news articles and news sources

In order to evaluate the consistence of the algorithms presented in this Section, we have considered some limit cases for which the algorithms should show a reasonable behaviour. These limit cases allow us to refine the algorithms and match the requirements described in Section 5.2. They are:

`LC1:` A unique source s_1 emits a stream of independent news articles with an average emission rate $1/\Delta$. We expect the source to have a stationary mean value rank μ independent of time and the size of the observation window ω. μ should be an increasing function in $1/\Delta$.

`LC2:` Two news sources s_1, s_2, where s_1 produces a stream of independent news articles with average rate $1/\Delta$, and s_2 re-posts the same news stream generated by s_1 with a given average

delay. Essentially, the source s_2 is a mirror of s_1. Hence, the two sources should have a similar rank.

Note that the limit cases are intended as basic situations where the algorithms proposed in this Chapter are expected to have an intuitive behaviour. At this level, the limit case LC2 does not give a rank bonus to the first source which breaks the news article. Nevertheless, this desired property is introduced later on in this Chapter.

5.4.1 Time-unaware ranking algorithms

Any algorithm described in this Section satisfies only a subset of the requirements described in Section 5.2. Indeed, they are naive approaches that one has to rule out before proposing more sophisticated algorithms. In particular, these methods do not deal with the news flow as a data stream, but assume that they are available as a static data set. In the next Section 5.4.2 we introduce algorithms which overcome the limit of those given here.

`Algorithm NTA1`

The naive approach is that a news source has a rank proportional to the number of pieces of news it generates and, conversely, a news article should rank high if there are many other news stories close to it. Formally, denoting the vector of sources and news ranks by $\mathbf{r} = [\mathbf{r}_S, \mathbf{r}_N]^T$ it can be computed as

$$\mathbf{r} = A\mathbf{u},$$

where $\mathbf{u} = [\mathbf{u}_S, \mathbf{u}_N]^T$ is the vector with all entries equal to one. Given the structure of A, this means that

$$\begin{cases} \mathbf{r}_S = B\mathbf{u}_N \\ \mathbf{r}_N = B^T\mathbf{u}_S + \Sigma\,\mathbf{u}_N = \mathbf{u}_S + \Sigma\,\mathbf{u}_N, \end{cases}$$

that is, each source receives a rank equal to the number of news articles emitted by that source, while each single piece of news has a rank proportional to the number of similar news articles.

This algorithm shows bad behavior in the limit case `LC1`. Indeed, the rank r_{s_1} of a unique news source s_1, will increase limitlessly with the number of observed news articles. Besides, algorithm `NTA1` satisfies the requirements (R1) and (R2) but not (R3), (R4) and (R5).

`Algorithm NTA2`

The second algorithm exploits the mutual reinforcement requirement between news articles and news sources similarly to the way the HITS algorithm [125] relates Web hubs and authorities. The key idea is that the news sources rank is proportional to the rank of the articles posted by them, and the rank of the articles is proportional to a linear combination among the rank of the sources which emitted them and the rank of similar articles. This mutual relationship can be expressed mathematically as follow. Let us consider the fixed point equation

$$\mathbf{r} = A\mathbf{r}. \tag{5.1}$$

From the block structure of A we get

$$\begin{cases} \mathbf{r}_S = B\mathbf{r}_N \\ \mathbf{r}_N = B^T\mathbf{r}_S + \Sigma\,\mathbf{r}_N. \end{cases}$$

From Equation (5.1), it turns out that in order to have a non-zero solution, $\mathbf{r}$ should be a right eigenvector corresponding to an eigenvalue equal to 1, but this is not true in general. In particular, this is not valid for case `LC1` and $\mathbf{r} = \mathbf{0}$ is the only solution of (5.1). This algorithm is also not stream oriented like the NTA1. A major difference with NTA1 is that NTA2 satisfies the requirements (R1), (R2) and (R3).

It is easy to show that the class of non time-aware algorithms does not satisfy at least one of the limit cases defined in Section 5.4. Moreover, the fixed time-window scheme can not explore precise temporal information within a window, and looses the opportunity to discover the temporal relationship between news articles released at a time not covered by the current window.

5.4.2 Time-aware ranking algorithms

In order to manage a news data stream, we have to design time-aware mechanisms which do not use fixed time observation windows over the flow of information, the main point being that, the importance of a given news item is strictly related to the time of its emission. Hence, we can model this phenomenon by introducing a parameter α which accounts for the decay of "freshness" of the news story. This α depends on the category to which the news article belongs. For instance, it is generally wise to assume that sports news will decay more rapidly than health news.

We denote by $R(n,t)$ the rank of news article n at time t, and analogously, $R(s,t)$ is the rank of source s at time t. Moreover, by $S(n_i) = s_k$ we mean that n_i has been posted by source s_k.

Decay rule: We adopt the following exponential decay rule for the rank of n_i which has been released at time t_i:

$$R(n_i,t) = e^{-\alpha(t-t_i)}R(n_i,t_i), \quad t > t_i. \tag{5.2}$$

The value α is obtained from the half-life decay time ρ, that is the time required by the rank to halve its value, with the relation $e^{-\alpha\rho} = \frac{1}{2}$. In the following, we will specify the parameter ρ, expressed in hours, instead of α. In addition, we will discuss how to obtain the formula of an effective algorithm for ranking news articles and sources, and demonstrate that naive time-aware algorithms exhibit a bad behaviour in many cases: either they do not work in the above limit cases or they do not respect proprieties R1-R5. Furthermore, we will refine these algorithms in order to maintain complete control over the ranking process.

`Algorithms TA1`

The first class of time-aware algorithms assign a news source the sum of the ranks of news information which have been generated by that source in the past in accordance with the above decay rule. The algorithms belonging to this class only differ from each other in the way they rank each news article at the time of its first posting.

Setting the rank of a news article at the time of its initial posting to one, we have

$$\begin{cases} R(s_k,t) = \sum_{S(n_i)=s_k} R(n_i,t) \\ R(n_i,t_i) = 1. \end{cases} \tag{5.3}$$

Assuming that the source s_k did not post any news information in the interval $[t, t+\tau]$, we see that the variation of ranks after an elapsed time of τ is described by the two following relationships

$$\begin{aligned} R(n_i,t+\tau) &= e^{-\alpha\tau}R(n_i,t), \quad t \geq t_i \\ R(s_k,t+\tau) &= e^{-\alpha\tau}R(s_k,t), \end{aligned} \tag{5.4}$$

We underline that this algorithm smooths the effect of previously issued news articles, and it meets the limit case `LC1`. Indeed, assuming case `LC1` is satisfied, for the stationary mean value μ of the rank of s_1, we have

$$\mu = \theta\mu + 1, \tag{5.5}$$

where $\theta = e^{-\alpha\Delta}$. From (5.5) we derive the mean value of the rank $\mu = 1/(1-\theta)$ in the case of a single source emitting independent news articles with an average rate $1/\Delta$. We point out that this algorithm satisfies Requirements (R1), (R4) and (R5) but it does not satisfy (R3) since the rank attributed to a news article does not depend on the rank of the source which posted it. For accounting Requirement (R3), we can still consider equation (5.3), changing the rank attributed

to a piece of news when it is released. For example, we can define the rank of a news story as a portion of the rank of its source in the instant before it is emitted. The algorithm becomes

$$\begin{cases} R(s_k,t) = \sum_{S(n_i)=s_k} R(n_i,t), \\ R(n_i,t_i) = c \ \lim_{\tau\to 0^+} R(S(n_i), t_i-\tau), \end{cases}$$

where $0 < c < 1$. As a starting point we assume that $R(s_k, t_0) = 1$, however, with any non-zero initial conditions the limit case `LC1` has a bad behavior again. There is no stationary mean value of the rank even for a single source s_1 emitting a stream of independent news articles. In fact, assuming μ to be the stationary mean value of $R(s,t)$, we have

$$\mu = \theta\mu + c\,\theta\mu,$$

which cannot be solved for $\mu \neq 0$.

In order to solve the problem, we change the starting point in (5.3) again to smooth the influence of the news source on the rank of the news articles. Let us set

$$R(n_i,t_i) = \left[\lim_{\tau\to 0^+} R\left(S(n_i), t_i-\tau\right)\right]^{\beta}, \quad 0<\beta<1.$$

The parameter β is similar to the magic ε accounting for the random jump in Google's Pagerank [159]. In fact, as for the random jump probability, the presence of β here is motivated by both a mathematical and a practical reason. From a mathematical view point, the fixed point equation involving the sources, has a non zero solution. From a practical point of view, by changing β we can tune how the amount that the arrival of a single fresh piece of news can increase the rank of a news source. In fact, let t_{i-1} be the time of emission of the previous news article from source s_k, and let t_i be the time of release of n_i by s_k. If in the interval (t_{i-1}, t_i) no article has been issued by s_k, we have

$$R(s_k,t_i) = e^{-\alpha(t_i-t_{i-1})}R(s_k,t_{i-1}) + R(s_k,t_{i-1})^{\beta}.$$

For the limit case `LC1` the fixed point equation now becomes

$$\mu = \theta\mu + (\theta\mu)^{\beta}$$

which has the solution $\mu = \left(\frac{\theta^{\beta}}{1-\theta}\right)^{\frac{1}{1-\beta}}$. In this model we can also very easily deal with the limit case `LC2`.

`Algorithm TA2`

We have seen that algorithms in the class `TA1` satisfy the limit cases as well as Requirements (R1), (R3), (R4) and (R5). However, they fail to satisfy the Requirement (R2), since the rank of a news article is not related to the rank of those similar to it. This can be said to be a desired requirement, due to the fact that if an article is known to be of interest, there will be a large number of news sources posting similar pieces of information. Therefore, a good news ranking algorithm working over a stream of information should also exploit some data stream clustering techniques.

Formally, this can be described as follows: Let us set the rank of a piece of news at emission time t_i to be

$$\begin{aligned} R(n_i,t_i) &= \left[\lim_{\tau\to 0^+} R\left(S(n_i), t_i-\tau\right)\right]^{\beta} + \\ &+ \sum_{t_j<t_i} e^{-\alpha(t_i-t_j)}\sigma_{ij}R(n_j,t_j)^{\beta}, \end{aligned} \tag{5.6}$$

where $0 < \beta < 1$. In this case the rank of an article is dependent on the rank of the source and on the rank of similar news articles previously issued, whose importance has already decayed by a negative exponential factor. The rank of the sources is still

$$R(s_k, t) = \sum_{S(n_i)=s_k} R(n_i, t).$$

While studying the behaviour of this algorithm on the limit cases `LC2` we experimentally learned that a news source mirroring another achieves a finite rank significantly greater than that of the mirrored one.

5.5 The final time-aware algorithm: TA3

In order to fix the behaviour of the formula while assigning ranks to news sources and to deal with the limit case `LC2`, we modify "a posteriori" a mirrored source's rank. In particular, a source which has produced news stories in the past, which will be highly mirrored in the future, receives a "bonus" as acknowledgement of its importance. This modification captures the intuitive idea of "breaking news". News sources which produce a lot of "breaking news" are considered very important. The final equation for news sources and news stream becomes

$$\begin{aligned} R(s_k, t) &= \sum_{S(n_i)=s_k} e^{-\alpha(t-t_i)} R(n_i, t) + \\ &+ \sum_{S(n_i)=s_k} e^{-\alpha(t-t_i)} \sum_{\substack{t_j > t_i \\ S(n_i) \neq s_k}} \sigma_{ij} R(n_j, t_j)^\beta, \\ R(n_i, t_i) &= \left[\lim_{\tau \to 0^+} R\left(S(n_i), t_i - \tau\right) \right]^\beta + \\ &+ \sum_{t_j < t_i} e^{-\alpha(t_i - t_j)} \sigma_{ij} R(n_j, t_j)^\beta. \end{aligned} \tag{5.7}$$

The rank of a news source s_k is then provided by the ranks of the piece of news generated in the past, plus a factor of the rank of news articles similar to those issued by s_k and posted later on by other sources. The equation for ranking the articles remains the same as (5.6). Note that if an article n aggregates with a set of pieces of news posted in the future, we do not assign an extra bonus to n (acknowledging a posteriori the importance of n). Principally, we are concerned with giving privilege to the freshness of the news posted instead of its clustering importance. However, the news source which was first posted a highly aggregating article is awarded an extra rank, due to the fact that it made a "scoop" or posted some "breaking news" (in journalistic jargon).

In a future study, we plan to privilege those news articles produced in the past which have many similar pieces of information produced after them. The key intuition is that we may want to mitigate the decay of importance of those articles that are talking about a current hot topic, even when they have been produced in the past. We believe that this is one of the most convenient ways of controlling the decay of importance. Another option is to monitor the number of times that an article has been clicked upon. This last alternative requests a certain initial latency for accumulating enough click information before being useful.

This algorithm is coherent with all the desirable requirements described in Section 5.2. Nevertheless, it is more complicated than those analyzed in the previous Sections, adding that it is not easy to compose a formula for the stationary mean value of the source, however, as shown in Figure 5.4, the limit cases `LC1` and `LC2` are satisfied.

5.5.1 Using a clustering technique

The naive clustering used for the first set of tests set $\sigma_{ij} = 1$ if n_i and n_j are the same (i.e. they are mirrored). In our news collection, these cases are very limited. Hence, by using these values

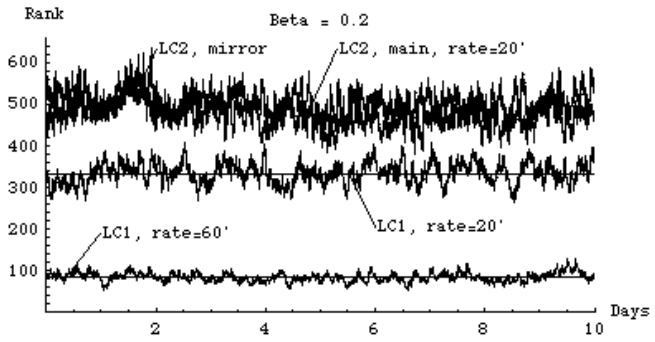


Figure 5.4: Simulated behavior of the limit cases LC1 and LC2 with $\beta = 0.2$. From below, the two straight lines represent the theoretical values of LC1 with a decay rate ρ of 60 min and of 20 min. There is a good relationship between the theoretical and the actual values of source ranks. In the upper part, the ranks of two sources emitting the same news stream are plotted.

of σ_{ij} the news sources' ranks are highly correlated with the simple counting of the posted news articles. A more significant indication of this may be obtained by taking a continuous measure of the lexical similarity between the abstracts of the news posting. These abstracts are directly extracted by the index of the news engine itself. In our current implementation, the news abstract is represented by using the canonical "bag of words" representation and the abstracts are filtered out by a list of stop words. The lexical similarity is then expressed as a function of the common words shared by news abstracts. We point out that dealing with a continuous similarity measure produces a full matrix Σ, whose dimensions increases over time, although fortunately the decay rule allows us to consider just the most recently produced part of the matrix, maintaining its size as proportional to the $newsflow(t, c)$, and therefore satisfying the Requirement (R5).

Our research pointed out that the relationship between Clustering and Ranking produces mutual benefits even for news ranking. The above theoretical results show that the better the rank provided by the news sources is, the better the clustering of news articles by similar topics. Besides they show that the better the clustering built on-the-fly by our system is, the better the results of the ranking algorithms.

5.5.2 Ranking the events

An interesting feature of our algorithm is the possibility to analyze the behaviour of the mean value of the ranks of all the sources, over time and for each given category. This measure gives us an idea of the activity of that particular category and is related to particularly relevant events. More specifically, we define the mean value of the rank of all the sources at a given time t, that is

$$\mu(t) = \frac{\sum_{s_k \in S} R(s_k, t)}{|S|}. \tag{5.8}$$

In Section 5.7 we discuss this mean value for a particular category.

5.6 The anatomy of comeToMyHead, the news search engine

For evaluating the quality of the above theoretical results, we used the data set collected by comeToMyHead, a News Search engine gathering news articles from more than 2000 continuously updated sources. The data set consists of about 300.000 pieces of news collected over a period of two months (from 8/07/04 to 10/11/04) and classified in 13 different categories (see Figure 5.7). Each article n is uniquely identified by a triple $< u, c, s >$, where u is the URL where the news article is located, c is its category, and s is the news source which produced n. The author of this

Thesis freely provides the data set for any non-commercial use or for any academic research [218]. The current data set contains more than 1 million news articles.

comeToMyHead is a on-going project for developing a complete news search engine which is not only used to collect a dataset of piece of information, or to rank the news article results. The search engine is useful for retrieving, ranking, indexing, classifying, clustering and delivering personalized news information extracted both from the Web and from news feeds. We remark that the project is still in its infancy and needs further improvements. Nevertheless, an almost complete framework for news searching has been developed during our Ph.d. studies and is described briefly in an positional paper [94]. In future, we plan to focus more and more of our research on this activity to produce a more sophisticate and complete version of the system. The system is made of the modules depicted in Figure 5.6. It has a Web interface at http://newsengine.di.unipi.it/. In the following, we will briefly describe these modules.

The retrievers. This module gathers news from a set of selected remote news sources. It supports different syndication formats, such as Atom [214] and RSS [191]. Besides, it is possible to feed the system with selected information extracted from remote Web pages. Currently, we have selected a list of news sources consisting of about 2000 different Web journals. For efficiency reasons, the space of news sources is partitioned and this module is composed of several processes which run in parallel. The data is collected 24h/24h, and the stream of information is stored into a local database $\mathcal{N}_{\mathtt{db}}$.

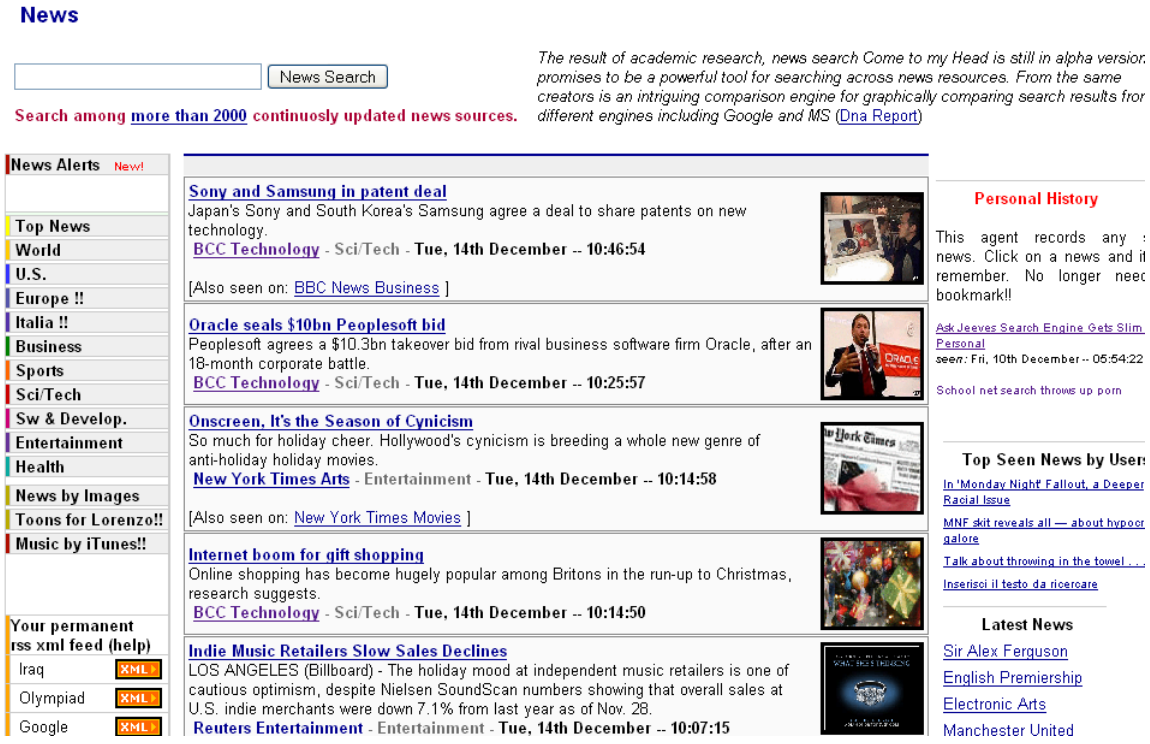


Figure 5.5: comeToMyHead, the Web interface of our News engine

The images processor. The image processor module analyzes information stored in the $\mathcal{N}_{\mathtt{db}}$. It tries to enrich any news with an associated image. In the easy case (e.g. Toons category), the news source has already associated an image to a given piece of news in the RSS/Atom feed. This association is expressed as an HTML fragment, so that we can easily download the image locally and create a suitable thumbnail of it. In other situations, we have an item of news n, extracted by a Web page p or by an RSS/Atom feed, with no associated image which refers to a Web page p. We download the images contained in p locally and use many heuristics to identify the most

suitable image to be associated with n. We take into account contextual information (e.g. where the image is placed in the HTML source) as well as content information (e.g. image's size, colors and other features).

The feature selection index. We are adopting an experimental feature selection index $\mathcal{F}_{\mathtt{db}}$ to design an efficient method for extracting meaningful terms from the news. As suggested in [76], we index the whole DMOZ and implement a `TFIDF` measure that is *DMOZ-category* centered. $\mathcal{F}_{\mathtt{db}}$ is built at preprocessing time and then used for selecting and ranking the key features used for News classification and clustering on-the-fly.

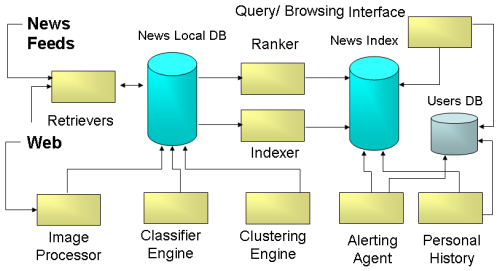


Figure 5.6: Architecture of the News Engine.

The classifier engine. All the news in $\mathcal{N}_{\mathtt{db}}$ needs to be classified. The categories defined in the system are given in figure 5.7. We are experimenting a naive Bayesan classifier. Note that a (relatively large) part of the RSS/Atom feeds are already manually classified from the originating news source. Consequently, the key idea for classifying is to use the classifier in a *mixed mode*: as soon as an article which has already been classified by a news source is seen, the classifier is switched onto training mode; the remaining unclassified news articles are categorized with the classifier in the categorizing mode.

The clustering engine. We are experimenting a variant of the k-means algorithm with distance threshold for creating a *flat clustering* structure, where the number of clusters is not known a-priori. This is very useful for discovering similar news items or news mirrored by many sources.

The ranker engine. comeToMyHead uses a ranking algorithm, simpler than the one discussed in this Chapter, and based on (1) news freshness and on (2) news clustering aggregation. We plan to adopt the full ranking algorithm described on-line in the near future. In the next Section we will describe the ranking experiments carried out off-line on a collection of news articles extracted from COMETOMYHEAD's dataset.

The indexer. Classified and Clustered news coming from $\mathcal{N}_{\mathtt{db}}$ are indexed by this module, which produces an inverted list index. For each piece of news we store the source, the URL, the title, the associated category, the description, a rank value, and the publication date.

Query and Browsing interface. The index is accessed by the search and browsing interface. A typical user can access the news by category, search within any category and search all the indexed news. All the results are available with a public Web interface. Moreover, we provide an RSS search mode. Using this feature, a user can submit a list of keywords and they receive a permanent XML feed with the most updated news about that topic. In this way, comeToMyHead acts as a public aggregator for news feeds.

Personalized alerting agent. We integrated a mail alerting system, which produces a personalized daily summary of the news chosen by users. In fact, each user can login into the system

and record a private set of queries. As soon as a fresh and related news article appears on the system, it is aggregated into a mail sent to the user daily. Information about users is stored in a local database $\mathcal{U}_{\mathsf{db}}$.

Personal history tracker. This module allows transparent tracking of the news selected by the users through different browsing or searching sessions. Currently, we use this feature for alleviating users from the need to bookmark interesting news. Data is stored in the $\mathcal{U}_{\mathsf{db}}$ and kept private. Therefore, preserving the users' privacy it is accessible. In this case, the system does not request an explicit login, but works using a cookie mechanism. We also plan to use this kind of data in an anonymous form, for providing personalized search results.

Helper modules. comeToMyHead provides other services available through the Web interface. They are: (i) "*Top Ten*", the most accessed news for each category; (ii) "*Top News Sources*", the most active news sources for each category; (iii) "*Latest News*", the list of most frequent named entities dynamically extracted at query time from each category.

5.7 Experimental results

We carried out our experiments using a PC with a Pentium IV 3GHz, 2.0GB of memory and 512Kb of L2 cache. The code is written in Java and the ranking of about 20.000 news items requires few minutes, including the clustering algorithm.

5.7.1 Evaluation of the ranking algorithms

We have used this particular search engine to gather a collection of news articles from several different sources over a period of two months. Our experimental settings are based on data collected by comeToMyHead from more than 2000 sources over two months and classified in 13 different categories, consisting of about 300,000 news items. The measurements start from 8/17/04 and discard the first 10 days of observation, to allow our ranking algorithm to achieve a stationary behaviour.

Category	# Postings	Category	# Postings
Business	34547	Entertainment	43957
Europe	19000	Health	11190
Italia	7865	Music Feeds	690
Sci/Tech	25562	Software & Dev.	2356
Sports	39033	Toons	1405
Top News	54904	U.S.	10089
World	53422		

Figure 5.7: How the news postings gathered in two months by comeToMyHead are distributed among the 13 categories.

Sensitivity to the parameters

The first group of experiments have addressed sensitivity to changes in the parameters, and we must add that our ranking scheme is dependent on two parameters, ρ, accounting for the decay rate of news article freshness, and β, which gives us the amount of a source's rank that we wish to transfer to each news posting.

As a measure of concordance between the ranks produced with different parameter values, we adopted the well known Spearman [170] and Kendall-Tau correlations [123]. We report the ranks computed for the category "World" with algorithm `TA3`, for values of $\beta_i = i/10$, where $i = 1, 2, \ldots, 9$ and for $\rho = 12$ hours, 24 hours and 48 hours. In Figure 5.8, for a fixed ρ the abscissa β_i represents the correlation between the ranks obtained with values β_i and β_{i-1}. From this plot we can see

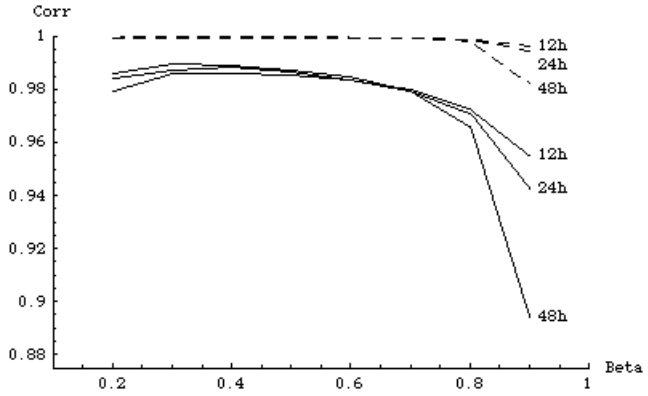


Figure 5.8: For the category "World", the figure represents the correlations between ranks of news sources obtained with two successive values of β differing by 0.1. The solid lines are the Kendall-Tau measure, the dashed lines are the Spearman correlation coefficients.

that the Kendall-Tau correlation is a more sensitive measure than the Spearman correlation, and that the algorithm is not very sensitive to the changing parameters involved. Owing to the fact that we do not have any way of establishing the optimal choice of these parameters, this could be said to be an attractive requirement.

It is also crucial that we compare the source rank obtained by our algorithm with the one obtained by a simpler schema. For this reason, we compare the mean source ranks over the observed period generated with algorithm `TA3` with the naive rank obtained using method `NTA1`. We recall that `NTA1` assigns a rank equal to the number of news items posted to a source. A matrix of Kendall-Tau correlation values is obtained by comparing the two ranks with β varying from 0.1 to 0.9 and for ρ varying from 5 hours to 54 hours. In Figure 5.9 this matrix is plotted as a 3-D graph. The correlation values show how the algorithm `TA3` differentiates from the naive NTA1.

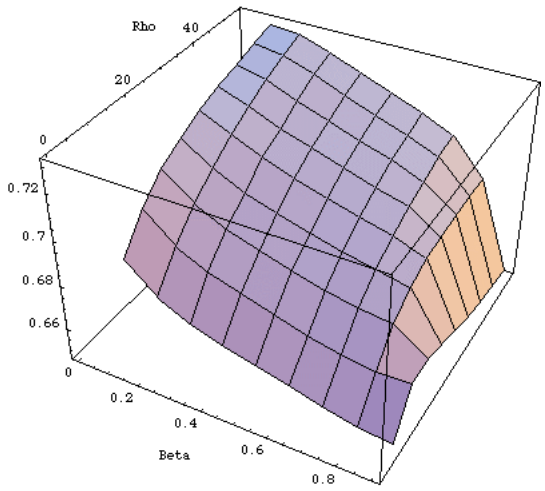


Figure 5.9: A 3-D plot of the Kendall correlation between the mean news source rank vector produced by the final time-aware algorithm `TA3` in a period of observation and the rank produced by the naive algorithm `NTA1` that simply counts the news articles emitted, for various values of ρ and β. The correlation values show how the algorithms differ in the ranks given to the news sources.

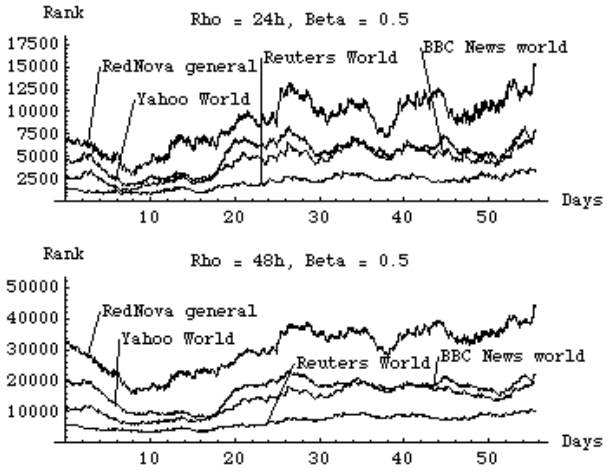


Figure 5.10: Top News Source for the "Word" category, with decay time $\rho = 24$h and 48h and $\beta = 0.5$. Note that for the same value of β a greater decay time ρ gives us smoother functions and higher rank values. However, it does not change the order of the most authoritative sources.

Ranking news articles and news sources

The second group of experiments addresses the principal goal of this Chapter, that is to say, the problem of ranking news articles and news sources. Figure 5.10 shows the evolution of the rank of the top four sources in the category "World" over a period of 55 days. The two plots are obtained by choosing a combination of $\beta = 0.5$ and $\rho = 24{,}48$ hours. RedNova [232] results as the most authoritative source, followed by Yahoo! World[197], Reuters World [233] and BBC News World [215][1]. We observed that the most authoritative sources remain the same when both ρ and β have been changed.

Source	# Postings
RedNova general	3154
Yahoo World	1924
Reuters World	1363
Yahoo Politics	900
BBC News world	1368
Reuters	555
Xinhua	339
New York Times world	549
Boston Globe world	357
The Washington Post world	320

Figure 5.11: Top ten news sources for the category "World" ($\rho = 24$h and $\beta = 0,2$). The second column contains the number of news articles posted by each news agency. Note that "Yahoo Politics" is considered more important than "BBC News world", regardless of the number of news items posted.

In Figure 5.11 we report the top ten news sources for the category "World" returned by our algorithm setting $\rho = 24$ hours and $\beta = 0.2$. Please note that "Yahoo Politics" is considered to be more important than "BBC News world" due to the 2. We remark, that these ranks reflect the results of a computer algorithm, rather than the opinion of the author. Similar behaviour is shown by the other categories, as well.

[1] We remark that these ranks express the results of a computer algorithm, and they do not express the author's opinion.

In Figure 5.12, 5.13 we report the top ten news articles for categories "World" and "Sports", using $\rho = 24$ hours and $\beta = 0.2$. The news posting in these tables are those which score a higher absolute rank over the period of observation. Note that our algorithm ranks any posted articles, and for top pieces of news it is common to recognize re-issues of the same piece of information in the top list. In these cases, our algorithm prefers fresher news articles and those posted by more authoritative news agencies.

Posted	News Source	News Abstract
10/11	RedNova general	Israeli Airstrike Kills Hamas Militant
10/11	RedNova general	Frederick Gets 8 Years in Iraq Abuse Case
10/5	RedNova general	Kerry Warns Draft Possible if Bush Wins
9/8	RedNova general	Iran Says U.N. Nuclear Ban 'Illegal'
9/12	RedNova general	Video Shows British Hostage Plead for Life
10/11	Yahoo World	Israeli Airstrike Kills Hamas Militant (AP)
9/11	RedNova general	Web Site: 2nd U.S. Hostage Killed in Iraq
9/19	RedNova general	British Hostage in Iraq Pleads for Help
9/22	Yahoo World	Sharon Vows to Escalate Gaza Offensive (AP)
9/16	Channel News Asia	Palestinian killed on intifada anniversary

Figure 5.12: Top ten news articles during the total observation period for the category "World" (ρ =24h and $\beta = 0.2$).

Posted	News Source	News Abstract
8/17	Reuters	Argentina Wins First Olympic Gold for 52 Years
8/18	Reuters	British Stun US in Sprint Relay
8/18	NBCOlympics	Argentina wins first basketball gold
9/9	Reuters Sports	Monty Seals Record Ryder Cup Triumph for Europe
8/18	Reuters Sports	Men's Basketball: Argentina Beats Italy, Takes Gold
10/11	Yahoo Sports	Pot Charge May Be Dropped Against Anthony (AP)
10/10	Reuters Sports	Record-Breaking Red Sox Reach World Series
8/17	China Daily	China's Xing Huina wins Olympic women's 10,000m gold
8/17	Reuters Sports	El Guerrouj, Holmes Stride Into Olympic History
8/18	Reuters Sports	Hammer Gold Medallist Annus Loses Medal

Figure 5.13: Top ten news articles during all the observation period for the category "Sports" (ρ =24h and $\beta = 0.2$).

In Figure 5.14 and 5.15, the top ten fresh news articles for the category "World" and "Sports" are listed for the last day of observation. In these lists it is possible to recognize the posting of news articles regarding the same event and due to the fact that these news articles are all fresh, the ranking essentially depends on the rank of the source.

Posted	News Source	News Abstract
10/11	RedNova general	Israeli Airstrike Kills Hamas Militant
10/11	RedNova general	Frederick Gets 8 Years in Iraq Abuse Case
10/11	CNN International	Israeli airstrike kills top Hamas leader
10/11	Yahoo Politics	Bush Criticizes Kerry on Health Care (AP)
10/11	RedNova general	Man Opens Fire at Mo. Manufacturing Plant
10/11	Yahoo Politics	Bush, Kerry Spar on Science, Health Care (AP)
10/11	Yahoo Politics	Smith Political Dinner Gets Bush, Carey (AP)
10/11	RedNova general	AP Poll: Bush, Kerry Tied in Popular Vote
10/11	Yahoo World	Fidel Castro Fractures Knee, Arm in Fall (AP)
10/11	Boston Globe	US Army Reservist sentenced to eight years for Abu Ghraib abuse

Figure 5.14: Top ten news articles the last day of the observation period for the category "World" (ρ =24h and $\beta = 0.2$), only fresh news articles are present.

Posted	News Source	News Abstract
10/11	Yahoo Sports	Pot Charge May Be Dropped Against Anthony (AP)
10/11	Yahoo Sports	Anthony Leads Nuggets Past Clippers (AP)
10/11	NDTV.com	Tennis: Top seeded Henman loses to Ivan Ljubicic
10/11	Reuters	UPDATE 1-Lewis fires spectacular 62 to take Funai lead
10/11	Reuters Sports	Cards Secure World Series Clash with Red Sox
10/11	Yahoo Sports	Court: Paul Hamm Can Keep Olympic Gold (AP)
10/11	Yahoo Sports	Nuggets' Anthony Cited for Pot Possession (AP)
10/11	Reuters	Chelsea won't sack me, says Mutu
10/11	Reuters Sports	Record-Breaking Red Sox Reach World Series
10/11	Yahoo Sports	Dolphins Owner Undecided About Coach, GM (AP)

Figure 5.15: Top ten news articles the last day of the observation period for the category "Sports" (ρ =24h and $\beta = 0.2$), only fresh pieces of news are present.

Ranking news events

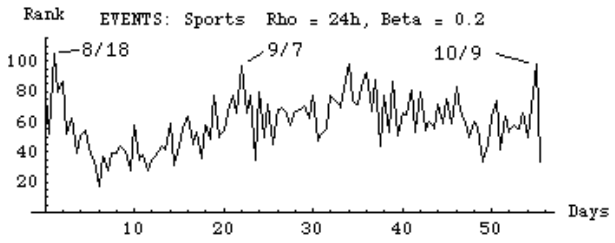


Figure 5.16: For the category "Sports" a plot of the function $\mu(t)$ is represented. Peaks correspond to particularly significant events.

In Figure 5.16 the function $\mu(t)$ defined in (5.8) is plotted over time. The value at time t represents the mean of the source ranks in the "Sports" category, hence peaks may correspond to

particularly significant events.

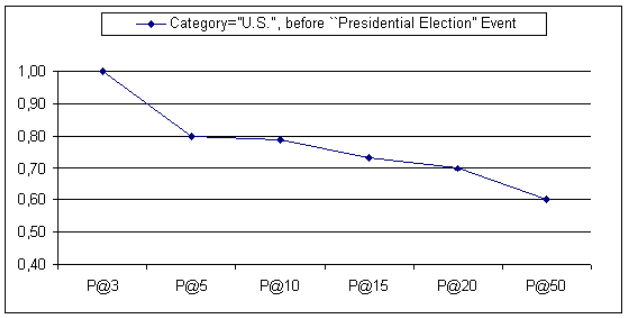


Figure 5.17: P@N for "U.S." during the period of observation.

Evaluating Precision

A further interesting measure would be to consider the quality of ranked news articles. To perform this evaluation we consider the standard P@N measure over the news stories, defined as $P@N_{news} = \frac{|C \bigcap R|}{|R|}$ where , R is the subset of the N top news articles returned by our algorithm, and C is the set of relevant manually tagged postings. More specifically, we fixed a particular observation window over the data stream of news articles and the news items, before asking three people to manually assess the relevance of the top articles by taking into account the particular time instants and the category to which the pieces of news belonged. Only the precision of the final algorithm in Section 5.5 has been evaluated since the earlier variations of the algorithm do not satisfy the mathematical requirements given in Section 5.2. In Figure 5.17 we give the P@N reported for the top news articles in the category "U.S." during the period of observation.

5.8 Conclusions

In this Chapter we have presented an algorithm for ranking news articles and news sources. The algorithm has been constructed step-by-step, ruling out simpler ideas that failed to bring results when used for intuitive cases. Our research was motivated by what is currently a significant interest in commercial news engines and results from a noticeable lack of academic study in this area. An extensive testing of more than 300,000 news items, posted by 2000 sources over two months has been performed, showing very encouraging results both for news articles and news sources.

In a future work, we plan to privilege those news articles produced in the past which have many similar a pieces of information produced after them. The key intuition is that we may want to mitigate the decay of importance of those articles that are talking about a currently hot topic, even when they have been produced in the past. We believe that this is one of the most convenient way of controlling the decay of importance. Another option is to monitor the number of times than an article has been clicked upon. This last alternative requests a certain initial latency for accumulating enough click information before being useful.

The methodology proposed in this Chapter has a larger area of application than just the ranking of news articles and press agencies, and we plan to exploit the ideas discussed in this Chapter for other types of problems, such as that of ranking publications, authors and scientific journals or ranking of blog postings and blog sources. In general, the approach proposed is particularly interesting when the user is interested in obtaining fresh information annotated with temporal timestamp, such as news articles, blog postings, usenet posting, emails, fresh web pages, desktop searching. In all these cases, either there is not enough time to have links pointing to fresh information or the concept of hyperlink is not applicable to the specific search domain. Another interesting

line of academic research we plan to further expand is the evaluation of different models for clustering, classifying and personalized delivering of the news items currently collected by comeToMyHead.

The line of research discussed in this Chapter has been recently elaborated in [90, 140, 227].

A system for visualizing of large amounts of news stories has been presented in [90]. The system is not concerned with ranking, but presents an alternative to our approach for event detection. In the first phase, the news stories are preprocessed for the purpose of name-entity extraction. Next, a graph of relationships between the extracted name entities is created, where each name entity represents one vertex in the graph and two name entities are connected if they appear in the same document. In the future we aim to integrate a similar approach with our clustering algorithm.

Another recent work is [140], where the discovery of previously unidentified events in a historical news corpus is discussed. This approach is complementary to ours which is stream based.

The most relevant result which is similar to ours was given in a patent filed by Google [227]. The patent discusses different methodologies for news articles ranking. Like our approach, the one followed by GOOGLE "ranks the list of (search results) based at least in part on a quality of (their) identified sources". They also claim that: "there exists a need for systems and methods for improving the ranking of news articles based on the quality of the news source with which the articles are associated." Another similar concept is the idea that a news source posting many "breaking-news stories" should be boosted in ranking. In addition, Google generates results based on the relevance of content to a particular term, and on the time at which any particular news article is first published on-line. We remark that our study has been developed at the same time and independently from that of GOOGLE. The patent was filed before our study, but it was only disclosed to the public audience after our papers were produced [53, 94].

Chapter 6

A Methodology for Fast Pagerank Computation

Simplicity is the ultimate sophistication, Leonardo da Vinci

Abstract

Link Analysis was the breakthrough technique used by search engines to rank results. This Chapter is devoted to the discussion of methodologies for accelerating Pagerank. This study has been motivated by the dynamic nature of the Web. In fact, it has been observed that the Web changes very rapidly, with more than 25% of links being changed and a 5% increase in "new content" being created [45]. This information indicates that search engines need to update link-based ranking metrics on a regular basis, and that a week-old ranking may not sufficiently reflect the current importance of the pages. The results have a more general application, since Pagerank has also become an useful paradigm of computation in many Web search algorithms, such as spam, collusion detection or trust networks, where the input graphs have different levels of granularity or different link weight assignments.

Using numerical techniques, we will demonstrate that Pagerank may be transformed into a linear system of equations, where the coefficient matrix is as sparse as the Web graph itself. Then, we will evaluate the effect of many different permutations on the matrix and on each of the rearranged matrices, by applying several solving methods such as Power, Jacobi, Gauss-Seidel and Reverse Gauss-Seidel, to each of the rearranged matrices. In addition, the disclosed structure of the permuted matrix makes it possible for us to use block methods, which are actually more powerful than scalar ones. We have tested the above methods on a Web graph of approximately 24 million nodes and more than 100 million links crawled from the net. Our best result, achieved by a block method, is a 65% reduction in Mflops and a 92% reduction in time in relation to the Power method, taken as a reference method in computing Pagerank.

Ideas similar to the one proposed in this Chapter have also been proposed independently by YAHOO![87, 86] at the same time as our publications in [51, 52].

6.1 Our contribution

Link Analysis were the breakthrough algorithms used by search engines to rank results. This Chapter is devoted to a discussion of methodologies for accelerating Pagerank [27, 159]. Is has been noted that the Web changes rapidly, with more than 25% of links being changed and a 5% increase in "new content" being created [45] over the course of a week. This information indicates that search engines need to update link-based ranking metrics on a regular basis, and that a week-old ranking may not sufficiently reflect the current importance of the pages. In addition, it indicates that a ranking algorithm based on incremental updates cannot provide good results. For these reasons, the research community has recently been directing its attention to the matter of reducing the computational time required by Pagerank.

We point out that Pagerank has also become a useful paradigm of computation in many Web search algorithms, such as spam detection [98] or trust networks [115] where the input graphs

have different levels of granularity (HostRank) or different link weight assignments (internal, external, etc). There are also situations when it is useful to assign many Pagerank values to each Web page. For instance, [101, 103, 111] suggest computing a different Pagerank for each page according to different users profiles, and Google recently presented the service as a beta-version (see http://labs.google.com/personalized/). Another similar is raised in [190], which demonstrates some heuristics for collusion-proof Pagerank algorithms requiring the computation of many different Pagerank vectors with respect to the different choices of parameters. For each algorithm, the critical computation is a Pagerank-like vector of interest. Thus, methods for accelerating and parallelizing Pagerank have a more general impact.

The core of Pagerank exploits an iterative weight assignment of ranks to Web pages until a fixed point is reached and we can see that this fixed point is the (dominant) eigenpair of a matrix obtained by the Web graph itself. Brin and Page originally suggested that this pair might be computed using the well-known Power method [88], and we would also like to refer you to their fine interpretation of Pagerank in relation to Markov Chains.

Previous approaches to accelerating Pagerank have branched off in various different directions, for example, an attempt to compress the Web graph so that it may fit into the main memory [24], or the implementation of the algorithms in the internal memory [100, 40] (cfr. Chapter 3)

One very interesting line of research is one which exploits efficient numerical methods in order to reduce computation time. In our opinion, these types of numerical techniques are the most promising and in recent years we have seen many intriguing Power iterations acceleration results [117, 100, 137]. Models are presented in literature [7, 137, 159] which treat pages with no out-links in varying ways.

In this Chapter we shall consider the original Pagerank model already introduced in Chapter 3. Mathematically, this corresponds to defining a matrix $\tilde{P}$ as

$$\tilde{P} = \alpha \bar{P} + (1 - \alpha)\ \mathbf{e}\mathbf{v}^T, \tag{6.1}$$

where $\mathbf{e}$ is the vector with all entries equal to 1, and α is a constant, $0 < \alpha < 1$. At each step, the random surfer follows the transitions described by $\bar{P}$ with probability α, while with probability $(1 - \alpha)$ (s)he "bothers to follows links" and jumps to any other node in V according to the personalization vector $\mathbf{v}$. The matrix $\tilde{P}$ is stochastic and irreducible, and both of these conditions imply that the Pagerank vector $\mathbf{z}$ is the unique steady state distribution of the matrix $\tilde{P}$ such that

$$\mathbf{z}^T \tilde{P} = \mathbf{z}^T. \tag{6.2}$$

From (6.1) it turns out that the matrix $\widehat{P}$ is explicitly

$$\tilde{P} = \alpha(P + \mathbf{d}\mathbf{v}^T) + (1 - \alpha)\ \mathbf{e}\mathbf{v}^T. \tag{6.3}$$

The most common numerical method for solving the eigenproblem (6.2) is the Power method [88]. Since $\tilde{P}$ is a rank one modification of αP, it is possible to implement a Power method which multiplies only the sparse matrix P by a vector and, at each step, upgrades the intermediate result with a constant vector, as suggested by Haveliwala in [100].

However, the eigenproblem (6.2) can be rewritten as a linear system. By substituting (6.3) into (6.2) we get $\mathbf{z}^T(\alpha P + \alpha \mathbf{d}\mathbf{v}^T) + (1-\alpha)\mathbf{z}^T\mathbf{e}\mathbf{v}^T = \mathbf{z}^T$, which means that the problem is equivalent to the solution of the following linear system of equations

$$S\mathbf{z} = (1 - \alpha)\mathbf{v}, \tag{6.4}$$

where $S = I - \alpha P^T - \alpha \mathbf{v}\mathbf{d}^T$, and we make use of the fact that $\mathbf{z}^T\mathbf{e} = \sum_{i=1}^{N} z_i = 1$. The transformation of the eigenproblem (6.2) into (6.4) opens the way to a large variety of numerical methods which are not fully investigated in the bibliography. In the next Section we will present a lightweight solution to handling the non-sparseness of S.

By using numerical techniques, we demonstrate that this problem may be transformed into an equivalent linear system of equations, where the coefficient matrix is as sparse as the Web

graph itself. Moreover, since many numerical iterative methods for a linear system solution may benefit from a reordering of the coefficient matrix, we have rearranged it by increasing the data locality and reducing the number of iterations required by the solving methods (see Section 6.3). In particular, we will evaluate the effect of many different permutations on the matrix and on each of the rearranged matrices, by applying several methods such as Power, Jacobi, Gauss-Seidel and Reverse Gauss-Seidel [179], to each of the rearranged matrices.

The disclosed structure of the permuted matrix makes it possible for us to use block methods, which are actually more powerful than scalar ones. Please note that the phase of reordering a matrix according to a given permutation requires additional computational effort, however, the time spent reordering the matrix is negligible in relation to the time required to solve the system. One important consideration to be made is that the same reordering can be used to solve several Pagerank problems with different personalization vectors. We have tested the above methods on a Web graph of approximately 24 million nodes and more than 100 million links crawled from the net. Our best result, achieved by a block method, is a 65% reduction in Mflops and a 92% reduction in time in relation to the Power method, taken as a reference method in computing the Pagerank vector. This study is also described in the following papers [51, 54, 52].

Key Contributions
1. True sparse linear system formulation for PageRank
2. Markov chain indexes reordering for fast computation
3. Testing of many different solving strategies

Figure 6.1: Key contributions of this Chapter

6.2 A sparse linear system formulation

In this Section, we will demonstrate how the Pagerank vector can be computed as the solution of a sparse linear system. We will remark, that the way in which the dangling node is handled is crucial, for the reason that the nodes can reach vast numbers.

According to Kamvar et al. [119], a 2001 sample of the Web containing 290 million pages had only 70 million non-dangling nodes. This mass of nodes without outlinks includes both those pages which do not point to any other page, and also those whose existence is inferred by hyper-links but not yet reached by the crawler. A dangling node can represent pdf, ps, txt or any other file format gathered by a crawler, with no hyper-links pointing outside.

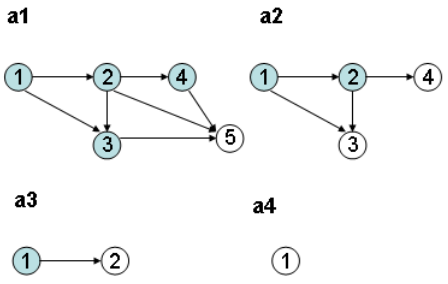


Figure 6.2: Removing the dangling node in figure (a1) generates new dangling nodes, which are in turn removed (a2, a3, a4). By the end of the process, no node has received a Pagerank assignment.

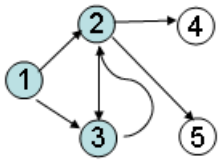
Page et al. [159], adopted the drastic solution of removing the dangling nodes completely. In this way, the size of the problem is reduced, but a considerable amount of information available on the Web is ignored. This has had an impact on both the dangling nodes - which are simply not ranked - and on the remaining nodes. In fact, we should note that the removal of the dangling

nodes changes the ranking of the Web Graph since it alters the transition probability matrix: imagine an "important" node that used to have 100 out-edges, but after the change has only 2: the mass of its remaining neighbors will considerably increase.

Moreover, removing this set of nodes could potentially create new dangling nodes, which in turn would have to be removed (see fig 6.2). Therefore, the nodes with an assigned Pagerank could account for a small percentage of the Web graph. Formally, hremoving dangling nodes recursively results in a uniquely defined sub-graph called the 1-out core. We called the rest of the graph the "tail of the matrix". In [54], we estimated the tail of our large crawled Web matrix to something more than 2 Million pages with more than 5 Million nonzeros (links).

Arasu et al. [7] handled dangling nodes in a different way compared to the natural model presented in Section 6.1. They modified the Web graph by imposing that every dangling node has a self loop. In terms of matrices, $\bar{P} = P + F$ where $F_{ij} = 1$ iff $i = j$ and $\text{outdeg}(i) = 0$. The matrix $\bar{P}$ is row stochastic and the computation of Pagerank is solved using a random jump similar to the equation (6.3), where the matrix F replaces D. This model is different from the natural model, as it is evident from the following example.

Example 6.2.1 *Consider the graph in Figure 6.3 and the associated transition matrix. The Pagerank obtained, by using the natural model, orders the nodes as* $(2, 3, 5, 4, 1)$. *Arasu's model orders the nodes as* $(5, 4, 2, 3, 1)$. *Note that in the latter case, node 5 ranks better than node 2, which is not what one expects.*



$$P = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Figure 6.3: An example of graph whose rank assignment differs if the dangling nodes are treated as in the model presented by Arasu et al. in [7].

From the above observations we believe that it is important to take into account the dangling nodes, and we consider the natural model as the one which best captures the behaviour of a random surfer. The dense structure of the matrix S poses serious problems to the solution of the linear system (6.4), and for this reason, the matter has been largely addressed as an eigenproblem, as we have seen very few attempts to solve the problem formulated as a linear system. Computing the Pagerank vector with the Power method allows the sparsity of the matrix P to be exploited. In fact, it is common to implement the Power method in such a way that the matrix-vector multiplications involve only the sparse matrix P, while the rank-one modifications are handled separately [100].

In the following, we shall demonstrate the way in which the dangling nodes may be managed in a direct and lightweight manner, so as to make it possible to use iterative methods for linear systems. In particular, we will formally prove the equivalence of (6.4) to the solution of a system involving only the sparse matrix $R = I - \alpha P^T$. Next theorem makes use of a very powerful tool: the Sherman-Morrison formula (see [88], paragraph 2.1.3). It is well known that Sherman-Morrison formula is unstable, however we are using it here only as an elegant technique for proving the theorem without any need to implement it.

Theorem 6.2.2 *The Pagerank vector* $\mathbf{z}$ *solution of* (6.4) *is obtained by solving the system* $R\mathbf{y} = \mathbf{v}$ *and taking* $\mathbf{z} = \mathbf{y}/\|\mathbf{y}\|_1$.

Proof. Since $S = R - \alpha\, \mathbf{v}\mathbf{d}^T$ equation (6.4) becomes $(R - \alpha\, \mathbf{v}\mathbf{d}^T)\mathbf{z} = (1 - \alpha)\, \mathbf{v}$, that is, a system of equations where the coefficient matrix is the sum of a matrix R and a rank-one matrix. Note that R is not singular since $\alpha < 1$ and therefore all the eigenvalues of R are different from zero. We can use the Sherman-Morrison formula (see [88], paragraph 2.1.3), for computing the inverse

of the rank-one modification of R. Consequently, we have

$$(R - \alpha \mathbf{v}\mathbf{d}^T)^{-1} = R^{-1} + \frac{R^{-1}\mathbf{v}\mathbf{d}^T R^{-1}}{1/\alpha + \mathbf{d}^T R^{-1}\mathbf{v}}. \tag{6.5}$$

From (6.5), denoting the solution of the system $R\mathbf{y} = \mathbf{v}$ by $\mathbf{y}$, we have

$$\mathbf{z} = (1-\alpha)\left(1 + \frac{\mathbf{d}^T\mathbf{y}}{1/\alpha + \mathbf{d}^T\mathbf{y}}\right)\mathbf{y},$$

that means that $\mathbf{z} = \gamma\mathbf{y}$, and the constant $\gamma = (1-\alpha)\left(1 + \frac{\mathbf{d}^T\mathbf{y}}{1/\alpha+\mathbf{d}^T\mathbf{y}}\right)$ can be computed normalizing $\mathbf{y}$ in such a way $\|\mathbf{z}\|_1 = 1$. ∎

To summarize, we have shown that in order to compute the Pagerank vector $\mathbf{z}$, we can solve the system $R\mathbf{y} = \mathbf{v}$, and then normalize $\mathbf{y}$ to obtain the Pagerank vector $\mathbf{z}$. This means that the rank one matrix D in the Pagerank model which accounts for the dangling pages, plays a role only in the scaling factor γ. Moreover, the computation of γ is not always necessary and this step can be occasionally be omitted.

Note that the matrix used by Arasu and al. [7] is also sparse due to the way they deal with the dangling nodes, but the Pagerank obtained doesn't rank the nodes in a natural way (see Example 6.3). Instead, our approach guarantees a more natural ranking and handles the density of S by transforming a dense problem into one which uses the sparse matrix R.

Bianchini and al. in [20] prove that the iterative method derived from (6.2) and involving $\tilde{P}$ which produces the same sequence of vectors of the Jacobi method applied to matrix R. Recently Eiron et al. [67] proposed another way of dealing with dangling nodes. They assigned a rank to dangling and non dangling pages separately and their algorithm requires the knowledge of a complete strongly connected subgraph of the Web.

6.2.1 The conditioning of the problem in the new formulation

When solving a linear system, particular attention must be payed to the conditioning of the problem, the magic number accounting for the "hardness" of solving a linear system. The condition number of a matrix A is defined as $\mathrm{cond}(A) = \|A\|\,\|A^{-1}\|$ for any matrix norm. It is easy to show, as proved by Kamvar and Haveliwala [116], that the condition number in the 1-norm of S is $\mathrm{cond}(S) = \frac{1+\alpha}{1-\alpha}$, which means that the problem tends to become ill-conditioned as α goes to one. For the conditioning of R we can prove the following theorem.

Theorem 6.2.3 *The condition numbers expressed in the 1-norm of matrices S and R are such that*

$$\mathrm{cond}(R) \leq \mathrm{cond}(S).$$

Moreover, the inequality is strict if and only if from every node there is a direct path to a dangling node.

Proof. In order to prove the theorem we have to show that $\mathrm{cond}(R) \leq \frac{1+\alpha}{1-\alpha}$. We have

$$\|R\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{n} |r_{ij}| = \max_{j=1,\ldots,n} \left(1 + \alpha \sum_{i=1, i\neq j}^{n} p_{ji}\right).$$

Then, if $P \neq O$, $\|R\|_1 = 1 + \alpha$. Note that R^{-1} is a non negative matrix, hence

$$\|R^{-1}\|_1 = \|\mathbf{e}^T R^{-1}\|_1 = \|R^{-T}\mathbf{e}\|_\infty.$$

Moreover,

$$R^{-T} = (I - \alpha P)^{-1} = \sum_{i=0}^{\infty} \alpha^i P^i. \tag{6.6}$$

Since every entry of the vector $P\mathbf{e}$ is less or equal 1, we have $P^i\mathbf{e} \leq \mathbf{e}$, hence

$$\|R^{-1}\|_1 = \|\sum_{i=0}^{\infty} \alpha^i P^i \mathbf{e}\| \leq \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha},$$

which proves that $\text{cond}(R) \leq \text{cond}(S)$.

Let us now prove that the inequality is strict if it is possible to reach a dangling node with Markov chain state transitions, from every page. Since P has dangling nodes, it is reducible. Let k, $k \geq 1$ be the number of strongly connected components of the Web graph. We can permute rows and columns of P grouping together the nodes belonging to the same connected component and listing the dangling nodes in the last rows. Therefore, P can be expressed in this reduced form

$$P = \begin{bmatrix} P_{11} & P_{1,2} & \cdots & \cdots & P_{1,k} \\ & P_{2,2} & \cdots & & P_{2,k} \\ & & \ddots & & \vdots \\ & & & P_{k-1,k-1} & P_{k-1,k} \\ O & & & & O \end{bmatrix},$$

where the diagonal blocks P_{ii} are irreducible. By hypothesis, from every Web page we can reach a dangling node with a finite number of clicks. In terms of the matrix P, this means that, for every i, $1 \leq i \leq k$, each block P_{ii} has at least a row whose sum is strictly less than 1. An extension of the Gershgorin circle Theorem [179][Theorem 1.7, page 20] ensures that the spectral radius of every diagonal block P_{ii} is less than 1. Since the eigenvalues of P are the eigenvalues of the diagonal blocks this guarantees $\rho(P) < 1$. Let us consider the iterative method $\mathbf{x}_{i+1} = P\mathbf{x}_i$, since $\rho(P) < 1$ this method is convergent to the zero vector, for every choice of the starting vector $\mathbf{x}_0$. Then, choosing $\mathbf{x}_0 = \mathbf{e}$, an integer i exists such that $P^i\,\mathbf{e} < \mathbf{e}$. From (6.6) we have

$$\|R^{-T}\|_1 = \|\sum_{i=0}^{\infty} \alpha^i P^i \mathbf{e}\|_\infty < \|\sum_{i=0}^{\infty} \alpha^i \mathbf{e}\| = \frac{1}{1-\alpha},$$

which proves the "if" part. To prove the "only if" part, lets assume by contradiction that there is at least a page from which it is not possible to reach a dangling page, and lets assume that this page belongs to the h-th connected component. Since P_{hh} is irreducible, this means that from every node in P_{hh} it is not possible to reach a dangling page. Hence, there is at least an index i, $1 \leq i \leq k-1$, such that the strip corresponding to the i-th block is zero, except the diagonal block P_{ii}. This means that the diagonal block P_{ii} has an eigenvalue equal to one, hence $\rho(P) = 1$ and $P_{ii}\mathbf{e}_i = \mathbf{e}_i$, which implies $\text{cond}(R) = (1+\alpha)/(1-\alpha)$. ∎

Theorem 6.2.3 proves that the condition number of matrix R is always less than or equal to the condition number of S. This means that to compute the Pagerank vector by solving the system with matrix R is never worse than computing it by solving the system involving S. In fact, the system in R is less sensitive to the errors due to the finite representation of the numbers appearing in P than the system involving S.

Note that if the condition which guarantees the strict inequality between the condition number of S and R is not satisfied, a reordering of R exists allowing the original problem to split into two disjoint subproblems. As we will see in Section 6.3, this is computationally convenient and moreover, the conditioning of the two subproblems should be compared to the conditioning of the two subproblems regarding S.

6.3 Exploiting the Web matrix permutations

In Section 6.2 we have shown how to transform the linear system involving the matrix S into an equivalent linear system, where the coefficient matrix R is as sparse as the Web graph. The new sparse formulation allows the effectiveness of other iterative procedures to be exploited in

order to compute the Pagerank vector, which previously was not applicable when dealing with S. To solve the linear system $R\mathbf{y} = \mathbf{v}$ two convenient strategies are the Jacobi and Gauss-Seidel methods [179], because they use space comparable to that used by the Power method. These methods are convergent if and only if the spectral radius of the iteration matrix is strictly lower than one. Moreover, the rate of convergence depends on the spectral radius and the lower the radius is the faster the convergence is. Since $R = I - \alpha P$ where P is a nonnegative matrix, R is a so called M-matrix [179], and it is well known that both the Jacobi and Gauss-Seidel methods are convergent and that the Gauss-Seidel method applied to M-matrices is always faster than the Jacobi method [179].

When solving a sparse linear system, a common practice [63] is to look for a reordering scheme that reduces the (semi)bandwidth for increasing data locality and hence the time spent for each iteration. Moreover, if the matrix is reducible, the problem can be split into smaller linear systems which can be solved in cascade.

The simpler permutation scheme is the one which separates dangling from non-dangling pages. In this case, the matrix system involving R has a very simple shape

$$\begin{bmatrix} I - \alpha P_1^T & O \\ -\alpha P_2^T & I \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix}, \tag{6.7}$$

and once the system $(I - \alpha P_1^T)\mathbf{y}_1 = \mathbf{v}_1$ is solved, the vector $\mathbf{y}_2$ is computed with only a matrix vector multiplication as $\mathbf{y}_2 = \mathbf{v}_2 + \alpha P_2^T \mathbf{y}_1$. Two recent papers, one by Eiron et al. [67] and another by Langville and Meyer [130], have come up with the same solution to the problem but use different reasoning. The problem involving P_1, is still solved using the Power method, and they did not conduct a further investigation into other possible reordering schemes which could increase the benefits of permutation strategies.

Note that once R is reordered into a block triangular matrix, a natural way of solving the system is to use forward/backward block substitution. For instance, on the lower block triangular system

$$\begin{bmatrix} R_{11} & & & \\ R_{21} & R_{11} & & \\ \vdots & & \ddots & \\ R_{m1} & \cdots & & R_{mm} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_m \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_m \end{bmatrix},$$

the solution can be computed as follows

$$\begin{cases} \mathbf{y}_1 = & R_{11}^{-1}\mathbf{v}_1, \\ \mathbf{y}_i = & R_{ii}^{-1}\left(\mathbf{v}_i - \sum_{j=1}^{i-1} R_{ij}\mathbf{y}_j\right) \quad \text{for } i = 2, \ldots, m. \end{cases} \tag{6.8}$$

This requires the solution of m smaller linear systems, where the coefficient matrices are the diagonal blocks in the order they appear. Note that this strategy is always better than applying an iterative method to the whole matrix.

Moreover, for some iterative methods, it may happen that a permutation in the matrix reduces the spectral radius of the iteration matrix and hence the number of iterations needed to reach convergence. However, this is not the case for the Jacobi method, since the spectral radius of the iteration matrix is invariable when undergoing permutation. In the same way, the rate of convergence of the Power method is also independent from the matrix reordering, since it only depends on the spectral properties of the matrix and the starting vector. An example of this approach is given in [119] where how the matrix is reordered by sorting the URLs lexicographically is shown, which may help to construct a better starting vector for the Power method and improve data locality.

A rather more challenging approach is Web matrix reordering in the Gauss-Seidel method, where opportune permutations can lead to both an increase in data locality and to an iteration matrix with a reduced spectral radius.

The permutation strategies we have proposed have two different purposes. The primary goal is to increase data locality and to decrease the spectral radius of the iteration matrix, wherever

possible. The secondary goal is to discover a block triangular structure in the Web matrix in order to apply block methods as described in equation (6.8).

Applying a permutation to a sparse matrix as well as finding the permutation which satisfies some desiderata, is a costly operation. However, it is often convenient to make more efforts to decrease the running time of the solving method when the same matrix is used many times to solve different Pagerank problems, as required for personalized Web search [101] or in the case of heuristics for collusion-proof Pagerank [190].

The permutations we have considered are obtained by combining different elementary operations. A very effective reordering scheme, denoted by $\mathcal{B}$, is the one obtained by permuting the nodes of the Web graph according to the order induced by a BFS visit. The BFS visit makes it possible to discover the reducibility of the Web Matrix, since this visit assigns contiguous permutation indexes to pages pointed to by the same source. Therefore, this permutation produces lower block triangular matrices. It has been observed by Cuthill and McKee [55] that the BFS strategy for reordering sparse symmetric matrices produces a reduced bandwidth when the children of each node are inserted in decreasing degree order. For this reason, even if P is not symmetric, we can examine other reordering schemes which are obtained by sorting the nodes in terms of their degree. In particular we consider the permutation which reorders the pages for decreasing out-degree; denoting this scheme as $\mathcal{O}_{\mathrm{d}}$ while the permutation $\mathcal{O}_{\mathrm{a}}$ sorts the pages of the Web matrix for ascending out-degree. Note that these permutations list the dangling pages in the last and in

Full	Lower Block Triangular	Upper Block Triangular
$\mathcal{T}$	$\mathcal{T}\mathcal{B}$	$\mathcal{B}\mathcal{T}$
$\mathcal{O}_{\mathrm{d}}\mathcal{T}$	$\mathcal{O}_{\mathrm{d}}\mathcal{T}\mathcal{B}$	$\mathcal{O}_{\mathrm{d}}\mathcal{B}\mathcal{T}$
$\mathcal{O}_{\mathrm{a}}\mathcal{T}$	$\mathcal{O}_{\mathrm{a}}\mathcal{T}\mathcal{B}$	$\mathcal{O}_{\mathrm{a}}\mathcal{B}\mathcal{T}$
$\mathcal{I}_{\mathrm{d}}\mathcal{T}$	$\mathcal{I}_{\mathrm{d}}\mathcal{T}\mathcal{B}$	$\mathcal{I}_{\mathrm{d}}\mathcal{B}\mathcal{T}$
$\mathcal{I}_{\mathrm{a}}\mathcal{T}$	$\mathcal{I}_{\mathrm{a}}\mathcal{T}\mathcal{B}$	$\mathcal{I}_{\mathrm{a}}\mathcal{B}\mathcal{T}$

Figure 6.4: Web Matrix Permutation Taxonomy

the first rows of the Web matrix respectively. We also tested also the permutations obtained by reordering the matrix by ascending and descending in-degree; denoted by $\mathcal{I}_{\mathrm{a}}$ and $\mathcal{I}_{\mathrm{d}}$ respectively. Since $R = I - \alpha P^T$, our algorithms need to compute the transpose of the Web matrix, this operation is denoted by $\mathcal{T}$. The various operations can be combined obtaining different reorderings of the Web matrix as shown in Figure 6.4. In accordance with the taxonomy in Figure 6.4, we denote, for instance, by $R_{\mathcal{I}_{\mathrm{d}}\mathcal{T}\mathcal{B}} = I - \alpha\Pi(P)$, where the permuted matrix $\Pi(P)$ is obtained first by applying the $\mathcal{I}_{\mathrm{d}}$ permutation, then by transposing the matrix and applying finally the $\mathcal{B}$ permutation to the matrix reordered. The first column in Figure 6.4 produces full matrices, while the second and third columns produce block triangular matrices due to the BFS's visit order.

In Figure 6.5, we show a plot of the structure of a Web matrix rearranged according to each item of the above taxonomy. It is important to observe that on non symmetric matrices the BFS visit order transforms the matrix into a lower block triangular form, with a number of diagonal blocks which is greater than or equal the number of strongly connected components. However, the number of diagonal blocks detected with a BFS depends very much on the starting node of the visit. For example in Figure 6.6 starting from node 1 we detect just one component, while starting from node 4 we get four separate components.

In order to have diagonal blocks of a smaller size, and to split the problem into smaller subproblems, we have investigated other permutations exploiting further the reducibility of the matrix. Let $\mathcal{J}$ denote the reverse permutation, that is the permutation which assigns the index $n - i$ to node i-th. We can apply another $\mathcal{B}$ operation, sometimes after a reversion of the matrix with the operator $\mathcal{J}$, to some of the shapes in Figure 6.5. In this way, we have obtained the shapes in Figure 6.7.

Note that the smallest size of the largest diagonal block is achieved by $R_{\mathcal{O}_{\mathrm{a}}\mathcal{B}\mathcal{T}\mathcal{J}\mathcal{B}}$. The size of the largest component in $R_{\mathcal{O}_{\mathrm{a}}\mathcal{B}\mathcal{T}\mathcal{J}\mathcal{B}}$ is something more than 13 Million, while for the permutations in Figure 6.4 which only uses a BFS, we were able to reach a size of around 17 Million.

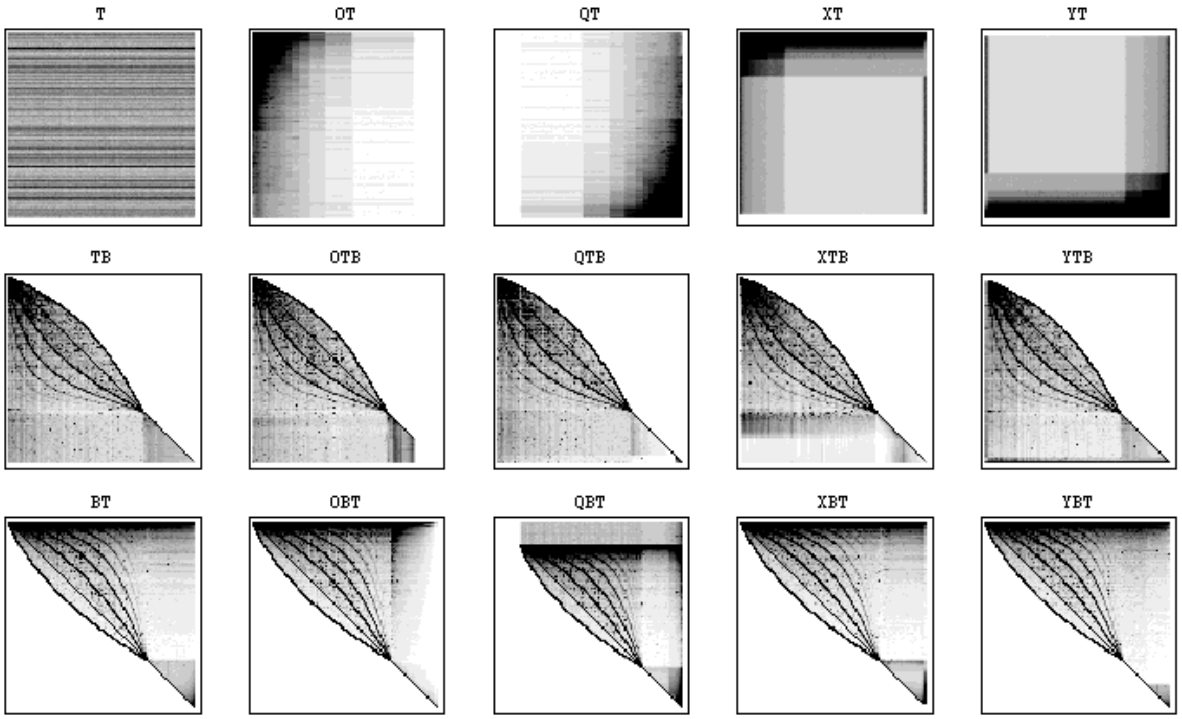
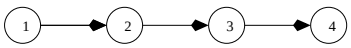


Figure 6.5: Different shapes were obtained by rearranging the Web matrix P in accordance to the taxonomy. The first row represents full matrices; the second and third lower and upper block triangular matrices. The Web graph is made up of 24 million nodes and 100 million links.



$$R_1 = \begin{pmatrix} 1 & -\alpha & 0 & 0 \\ 0 & 1 & -\alpha & 0 \\ 0 & 0 & 1 & -\alpha \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\alpha & 1 & 0 & 0 \\ 0 & -\alpha & 1 & 0 \\ 0 & 0 & -\alpha & 1 \end{pmatrix}.$$

Figure 6.6: An example of a connected graph. Applying a BFS visit starting with node 1 one just detect a component and R_1 is in an unreduced form, while starting with node 4 we get four different components and R_4 is fully reduced.

We adopted ad hoc numerical methods for dealing with the different shapes of the matrices in Figure 6.5 and 6.7. In particular, we compared the Power method and Jacobi iterations with the Gauss-Seidel, and the Reverse Gauss-Seidel. We recall that the Gauss-Seidel method computes $y_i^{(k+1)}$, the $i-$th entry of the vector at the $(k+1)-$th iteration step as a linear combination of $y_j^{(k+1)}$ for $j = 1, \ldots, i-1$ and of $y_j^{(k)}$ for $j = i+1, \ldots, n$. On the contrary, the Reverse Gauss-Seidel method computes the entries of the vector $\mathbf{y}^{(k+1)}$ bottom up, that is, it computes $y_i^{(k+1)}$ for $i = n, \ldots, 1$ as a linear combination of $y_j^{(k+1)}$ for $j = n, \ldots, i+1$ and of $y_j^{(k)}$ for $j = 1, \ldots, i-1$. Note that $R_{\mathcal{O}_{\rm d}\mathcal{T}} = JR_{\mathcal{O}_{\rm a}\mathcal{T}}J^T$ and $R_{\mathcal{I}_{\rm d}\mathcal{T}} = JR_{\mathcal{I}_{\rm a}\mathcal{T}}J^T$ where J is the anti-diagonal matrix, that is $J_{ij} = 1$ iff $i + j = n + 1$. This means that applying Gauss-Seidel to $R_{\mathcal{O}_{\rm d}\mathcal{T}}$ ($R_{\mathcal{I}_{\rm d}\mathcal{T}}$) is the same as applying Reverse Gauss-Seidel to $R_{\mathcal{O}_{\rm a}\mathcal{T}}$ ($R_{\mathcal{I}_{\rm a}\mathcal{T}}$).

The shapes of some of the matrices in Figures 6.5 and 6.7, encourage matrix's reducibility to be exploited by experimenting with block methods. Moreover, also the matrix $R_{\mathcal{O}_{\rm d}\mathcal{T}}$ is a lower block triangular, since it separates non-dangling nodes from dangling nodes. This matrix is a particular case of the one considered in equation (6.7).

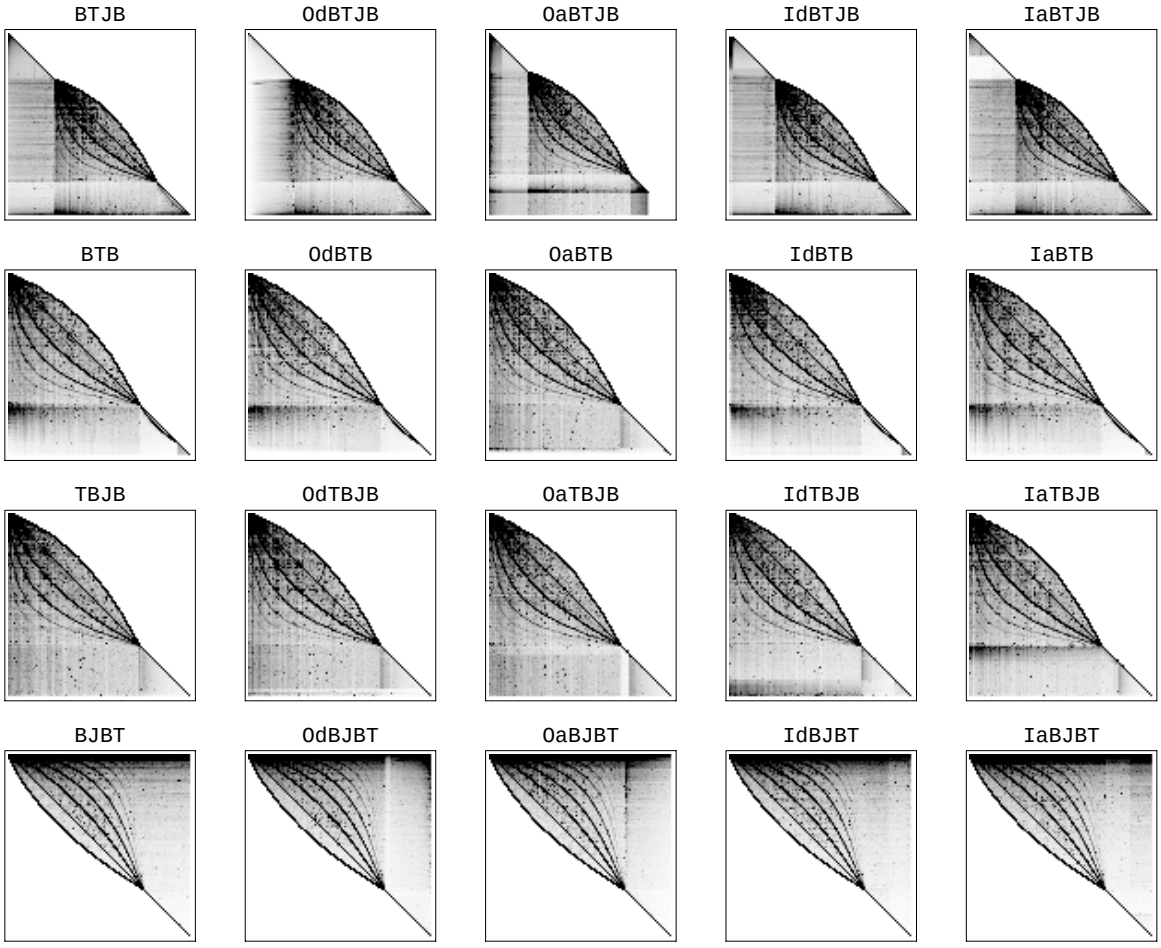


Figure 6.7: Different shapes obtained rearranging the Web matrix P made up of 24 million nodes and 100 million links according to permutations involving two BFS visits.

We tested both the Gauss-Seidel and the Reverse Gauss-Seidel methods, as solving methods for the diagonal block systems. We denote by `LB` and `UB` the methods obtained using Gauss-Seidel as the solver of the diagonal blocks on lower or upper block structures respectively. `LBR` and `UBR` use instead Reverse Gauss-Seidel to solve the diagonal linear systems. In sum, the solution strategies taxonomy is reported in Figure 6.8. In Section 6.4 we report the experimental results obtained by applying each method in Figure 6.8 to all the suitable matrices in Figure 6.5 and in Figure 6.7.

6.4 Experimental results

We have tested the methods discussed in the previous Sections using Web graphs obtained by crawling 24 million Web pages, containing about 100 million hyper-links and approximately 3 million dangling nodes. This data set was donated to us by the Nutch project (see http://www.nutch.org/). We have run our experiments on a PC with a Pentium IV 3GHz, 2.0GB of memory and 512Kb of L2 cache. A stopping criterion of 10^{-7} is imposed on the absolute difference between the vectors computed in two successive iterations. In order to have a fair comparison for the Power method, the tolerance has been changed "a posteriori" to obtain the same absolute error of the other methods. In fact, in the case of the Power method, the Pagerank vector has 1-norm equal to one while

Scalar methods	shapes
PM	all
Jac	all
GS	all
RGS	all

Block methods	shapes
LB	$R_{*\mathcal{B}}$ and $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$
LBR	$R_{*\mathcal{B}}$ and $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$
UB	$R_{*\mathcal{T}}$
UBR	$R_{*\mathcal{T}}$

Figure 6.8: Numerical Methods Taxonomy. PM is the Power method, Jac denotes the Jacobi method, GS and RGS are the Gauss-Seidel and Reverse Gauss-Seidel, respectively. All of them can be applied to each transformation of the matrix according to the taxonomy 6.4. Among block-methods we have LB and LBR which can be applied to all lower block triangular matrices and use GS or RGS to solve each diagonal block. Similarly, UB and UBR refer to the upper block triangular matrices.

Name	PM	GS	RGS	LB/UB	LBR/UBR
$\mathcal{T}$	3454 33093 152	1774 19957 92	1818 20391 94	– – –	– – –
$\mathcal{O}_{\mathrm{d}}\mathcal{T}$	2934 33093 152	1544 20825 85	1477 19957 104	1451 19680 96	1397 18860 92
$\mathcal{I}_{\mathrm{d}}\mathcal{T}$	3315 33309 153	1840 21259 98	1735 19957 92	– – –	– – –
$\mathcal{TB}$	1386 32876 151	606 21910 101	519 18439 85	401 16953 100	359 15053 82
$\mathcal{O}_{\mathrm{d}}\mathcal{TB}$	1383 33093 152	582 21476 99	505 18439 85	392 17486 97	369 15968 85
$\mathcal{O}_{\mathrm{a}}\mathcal{TB}$	1353 32876 151	610 23645 109	440 16920 78	424 18856 106	**315 13789 75**
$\mathcal{I}_{\mathrm{d}}\mathcal{TB}$	1361 33309 153	561 21259 98	511 19090 88	385 17196 98	380 16414 87
$\mathcal{I}_{\mathrm{a}}\mathcal{TB}$	1392 32876 151	619 22343 105	**450 16270 75**	400 17972 100	314 13905 75
$\mathcal{BT}$	1394 33093 152	522 18439 85	640 22560 104	379 15545 85	479 19003 104
$\mathcal{O}_{\mathrm{d}}\mathcal{BT}$	1341 33309 153	507 18873 87	579 21693 100	398 15937 87	466 18312 100
$\mathcal{O}_{\mathrm{a}}\mathcal{BT}$	1511 33093 152	591 18873 87	680 21693 100	357 15128 87	413 17387 100
$\mathcal{I}_{\mathrm{d}}\mathcal{BT}$	1408 33093 152	554 19306 89	626 21693 100	397 16075 88	450 18265 100
$\mathcal{I}_{\mathrm{a}}\mathcal{BT}$	1351 33093 152	497 18439 85	575 21693 100	386 15564 85	447 18310 100

Figure 6.9: Experimental Results: The numerical methods analyzed are listed in the columns and the rows describe some of the permutations applied to the matrix R. Each cell represents the running time in seconds, the number of Mflops and the number of iterations taken by the solving methods. Note that the results in the last two columns account for either the cost of the LB and LBR methods, applied to lower block triangular matrices, or for the cost of UB and UBR methods, applied to upper block triangular matrices. In bold we have highlighted our best results in terms of Mflops for scalar and block methods.

for the other methods we do not scale the vector at each step.

6.4.1 Evaluating the ranking algorithms

In Figure 6.9 we report the running time in seconds, the Mflops and the number of iterations for each combination of solving and reordering the methods described in Figure 6.8 on the matrices in Figure 6.5. We believe that the number of iterations and the number of floating point iterations of a method are fairer measures than the running time in seconds which is more implementation-dependent. However, the running time is the only factor accounting for an increase in data-locality when a permutation of the matrix does not change the number of iterations.

Some cells of equation 6.9 are empty since there are methods only suitable for particular shapes. Moreover, in Figure 6.9 the results in the last two columns are relative to LB and LBR methods for lower block triangular matrices and UB or UBR for upper block triangular matrices. We do not report in the table the behaviour of the Jac method, since it has always performed worse than the GS method. However, the measure of the gain using the GS rather than the Jac can be obtained from Figure 6.11. Since the diagonal entries of R are equal to 1, the Jacobi method is essentially equivalent to the Power method. In our case, the only difference is that Jac is applied to R while PM works on $\tilde{P}$, which incorporates the rank one modification accounting for the dangling nodes. Although we implemented PM using the optimizations suggested in [159, 100], this method requires a slightly greater number of operations. Using the results of Theorem 6.2.2, we get a reduction in

Name	GS	RGS	LB/UB	LBR/UBR
$\mathcal{TBJB}$	565 21476 99	488 18439 85	402 17534 99	327 13980 76
$\mathcal{O}_{\mathrm{d}}\mathcal{TBJB}$	613 23645 109	445 16920 78	426 18853 106	318 13790 76
$\mathcal{O}_{\mathrm{a}}\mathcal{TBJB}$	547 21476 99	478 18656 86	390 17891 99	362 16193 86
$\mathcal{I}_{\mathrm{d}}\mathcal{TBJB}$	568 22127 102	447 17354 80	407 18458 100	321 14326 78
$\mathcal{I}_{\mathrm{a}}\mathcal{TBJB}$	547 21476 99	487 18873 87	375 17175 99	368 16389 87
$\mathcal{BTJB}$	605 22560 104	443 16486 76	333 15905 101	**246 11617 73**
$\mathcal{O}_{\mathrm{d}}\mathcal{BTJB}$	577 22560 104	**415 16270 75**	336 15915 101	252 11779 73
$\mathcal{O}_{\mathrm{a}}\mathcal{BTJB}$	655 21693 100	544 18005 83	365 15896 99	280 12231 82
$\mathcal{I}_{\mathrm{d}}\mathcal{BTJB}$	623 22560 104	447 16270 75	328 15896 101	250 11921 74
$\mathcal{I}_{\mathrm{a}}\mathcal{BTJB}$	553 21693 100	**415 16270 75**	328 15919 99	245 11624 72
$\mathcal{BTB}$	510 21259 98	483 19957 92	382 17354 98	370 16410 91
$\mathcal{O}_{\mathrm{d}}\mathcal{BTB}$	519 21476 99	487 19957 92	383 16800 98	376 16403 91
$\mathcal{O}_{\mathrm{a}}\mathcal{BTB}$	551 21259 98	522 19957 92	370 17168 98	367 16404 91
$\mathcal{I}_{\mathrm{d}}\mathcal{BTB}$	515 21476 99	485 19957 92	368 16996 98	369 16410 91
$\mathcal{I}_{\mathrm{a}}\mathcal{BTB}$	509 21259 98	480 19957 92	366 16790 98	366 16396 91
$\mathcal{BJBT}$	529 20174 93	570 21693 100	413 16813 92	452 18286 100
$\mathcal{O}_{\mathrm{d}}\mathcal{BJBT}$	491 18656 86	566 21693 100	385 15545 85	450 18281 100
$\mathcal{O}_{\mathrm{a}}\mathcal{BJBT}$	501 19090 88	588 22560 104	399 16105 88	464 18841 103
$\mathcal{I}_{\mathrm{d}}\mathcal{BJBT}$	486 18439 85	563 21693 100	387 15565 85	449 18310 100
$\mathcal{I}_{\mathrm{a}}\mathcal{BJBT}$	504 19306 89	565 21693 100	398 16081 88	452 18269 100

Figure 6.10: Experimental Results on the shapes in figure 6.7. The numerical methods analyzed are listed in columns, and the rows describe the permutations applied to the matrix R. Each cell represents the running time in seconds, the number of Mflops and the number of iterations taken by the solving methods. In bold we have highlighted our best results in terms of Mflops for scalar and block methods. We omit the values obtained by the Jacobi method since they are almost unaffected by matrix permutations and can be found in Figure 6.9.

Mflops of about 3%. For the increased data locality, the running time of `Jac` benefits from matrix reordering, and we have a reduction of up to 23% over the Power iterations.

We now compare the proposed methods versus `PM` applied to the original matrix since this is the common solving method for computing the Pagerank vector. Other comparisons can be obtained from Figure 6.9. As one can expect, the use of `GS` and `RGS` on the original matrix already accounts for a reduction of about 40% in the number of Mflops and of about 48% in running time. These improvements are striking when the system matrix is permuted. The best performance of scalar methods is obtained using the $\mathcal{I}_{\mathrm{a}}\mathcal{TB}$ combination of permutations on `RGS` method. This yields an Mflops reduction of 51% compared to with `PM` and a further reduction of 18% compared to the `GS` both applied to the full matrix. The running time is reduced by 87%.

The common intuition is that the Gauss-Seidel method behaves better on a quasi-lower triangular while the Reverse Gauss-Seidel is faster when applied to quasi-upper triangular matrices. However, in this case the intuition turns out to be misleading. In fact, our Web matrix `RGS` works better on lower block triangular matrices and `GS` works better on quasi-upper triangular matrices. Even better results are obtained using block methods. `LB` applied to $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$ achieves a reduction of 41% in Mflops with respect to `PM`. Adopting this solving method, we explore just the matrix reducibility due to dangling nodes, as in equation (6.7). The best result is obtained for the $\mathcal{O}_{\mathrm{a}}\mathcal{TB}$ permutation when the `LBR` solving method is applied. In this case, we have a reduction of 58% in Mflops and of 90% in the running time. This means that our solving algorithm computes the Pagerank vector in about a tenth of the running time and with less than half the operations of the Power method.

We applied each of the methods in 6.8 also on the shapes in Figure 6.7, obtained with two BFS visits. The results are in Table 6.10 From Figure 6.10 we can see that we have a further gain when the matrix is split into a greater number of blocks. In fact we have a reduction of up to 92% on the running time, and up to 65% in terms of Mfops over the Power method.

The results given in Figures 6.9 and 6.10 do not take into account the effort spent in reordering the matrix. However, the most costly reordering scheme is the BFS visit of the Web graph, which

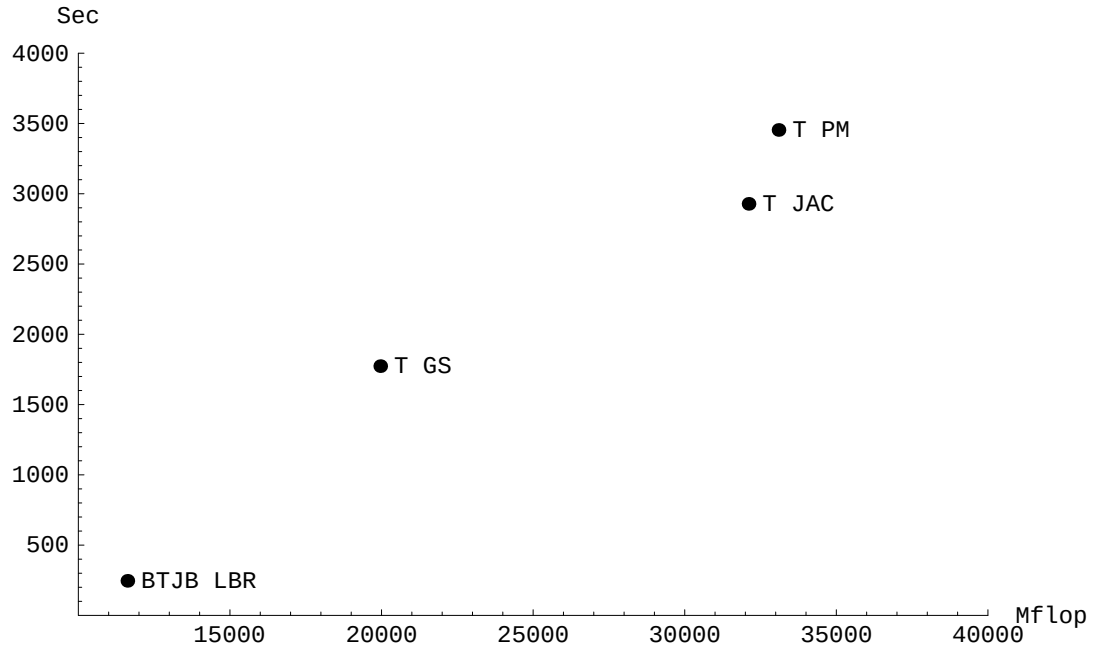


Figure 6.11: A plot of few results of Figure 6.9 and 6.10. On the x-axis the number of Mflops is shown and on the y-axis the running time in seconds is shown. Each point is labeled with the permutation applied and the solving method used. Power, Jacobi and Gauss-Seidel methods applying any permutation are compared with the best result.

can be efficiently implemented into semi-external memory as reported in [35, 150]. The running time spent for carrying out the BFS is comparable to that reported by Broder et al. in [32]. In their study, a Web graph with 100 million nodes was BFS-visited in less than 4 minutes. For our needs, this overhead is largely repaid from the speedup achieved by adopting standard solving methods on permuted matrices. Moreover, in the case of personalized Pagerank the permutations can only be applied once and reused for all personalized vectors $\mathbf{v}$. An intuitive picture of the gain obtained by combining the permutation strategies with the scalar and block solving methods is shown in Figures 6.11 and 6.12.

6.5 Conclusions

The ever-growing size of the Web graph and the ubiquitous Pagerank-based ranking algorithms imply that, in the future, the value and the importance of using fast methods for Web ranking will increase. Moreover, the increasing interest in personalized Pagerank justifies the effort required in "pre-processing" the Web graph matrix, so that the many Pagerank vectors needed may be computed more rapidly. The results have a more general application, since Pagerank has also become a useful paradigm of computation in many Web search algorithms, such as spam detection or trust networks, where the input graphs have different levels of granularity (HostRank) or different link weight assignments (internal, external, etc.).

We have shown how to view the problem of Pagerank computation a dense linear system which reflect the original formulation given by Brin and Page. Then, we have discussed how to handle the density of this matrix by transforming the original problem into one which uses a matrix as sparse as the Web graph itself. As opposed to the experiments carried out by Arasu et al. in [7], we have achieved this result without altering the original model. This result allows us to consider the Pagerank computation as a sparse linear system, an alternative to the eigenpair interpretation. Dealing

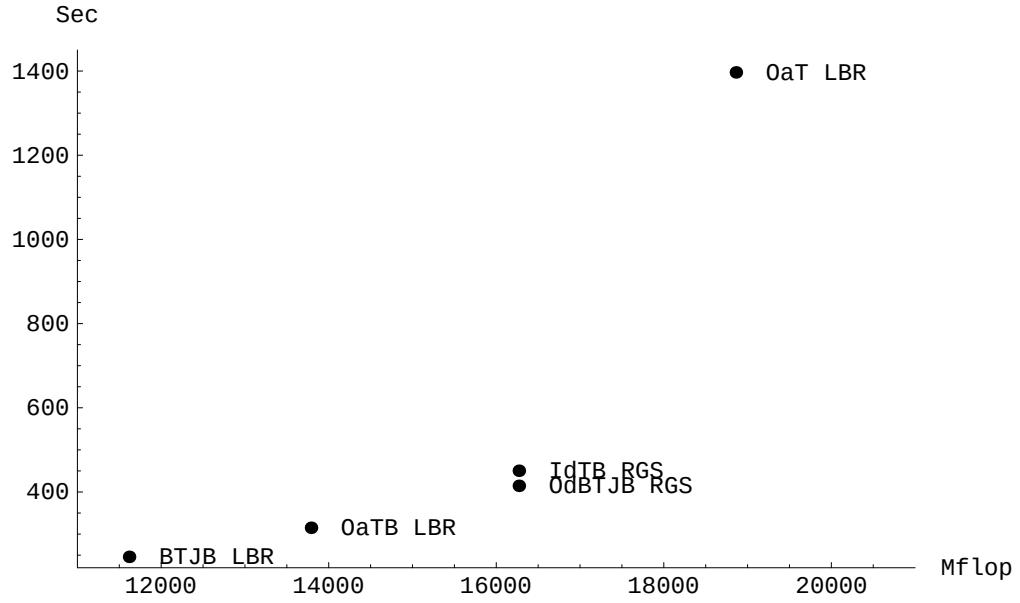


Figure 6.12: A plot of the more interesting results of Figure 6.9 and 6.10. On the x-axis the number of Mflops and on the y-axis the running time in seconds. Each point is labeled with the permutation applied and the solving method used. The best performance of a method, which permutes the node in dangling-nondangling without applying any BFS visit, is plotted with the best block and scalar methods which use one or two BFS visits.

with a sparse linear system opens the way for exploiting the reducibility of the Web matrix, by devising certain permutation strategies to speed up the Pagerank computation. We have demonstrated that, permuting the Web matrix according to a combination of in-degree or out-degree and sorting the pages following the order of the BFS visit, can effectively increase data locality and reduce the running time when used in conjunction with numerical methods, such as lower block solvers. Our best result achieves a 65% reduction in Mflops and a 92% reduction in terms of seconds required, compared to the Power method commonly used to compute the Pagerank. As a result, our solving algorithm requires almost a tenth of the time and considerably less than half of the Mflops used by the Power method. The previous improvement to the Power method is thanks to the studies of Lee et al. in [137], where an 80% reduction in time is achieved on a data set of roughly 400,000 nodes. In view of our results, the approach to speeding up Pagerank computation appears to be much more positive, especially when dealing with personalized Pagerank.

The line of research presented in this Chapter has been extended in [126] where our sparse linear system formula for Pagerank is exploited together with asynchronous iterative solving methods known in literature [80]. The key idea is that the elimination of the synchronizing phases, between two consecutive steps of the solving methods is expected to be advantageous on heterogeneous platforms for a large scale Web graph. A proof of convergence for asynchronous algorithms has also been carried out. The use of our sparse liner systems formulation together with fast strategies of permutation and asynchronous solving methods still remains an interesting open theme of research.

Another important work is the one proposed by Yahoo! in [87, 86]. They studied the problem of Fast Pagerank computation, independently from us. In particular, they evaluated the convergence of iterative stationary and Krylov subspace methods for the Pagerank linear system, including the convergence dependency on teleportation. As in our results, [87] demonstrates that linear system iterations converge faster than the simple power method and are less sensitive to the changes in teleportation. Their work is more focused on the evaluation of methodologies for solving the linear

system in a parallel computational environment, while our work is more focused on the evaluation of permutation strategies for reducing computation. The use of a unifying strategy which exploits permutations over linear system formulation with a asynchronous and parallel solving methods still remains an interesting open question.

Chapter 7

Conclusions and future works

I'll be more enthusiastic about encouraging thinking outside the box when there's evidence of any thinking going on inside it, Terry Pratchett

7.1 Discussion about Clustering and Ranking

The Web has become "the place" for accessing any type of information. There are billions of Web pages and, everyday, new content is produced. Therefore, the use of search engines is becoming a primary Internet activity, and search engines have developed increasingly clever ranking algorithms in order to constantly improve their quality. Nevertheless, there are still many open research areas of tremendous interest where the quality of search results can be improved. During this Ph.d, we have shown that clustering and ranking are WebIR tools linked by a mutual reinforcement relationship and that their joint use might further boost the quality of search results. They have been largely used separately, but their relationship has never been investigated deeply before in literature, as we have done in this Thesis.

On the one hand, a good clustering algorithm can provide a valuable basis for improving ranking. For instance, when a flat list of about ten search results provided by a search engine is not enough to provide a satisfying search experience, the clustering methodologies may emphasize hidden knowledge meanings and can organize the results in a hierarchy of folders annotated with meaningful labels. The hierarchy and the labels are not statically predefined, but dynamically derived by analyzing the search results themselves. We believe that clustering is essential in transforming the search experience into a *"personalized navigational search experience"*. The traditional paradigm for personalizing search results is to observe users while surfing the Web and their habits, and infer from those the profile of the user. Then, when the user submits the next search, the results are biased by the profile. It has been noted [236] that the limitation of this model for personalization is the need to maintain updated profiles, a problem which appears at least as difficult as the problem of ranking the Web itself. We believe that, in order to solve the problem, the key idea is to analyze the search results, show users the variety of themes therein, and let them explore their interests at that moment, which only they know. In other words, *the best personalization is carried out on-the-fly by people themselves.* Besides the aforementioned benefit, there are many other benefits of using clustering to improve ranking. For instance, we have shown that clustering may be used to infer implicit relationships between pages even when they are not explicitly interconnected or there are just few interconnections

On the other hand, a good ranking strategy can provide a valuable basis of information for clustering. For instance, in the context of web snippet clustering, the organization of snippets into a hierarchy of labeled folders is influenced by the quality of the analyzed snippets themselves. This analysis happens on behalf of the users, and it is not restricted to the first ten results but it is

applied to a larger set of search results inherently biased by the ranking criteria adopted by the search engines. A similar situation happens for ranking different types of Web objects, such as news articles, where the order induced by ranking metrics can influence their clustering.

In the next Sections we briefly highlight where the combined use of clustering and ranking has been proved to improve the quality of search results. We will review the main algorithmic, numeric, and software contributions of this Ph.D., and we will propose some interesting open lines of research which have arisen from our study.

7.2 A synthesis of our contributions

Freshness is an important factor for assessing the quality of search results. In fact, it has been noted [156] that, over the course of a week, the Web changes very rapidly, with more than 25% of links being changed and a 5% increase in "new content" being created. Therefore, search engines need to update link-based ranking metrics on a regular basis, and a week-old ranking may not sufficiently reflect the current importance of the pages. One contribution of this Ph.D. has been the introduction of a methodology for fast computation of Pagerank. By using numerical techniques, we have demonstrated in [51, 54, 52] that Pagerank may be transformed into a linear system of equations, where the coefficient matrix is as sparse as the Web graph itself. Then, we evaluated the effect of many different permutations on the matrix and many different solving methods. We tested our methodology on a crawled web graph of approximately 24 million nodes and more than 100 million links. Our best result is a 65% reduction in Mflops and a 92% reduction in time in relation to the Power method, which is taken as a reference method in computing the Pagerank vector. The results have a more general application, since Pagerank has also become a useful paradigm of computation in many Web search algorithms, such as spam detection or trust networks, where the input graphs have different levels of granularity (HostRank) or different link weight assignments (internal, external, etc.). For each algorithm, the critical computation is a Pagerank-like vector of interest. Thus, methods to accelerate and parallelize these kinds of algorithms are very important. Moreover, YAHOO! has independently developed a similar line of fast ranking research [87, 86] at the same time as our publications were published.

The integration of different types of fresh contents is another important aspect for assessing the quality of the search results. In fact, it has been noted [231] that, on an average day, about 94 million American adults use the Internet. Among them about 46% are trying to get news information. Surprisingly enough, this Internet activity is the third most common, just after emailing and traditional Web searching. In response to this growing trend, Ask Jeeves, Google, Yahoo, MSN and other major Web companies have established commercial search engines for indexing news feeds and integrated them with Web searching. However, despite significant commercial interest, no academic research has as yet focused on the question of ranking a stream of news articles and a set of news sources. It should be noted that news articles are hard to rank with traditional hyper-link based metrics, since it is possible that there is not enough time to have links pointing to the articles themselves. We addressed the problem in [53, 94] by proposing a framework which models the generation of a stream of news articles, the news article clustering by topic process, the evolution of a news story over time, the ranking of news articles, and the ranking of news sources. Clustering and ranking are used to discover similarities between articles and to impose an order to both the pieces of information and the producers of the information. A large part of these results are used in COMETOMYHEAD, a publicly-available news engine (http://newsengine.di.unipi.it) also developed for this purpose. It is not by chance that a patent [227] has been independently filed by GOOGLE on this subject, at the same time as our publications were published.

Another aspect for improving the quality of the search results is their personalization according to the users' activities and their organization into a hierarchy of labeled folders. We have presented different algorithms in [73, 74, 75, 76, 95] which are exploited by SnakeT, a publicly-available meta-search engine for Web snippet clustering (http://snaket.com). SnakeT achieves efficiency and efficacy performance close to the industrial state-of-the-art, namely VIVISIMO [206]

and MICROSOFT [216] but our proposal improves those solutions. The key idea is that a user issues a query using SnakeT, gets back a labeled folder hierarchy built automatically on the top of hundreds of search results. Then, (s)he selects a set of folders whose labels (themes) best fit his/her query needs. Given this selection, SnakeT personalizes the original ranked list by *filtering out* those results which do not belong to the selected folders on-the-fly. In this way, the personalization is then carried out by the people themselves. We point out that SnakeT does not use predefined ontology and the topics and their organization in a hierarchy are inferred on-the-fly analyzing the list of answers given by Web search engines. This approach of course does not require an explicit `login` by the user, a pre-compilation of a constrained user profile, a tracking of the user's past search behaviour, or a modification of the underlying search engines. We have shown how to plug SnakeT on top of any (*un*personalized) search engine in order to obtain a form of personalization that is fully-adaptive, privacy preserving, and scalable to an unbounded number of user needs. We point out that SnakeT is another relevant example of the benefits of using clustering and ranking. In fact the better the ranking is, the more relevant the results (and thus their snippets) are from which SnakeT distills the hierarchy of labeled folders. Vice versa, the more intelligible the labeled folders produced by SnakeT are, the more effective the personalization of the ranked list of query results is.

7.3 Future lines of research

The personalized navigational search paradigm should be explored further by integrating text hierarchical clustering with an analysis of the query and click logs. As discussed above, we believe that the best personalization is carried out on-the-fly by people themselves, and we do not believe that it is possible to maintain updated profiles for each single user by analyzing the query and click logs. Nevertheless, query and click logs can be useful in detecting similarities amongst Web pages, which cannot be inferred by just analyzing the text. Other types of non-textual similarities could be inferred by exploiting the hyper-textual links among the pages. Unfortunately, query and click logs and Web graphs are subject to data privacy and therefore they are not easily available for academic research. Another interesting idea to explore is the use of methodologies for Web snippet Clustering upon a small random sample instead of the full set of search results. This idea appears intriguing after the publication of [5], where a technique for efficiently sampling Web search engine query results has been discussed. In fact, it has been shown that efficient sampling could mitigate the ranking bias of query results. Consequently, a larger variety of topics could be covered, without having a negative impact on the computation time.

The ranking methodologies for news articles and news sources can be extended to other types of fresh information. For instance, there are many industrial proposals for blog searching, such as BLOGLINES, MYYAHOO and BLOGGER. However, despite significant commercial interest, no academic research has as yet focused on the question of ranking a stream of blogs and the individual bloggers themselves. In this context, the ranking metrics presented in this Ph.D., can be integrated with more traditional link-based metrics, as well as with other specific information to this domain such as the blogs and subscribers bipartite graph. Another interesting line of research could be the application of the methodologies we have proposed to rank a stream of academic publications, the authors, the publishers and the journals. In this field there are already some industrial proposals, such as GOOGLE SCHOLAR, as well as academic proposals, such as CITESEER. In general, our proposals can be extended to rank any data domain which is annotated with temporal information, such as email, usenet postings, desktop search, etc.

Fast ranking methodologies are needed due to the size of the Web and the way in which it changes rapidly. In the last three years we have counted at least a dozen different proposals for speeding-up Pagerank computation. We think that our proposal provides an improvement compared with the standard methods for computing Pagerank. We considered Pagerank computation as the solution for a linear system and we suggested to permute the Web matrix to reduce computation time and to exploit the characteristics of the different linear solvers. A future line of research

could investigate a unifying strategy that exploits permutations over a linear system formulation in a parallel computational environment. This approach could be integrated with iterative solving methods that do not require an explicit synchronization at the end of each iteration.

The results of this Doctoral Thesis could form a starting point to further boost the quality of modern search engines by exploiting the mutual benefits between clustering and ranking.

Sleep Sleep tonight
And may your dreams
Be realized
MLK, U2

Bibliography

[1] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu, and J. Park. Fast algorithms for projected clustering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 61–72, Philadelphia, U.S., 1999.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. Yu. Clustream: A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 81–92, Berlin, Germany, 2003.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of ACM SIGMOD international conference on Management of data*, pages 94–105, Seattle, Washington, U.S., 1998.

[4] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., U.S, 1993.

[5] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search engine results. In *Proceedings of 14th International World Wide Web Conference*, pages 245–256, Chiba, Japan, 2005.

[6] A. Andersson, N. J. Larsson, and K. Swanson. Suffix trees on words. *Algorithmica*, 23(3):246–260, 1999.

[7] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. Pagerank computation and the structure of the Web: Experiments and algorithms. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawai, U.S., 2002.

[8] M. Atallah, R. Gwadera, and W. Szpankowski. Detection of significant sets of episodes in event sequences. In *Proceedings of the International Conference on Data Mining*, pages 3–10, Brighton, UK, 2004.

[9] G. Attardi, A. Gulli, and F. Sebastiani. Theseus: categorization by context. In *Proceedings of 8th International World Wide Web Conference*, pages 136–137, Toronto, CA, 1999.

[10] A. Rungsawang B. Manaskasemsak. Parallel Pagerank computation on a gigabit pc cluster. In *Proceedings of Advanced Information Networking and Applications*, pages 273–277, Fukuoka, Japan, 2004.

[11] L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: the google cluster architecture. In *IEEE Micro*, pages 22–28, 2003, March.

[12] Z. BarYossef and M. Gurevich. Random sampling from a search engines index. In *Proceedings of 14th International World Wide Web Conference*, Edinburgh, U.K., 2006, to appear.

[13] K. Berberich, M. Vazirgiannis, and G. Weikum. T-rank: Time-aware authority ranking. In *Proceedings of Third International Workshop on Algorithms and Models for the Web-Graph, WAW*, pages 131–142, Rome, Italy, 2004.

[14] K. Bharat. Searchpad: Explicit capture of search context to support Web search. In *In Proceedings of the 10th International World Wide Web Conference*, pages 493–501, Amsterdam, The Netherlands, 2000.

[15] K. Bharat and A. Z. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of 7th International World Wide Web Conference*, pages 379 – 388, Brisbane, Australia, 1998.

[16] K. Bharat, A. Z. Broder, J. Dean, and M. R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *IEEE Data Engineering Bulletin*, 23(4):21–26, 2000.

[17] K. Bharat, A. Z. Broder, M. R. Henzinger, P. Kumar, and S. Venkatasubramanian. The connectivity server: Fast access to linkage information on the Web. In *Proceedings of 7th International World Wide Web Conference*, pages 469–477, Brisbane, Australia, 1998.

[18] K. Bharat, B.W. Chang, M. Henzinger, and M. Ruhl. Who links to whom: Mining linkage between Web sites. In *Proceedings of the IEEE International Conference on Data Mining*, pages 51–58, San Jose, California, U.S., 2001.

[19] K. Bharat and M. Henzinger. Improved algorithms for topic distillation in a hyper-linked environment. In *Research and Development in Information Retrieval*, pages 104–111, 1998.

[20] M. Bianchini, M. Gori, and F. Scarselli. Inside Pagerank. In *Journal of ACM Transaction on Internet*, volume 5, 2005.

[21] J. Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian Mixture and Hidden Markov Models. Technical report, University of Berkeley, 1997.

[22] P. Boldi. Totalrank: ranking without damping. In *Special interest tracks and posters of the 14th World Wide Web International Conference*, pages 898–899, Chiba, Japan, 2005.

[23] P. Boldi, M. Santini, and S. Vigna. Pagerank as a function of the damping factor. In *Proceedings of the 14th World Wide Web International Conference*, pages 557–566, New York, NY, USA, 2005.

[24] P. Boldi and S. Vigna. WebGraph framework i: Compression techniques. In *Proceedings of the 13th International World Wide Web Conference*, pages 595–602, New York, U.S., 2004.

[25] A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas. Link analysis ranking: algorithms, theory, and experiments. *ACM Transaction on Internet Technology*, 5(1):231–297, 2005.

[26] J.T. Bradley, D.V. de Jager, W.J. Knottenbelt, and Aleksandar Trifunovic. Hypergraph partitioning for faster parallel Pagerank computation. In *Proceedings of EPEW'05, Proceedings of the 2nd European Performance Evaluation Workshop*, Versailles, France, 2005.

[27] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of 7th International World Wide Web Conference*, pages 107–117, Brisbane, Australia, 1998.

[28] A. Z. Broder. Min-wise independent permutations. In *Journal of Computer and System Sciences*, volume 60, pages 630–659, 2000.

[29] A. Z. Broder. A taxonomy of Web search. In *SIGIR Forum 36(2)*, pages 3–10, 2002.

[30] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Y. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management*, pages 426–434, New Orleans, Louisiana, U.S., 2003.

[31] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proceedings of 6th International World Wide Web Conference*, pages 1157–1166, 1997.

[32] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener:. Graph structure in the Web. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, The Netherlands, 2000.

[33] A. Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient Pagerank approximation via graph aggregation. In *Proceedings of the 13th International World Wide Web Conference*, pages 484–485, New York, U.S., 2004.

[34] A. Z. Broder, M. Najork, and J. L. Wiener. Efficient url caching for world wide Web crawling. In *Proceedings of the 12th International World Wide Web Conference*, pages 679–689, Budapest, HU, 2003.

[35] A. L. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. Westbrook. On external memory graph traversal. In *Proceedings of Symposium on Discrete Algorithms*, pages 859–860, San Francisco, California, U.S., 2000.

[36] C. Carpineto and G. Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, 2004.

[37] S. Chakrabarti. *Mining the Web: Analysis of Hypertext and Semi Structured Data Mining the Web: Analysis of Hypertext and Semi Structured Data*. Morgan Kaufmann, 2003.

[38] P. Chase, R. DAmore, N. Gershon, R. Holland, R. Hyland, M. Maybury I. Mani, A. Merlino, and J. Rayson. Semantic visualization. In *ACL-COLING Workshop on Content Visualization and Intermedia Representation*, volume Montreal, Quebec, Canada, 1998.

[39] H. Chen and S. T. Dumais. Bringing order to the Web: automatically categorizing search results. In *Proceedings of SIGCHI00*, pages 145–152, The Hague, The Netherlands, 2000.

[40] Y. Chen, Q. Gan, and T. Suel. I/o-efficient techniques for computing Pagerank. In *Proceedings of ACM Conference on Information and Knowledge Engineering*, pages 549–557, McLean, Virginia, U.S., 2002.

[41] Y. Chen, Q. Gan, and T. Suel. Local methods for estimating Pagerank values. In *Proceedings of International Conference on Information and Knowledge Management*, pages 381–389, Washington, D.C., U.S., 2004.

[42] S. Chien, C. Dwork, S. Kumar, and D. Sivakumar. Towards exploiting link evolution. In *Proceedings of Workshop on Algorithms for the WebGraph*, Vancouver, 2002.

[43] P. A. Chirita, W. Nejdl, R. Paiu, and C. Kohlschuetter. Using odp metadata to personalize search. In *International Conference on Research and Development in Information Retrieval SIGIR05*, Salvador, Brazil, 2005.

[44] P. A. Chirita, D. Olmedilla, and W. Nejdl. PROS: A personalized ranking platform for Web search. In *Proceedings of 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 34–43, Eindhoven, Netherlands, 2004.

[45] J. Cho and S. Roy. Impact of Web search engines on page popularity. In *Proceedings of the 13th International World Wide Web Conference*, pages 20–29, New York, U.S., 2004.

[46] S. Chung and D. McLeod. Dynamic topic mining from news stream data. In *Proceedings of Cooperative Information Systems International Conference*, pages 653–670, Catania, Sicily, Italy, 2003.

[47] J. M. Cigarrán, A. Peñas, J. Gonzalo, and F. Verdejo. Evaluating hierarchical clustering of search results. In *SPIRE String Processing and Information Retrieval, 12th International Conference*, pages 49–54, Buenos Aires, Argentina, 2005.

[48] R. Cilibrasi and P. M. B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

[49] CNN.com. Better search results than Google? Next-generation sites help narrow internet searches. Associated Press, January 2004.

[50] D. Cohn and H. Chang. Learning to probabilistically identify authoritative documents. In *Proceedings of the 17th International Conference on Machine Learning*, pages 167–174. Stanford University, 2000.

[51] G. M. Del Corso, A. Gulli, and F. Romani. Fast Pagerank computation via a sparse linear system (extended abstract). In *Proceedings of Algorithms and Models for the Web-Graph: Third International Workshop, WAW*, pages 118–130, Rome, Italy, 2004.

[52] G. M. Del Corso, A. Gulli, and F. Romani. Fast Pagerank computation via a sparse linear system. *Journal of Internet Mathematics*, 2(3), 2005.

[53] G. M. Del Corso, A. Gulli, and F. Romani. Ranking a stream of news. In *Proceedings of 14th International World Wide Web Conference*, pages 97–106, Chiba, Japan, 2005.

[54] G. M. Del Corso, A. Gulli, and F. Romani. Comparison of Krylov subspace methods on the Pagerank problem. *Journal of Computational and Applied Mathematics*, 2005, to appear.

[55] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings 24th National Conference of ACM*, pages 157–172, 1969.

[56] D. Cutting, D. Karger, J. Pedersen, and J. W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *ACM/SIGIR Conference*, pages 318–329, Copenhagen, Denmark, 1992.

[57] S. Blair-Goldensohn D. R. Radev, Z. Zhang, and R. S. Raghavan. Interactive, domain-independent identification and summarization of topically related news articles. In *Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries*, pages 225–238, Darmstadt, Germany, 2001.

[58] S. Blair-Goldensohn D. R. Radev, Z. Zhang, and R. S. Raghavan. Newsinessence: A system for domain-independent, real-time news clustering and multi-document summarization. In *Proceedings of the Human Language Technology Conference*, San Diego, California, U.S., 2001.

[59] D. de Castro Reis, P. Golgher, A. da Silva, and A. Laender. Automatic Web news extraction using tree edit distance. In *Proceedings of the 13th International World Wide Web Conference*, pages 502–511, New York, U.S., 2004.

[60] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. In *Journal Royal Statistic Society, Series B*, volume 39, pages 1–38, 1977.

[61] Chris H. Q. Ding, Xiaofeng He, and Hongyuan Zha. A spectral method to separate disconnected and nearly-disconnected Web graph components. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 275–280, San Francisco, CA, U.S., 2001.

[62] B. Dom. An information-theoretic external cluster-validity measure. Technical Report RJ 10219, IBM Research Report, 2001.

[63] C. C. Douglas, J. Hu, M. Iskandarani, M. Kowarschik, U. Rüde, and C. Weiss. Maximizing cache memory usage for multigrid algorithms. In *Multiphase Flows and Transport in Porous Media: State of the Art*, pages 124–137. Springer, Berlin, 2000.

[64] D. Dreilinger and A. E. Howe. Experiences with selecting search engines using metasearch. *Journal of ACM Transaction on Information Systems*, 15(3):195–222, 1997.

[65] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 291–299, Baltimore, Maryland, U.S., 1999.

[66] D. M. Dunlavy, J. P. Conroy, and D. P. O'Leary. Qcs: A tool for querying, clustering, and summarizing documents. In *Proceedings of the Human Language Technology conference-NAACL*, pages 11–12, San Diego, California, U.S., 2003.

[67] N. Eiron, K. S. McCurley, and J. Tomlin. Ranking the Web frontier. In *Proceedings of the 13th International World Wide Web Conference*, pages 309–318, New York, U.S., 2004.

[68] 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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, U.S., 1996.

[69] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *Proceeedings of Symposium on Principles of Database Systems*, pages 47–58, Paris, France, 2004.

[70] D. Fallows, L. Rainie, and G. Mudd. The popularity and importance of search engines. Technical report, The Pew Internet & American Life Project, 2004.

[71] D. Fasulo. An analysis of recent work on clustering algorithms. Technical Report 01-03-02, Department of Computer Science and Engineering, University of Washington, 1999.

[72] P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino. Boosting textual compression in optimal linear time. In *Journal of the ACM*, volume 52, pages 688–713, 2005.

[73] P. Ferragina and A. Gulli. The anatomy of a hierarchical clustering engine for web-page, news and book snippets. In *Proceedings of the 4th IEEE International Conference on Data Mining*, pages 395–398, Brighton, UK, 2004.

[74] P. Ferragina and A. Gulli. The anatomy of SnakeT: A hierarchical clustering engine for web-page snippets. In *Proceedings of 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 506–508, Pisa, Italy, 2004.

[75] P. Ferragina and A. Gulli. Experimenting SnakeT: A hierarchical clustering engine for web-page snippets. In *Proceedings of 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 543–545, Pisa, Italy, 2004.

[76] P. Ferragina and A. Gulli. A personalized search engine based on web snippet hierarchical clustering. In *Proceedings of 14th International World Wide Web Conference*, pages 801–810, Chiba, Japan, 2005.

[77] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of Web pages. In *Proceedings of the 12th International World Wide Web Conference*, pages 669–678, Budapest, HU, 2003.

[78] W. B. Frakes and R. Baeza-Yates. *Clustering Algorithms*, chapter 16. Prentice Hall, Englewood Clifs, NJ, 1992.

[79] J. Freyne and B. Smyth. Communities, collaboration and cooperation in personalized web search. In *In Proceedings of Third Workshop on Intelligent Techniques for Web Personalization.*, Edinburgh, Scotland, 2005.

[80] A. Frommer and D.B. Szyld. On asynchronous iterations. In *Journal Compututation and Applied Mathematics*, volume 123, page 201216, 2000.

[81] N. Fuhr. Probabilistic models in information retrieval. *The Computer Journal*, 35(3):243–255, 1992.

[82] B. Fung, K. Wang, and M. Ester. Large hierarchical document clustering using frequent itemsets. In *Proceedings of SIAM International Conference on Data Mining*, San Francisco, U.S., 2003.

[83] E. Gabrilovich, S. Dumais, and E. Horvitz. Newsjunkie: Providing personalized newsfeeds via analysis of information novelty. In *Proceedings of the 13th International World Wide Web Conference*, pages 482–490, New York, U.S., 2004.

[84] E. Di Giacomo, W. Didimo, L. Grilli, and G. Liotta. A topology-driven approach to the design of Web meta-search clustering engines. In *Proceedings of 31st Annual Conference on Current Trends in Theory and Practice of Informatics*, pages 106–116, Liptovsk Jn, Slovakia, 2005.

[85] F. Giannotti, M. Nanni, and D. Pedreschi. Webcat: Automatic categorization of web search results. In *Proceedings of 11th Italian Symposium on Advanced Database Systems*, pages 507–518, Cetraro, Italy, 2003.

[86] D. Gleich and L. Zhukov. Scalable computing for power law graphs: Experience with parallel Pagerank. Technical report, Yahoo! Research Labs, 2005.

[87] D. Gleich, L. Zhukov, and P. Berkhin. Fast parallel Pagerank: A linear system approach. Technical Report YRL-2004-038, Yahoo! Research Labs, 2004.

[88] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, 1996. Third Edition.

[89] J. Grabmeier and A. Rudolph. Techniques of cluster algorithms in data mining. In *Journal of Data Mining and Knowledge Discovery*, volume 6(4), pages 303–360, 2002.

[90] M. Grobelnik and D. Mladenic. Visualization of news articles. In *Informatica, Journal of Computing and Informatics*, volume 29, 2004.

[91] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. In *Journal of IEEE Transactions on Knowledge and Data Engineering*, pages 515–528, 2003.

[92] X. Gui-Rong, Y. Qiang, H.J. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *International Conference on Research and Development in Information Retrieval SIGIR05*, Salvador, Brazil, 2005.

[93] J. Guillaume and M. Latapy. The Web graph: an overview. In *In 4mes rencontres francophones sur les Aspects Algorithmiques des Telecommunications (AlgoTel), INRIA*, 2002.

[94] A. Gulli. The anatomy of a news search engine. In *Proceedings of 14th International World Wide Web Conference*, pages 880–881, Chiba, Japan, 2005.

[95] A. Gulli and A. Signorini. Building an open source meta search engine. In *Proceedings of 14th International World Wide Web Conference*, pages 1004–1005, Chiba, Japan, 2005.

[96] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *Proceedings of 14th International World Wide Web Conference*, pages 902–903, Chiba, Japan, 2005.

[97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[98] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating Web spam with trustrank. In *In Proceedings of the 30th International Conference on Very Large Data Bases*, page 576587, Toronto, Canada, 2004.

[99] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. In *Journal of Intelligent Information Systems*, pages 107–145, 2001.

[100] T. Haveliwala. Efficient computation of PageRank. Technical report, Stanford University, 1999.

[101] T. Haveliwala. Topic-sensitive Pagerank. In *Proceedings of 11th International World Wide Web Conference*, pages 517–526, Honululu, U.S., 2002.

[102] T. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the Web. In *Proceedings of 11th International World Wide Web Conference*, pages 432–442, Honululu, U.S., 2002.

[103] T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing Pagerank. Technical report, Stanford University, 2003.

[104] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the Web. In *Proceedings of International Workshop on the Web and Databases*, 2000.

[105] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/Gather on retrieval results. In *Proceedings of SIGIR96*, pages 76–84, Zurich, Ch, 1996.

[106] M. Henzinger, B.-W. Chang, B. Milch, and S. Brin. Query-free news search. In *Proceedings of 12th International World Wide Web Conference*, pages 1–10, Budapest, HU, 2003.

[107] T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of Uncertainty in Artificial Intelligence, UAI'99*, pages 289–296, Stockholm, 1999.

[108] T. Hofmann. Learning probabilistic models of the Web. In *Proceedings of the 23rd ACM SIGIR*, pages 369–371, 2000.

[109] B. Hunt and M. Moran. *Search Engine Marketing, Inc.: Driving Search Traffic to Your Company's Web Site*. IBM Press, 2005.

[110] Sander J., Ester M., Kriegel H.-P., and Xu X. Density-based clustering in spatial databases, the algorithm GDBSCAN and its applications. In *Data Mining and Knowledge Discovery*. Kluwer Academic Publishers, 1998.

[111] G. Jeh and J. Widom. Scaling personalized Web search. In *Proceedings of the 12th International World Wide Web Conference*, pages 271–279, Budapest, HU, 2003.

[112] Z. Jiang, A. Joshi, R. Krishnapuram, and L. Yi. *Managing Business with Electronic Commerce 02*, chapter 4, Retriever: Improving Web Search Engine Results Using Clustering. A. Gangopadhyay, 2002.

[113] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, Edmonton, Alberta, Canada, 2002.

[114] K. Sparck Jones, S. Walker, and S. A. Robertson. Probabilistic model of information retrieval: Development and status. Technical Report TR-446, Cambridge University Computer Laboratory., 1998.

[115] S. Kamvar and M. T. Schlosserand H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International World Wide Web Conference*, pages 640–651, Budapest, Hungary, 2003.

[116] S. Kamvar and T. Haveliwala. The condition number of the Pagerank problem. Technical report, Stanford University, 2003.

[117] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation methods for accelerating Pagerank computations. In *Proceedings of the 12th International World Wide Web Conference*, pages 261–270, Budapest, Hungary, 2003.

[118] S. D. Kamvar, T. H. Haveliwala, and G. H. Golub. Adaptive methods for the computation of Pagerank. In *Proceedings of the International Conference on the Numerical Solution of Markov Chains*, University of Illinois at Urbana-Champaign, 2003.

[119] S. D. Kamvar, T. H. Haveliwala, C. Manning, and G. H. Golub. Exploiting the block structure of the Web for computing Pagerank. Technical report, Stanford University, 2003.

[120] S. D. Kamvar, D. Klein, and C. D. Manning. Spectral learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[121] J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. In *Proceedings 30th International Colloquium on Automata, Languages and Programming (ICALP '03)*, Eindhoven, The Netherlands, 2003.

[122] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. 1960.

[123] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 430:81–93, 1938.

[124] J. Kleinberg. Authoritative sources in a hyper-linked environment. In *Proceedings of 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, San Francisco, U.S., 1998.

[125] J. M. Kleinberg. Authoritative sources in a hyper-linked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[126] G. Kollias, E. Gallopoulos, and D. B. Szyld. Asynchronous iterative computations with Web information retrieval structures: The Pagerank case. unpublished, 2005.

[127] A. Kritikopoulos and M. Sideri. The compass filter: Search engine result personalization using Web communities. In *In Proceedings of Intelligent Techniques for Web Personalization.*, Acapulco, Mexico, 2003.

[128] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. The Web as a graph. In *In Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems,*, pages 1–10, Dallas, Texas, U.S., 2000.

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

[130] A. N. Langville and C. D. Meyer. Deeper inside Pagerank. *Journal of Internet Mathematics*, 2004.

[131] A. N. Langville and C. D. Meyer. A survey of eigenvector methods of Web information retrieva. *SIAM Review*, 2004.

[132] A. N. Langville and C. D. Meyer. Updating Pagerank with iterative aggregation. In *Proceedings of the 13th International World Wide Web Conference*, pages 392–393, New York, U.S., 2004.

[133] A. N. Langville and C. D. Meyer. Updating the stationary vector of an irreducible Markov chain with an eye on Google's Pagerank. In *SIMAX, To appear*, 2005.

[134] S. Lawrence and C. L. Giles. Accessibility of information on the Web. *Nature*, 400:107–109, 1999.

[135] D. J. Lawrie. *Language Models for Hierarchical Summarization*. PhD thesis, Amherst, 2003.

[136] D. J. Lawrie and W. B. Croft. Generating hierachical summaries for Web searches. In *Proceedings of SIGIR03*, pages 457–458, Toronto, Canada, 2003.

[137] C. P. Lee, G. H. Golub, and S. A. Zenios. A fast two-stage algorithm for computing Pagerank. Technical report, Stanford University, 2003.

[138] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. In *Proceedings of the 9th International World Wide Web Conference*, pages 387–401, Amsterdam, The Netherlands, 2000.

[139] D. D. Lewis. Reuters-21578 text categorization test collection distribution 1.0, 1999.

[140] Z. Li, B. Wang, M. Li, and W. Ma. A probabilistic model for retrospective news event detection. In *International Conference on Research and Development in Information Retrieval SIGIR05*, Salvador, Brazil, 2005.

[141] X. Long and T. Suel. Three-level caching for efficient query processing in large Web search engines. In *Proceedings of the 14th international conference on World Wide Web*, pages 257–266, Chiba, Japan, 2005.

[142] Y. Maarek and F. Smadja. Full text indexing based on lexical relations: An application: Software libraries. In *Proceedings of ACM SIGIR*, pages 198–206, Cambridge, Massachusetts, U.S., 1989.

[143] Y. S. Maarek, R. Fagin, I. Z. Ben-Shaul, and D. Pelleg. Ephemeral document clustering for Web applications. Technical Report RJ 10186, IBM Research, 2000.

[144] U. Manber and E. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal Computing*, 22(5):935–948, 1993.

[145] B. Mandhani, S. Joshi, and K. Kummamuru. A matrix density based algorithm to hierarchically co-cluster documents and words. In *Proceedings of the 12th International World Wide Web Conference*, pages 511–518, Budapest, HU, 2003.

[146] H. Mannilla, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. In *Proceedings of the Data Mining and Knowledge Discovery*, pages 259–289, Newport Beach, California, U.S., 1997.

[147] G. Manzini. Two space saving tricks for linear time LCP array computation. In *SWAT, Scandinavian Workshop on Algorithm Theory*, pages 372–383, Humlebaek, Denmark, 2004.

[148] G. Manzini and P. Ferragina. Engineering a lightweight suffix array construction algorithm. *Algorithmica*, 40(1):33–50, 2004.

[149] E. M. McCreight. A space-economical suffix tree construction algorithm. In *Journal of the ACM*, volume 23, pages 262–72, 1976.

[150] K. Mehlhorn and U. Meyer. External-memory breadthfirst search with sublinear I/O. In *Proceedings of European Symposium on Algorithms*, pages 723–735, Rome, Italy, 2002.

[151] M. Meila. Comparing clusterings. Technical Report 418, University of Washington, 2002.

[152] M. Morita and Y. Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *In Proceedings of 17th Annual International SIGIR*, pages 272–281, Dublin, Ireland, 1994.

[153] J. Mostafa. Seeking better Web searches. *Scientific American*, February 2005.

[154] S. Muthukrishnan. Data streams: algorithms and applications. In *Proceedings of the ACM-SIAM 14th symposium on Discrete algorithms*, page 413, Baltimore, Maryland, U.S., 2003.

[155] A. Y. Ng, A. X. Zheng, and M. I. Jordan. Stable algorithms for link analysis. In *Proceedings 24th Annual Intl. ACM SIGIR Conference*, pages 258–266, New Orleans, Louisiana, U.S., 2001.

[156] A. Ntoulas, J. Cho, and C. Olston. What's new on the Web? the evolution of the Web from a search engine perspective. In *Proceedings of the 13th International World Wide Web Conference*, pages 1–12, New York, U.S., 2004.

[157] S. Osinski and D. Weiss. Conceptual clustering using lingo algorithm: Evaluation on open directory project data. In *Proceedings of New Trends Intelligent Information Processing and Web Mining*, pages 369–377, Zakopane, Poland, 2004.

[158] H. Oyanagi, K. Kubota, and A. Nakase. Application of matrix clustering to Web log analysis and access prediction. In *WEBKDD*, San Francisco, California, U.S., 2001.

[159] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[160] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168, Seattle, Washington, U.S., 1998.

[161] W. Pratt, M. A. Hearst, and L. M. Fagan. A knowledge-based approach to organizing retrieved documents. In *Proceedings of AAAI99, 16th National Conference on Artificial Intelligence*, pages 80–85, Orlando, U.S., 1999.

[162] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, Chicago, U.S., 2005.

[163] D. Rafiei and A. Mendelzon. What is this page known for? computing Web page reputations. In *Proceedings of the 9th International World Wide Web Conference*, pages 823–835, Amsterdam, The Netherlands., 2000.

[164] K. Randall, R. Stata, R. Wickremesinghe, and J. Wiener. The link database: Fast access to graphs of the Web. Technical report, Compaq Systems Research Center, 2001.

[165] T.A. Runkler and J.C. Bezdek. Web mining with relational clustering. *Elsevier*, 2001.

[166] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill, 1983.

[167] E. Selberg. *Towards Comprehensive Web Search*. PhD thesis, University of Washington, 1999.

[168] X. Shen and C. X. Zhai. Exploiting query history for document ranking in interactive information retrieval. In *In Proceedings of the 26th annual international ACM SIGIR (Poster)*, pages 377–378, Toronto, Canada, 2003.

[169] V Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed Web crawler. In *Proceedings of the 18th International Conference on Data Engineering*, pages 357–368, San Jose, CA, U.S., 2002.

[170] C. Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:72–101, 1904.

[171] W. S. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1995.

[172] K. Sugiyama, K. Hatano, and M. Yoshikawa. Adaptive Web search based on user profile constructed without any effort from user. In *In Proceedings of 13th international conference on World Wide Web*, pages 675–684, New York, NY, U.S., 2004.

[173] J. Sun, H. Zeng, H.Liu, Y. Lu, and Z.Chen. Cubesvd: A novel approach to personalized Web search. In *Proceedings of 14th International World Wide Web Conference*, pages 382–390, Chiba, Japan, 2005.

[174] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger. The perfect search engine is not enough: A study of orienteering behavior in directed search. In *In Proceedings of Computer-Human Interaction Conference*, pages 415–422, Vienna, Austria, 2004.

[175] J. Teevan, S. T. Dumais, and E. Horvitz. Beyond the commons: Investigating the value of personalizing Web search. In *Workshop on New Technologies for Personalized Information Access*, pages 84–92, Edinburgh, Scotland, UK, 2005.

[176] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *International Conference on Research and Development in Information Retrieval SIGIR05*, Salvador, Brazil, 2005.

[177] J. A. Tomlin. A new paradigm for ranking pages on the world wide Web. In *Proceedings of the 12th International World Wide Wed Conference*, pages 350–355, Budapest, Hungary, 2003.

[178] E. Ukkonen. On-line construction of suffix trees. Technical Report A-1993-1, Department of Computer Science, University of Helsinki, 1993.

[179] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, 1962.

[180] Y. Wang and M. Kitsuregawa. On combining link and contents information for Web page clustering. In *Proceedings of Database and Expert Systems Applications*, pages 902–913, Aix en Provence, France, 2002.

[181] D. Weiss and J. Stefanowski. Web search results clustering in Polish: Experimental evaluation of Carrot. In *Proceedings of Intelligent Information Processing and Web Mining*, pages 209–218, Zakopane, Poland, 2003.

[182] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann Series, 2005.

[183] I.A. Witten, A.Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kauffam, 1999.

[184] Y. Wu and X. Chen. Extracting features from Web search returned hits for hierarchical classification. In *Proceedings of International Conference on Information and Knowledge Engineering*, pages 103–108, Las Vegas, U.S., 2003.

[185] O. Zamir. *Clustering Web Documents: A phrase based method for grouping seach engine results*. PhD thesis, University of Washington, 1999.

[186] O. Zamir and O. Etzioni. Grouper: a dynamic clustering interface to Web search results. In *Proceedings of 8th International World Wide Web Conference*, pages 1361–1374, Toronto, Canada, 1999.

[187] H. Zeng, Q. He, Z. Chen, and W. Ma. Learning to cluster Web search results. In *Proceedings of SIGIR04*, pages 210–217, Sheffield, U.K., 2004.

[188] B. Zhang, H. Li, Y. Liu, L. Ji, W. Xi, W. Fan, Z. Chen, and W. Ma. Improving Web search results using affinity graph. In *International Conference on Research and Development in Information Retrieval SIGIR05*, Salvador, Brazil, 2005.

[189] D. Zhang and Y. Dong. Semantic, hierarchical, on-line clustering of Web search results. In *Proceedings of Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference*, pages 69–78, Hangzhou, China, 2004.

[190] H. Zhang, A. Goel, R. Govindan, K. Mason, and B. Van Roy. Making eigenvector-based reputation system robust to collusion. In *Proceedings of Third International Workshop on Algorithms and Models for the Web-Graph, WAW*, pages 92–104, Rome, Italy, 2004.

[191] `http://blogs.law.harvard.edu/tech/rss/`.

[192] `http://labs.google.com/personalized/`.

[193] `http://mysearch.yahoo.com/`.

[194] `http://myjeeves.ask.com/`.

[195] `http://newsbot.msnbc.msn.com/`.

[196] `http://news.google.com/`.

[197] `http://news.yahoo.com/`.

[198] `http://rankcomparison.di.unipi.it/`.

[199] `http://roquefort.di.unipi.it/~gulli/listAllowed/testSnakeT/`.

[200] `http://roquefort.di.unipi.it/~gulli/listAllowed/testing/`.

[201] `http://searchengineshowdown.com/stats/`.

[202] `http://searchenginewatch.com/`.

[203] `http://searchenginewatch.com/reports/article.php/2156481`.

[204] `http://searchenginewatch.com/searchday/article.php/34711_3527636`.

[205] `http://snowball.tartarus.org/`.

[206] `http://media.timewarner.com/media/newmedia/cb_press_view.cfm?release_num=55254348`.

[207] `http://www-unix.mcs.anl.gov/petsc/petsc-as/`.

[208] `http://www.a9.com/`.

[209] `http://www.alltheweb.com/`.

[210] `http://www.altavista.com/`.

[211] `http://www.amazon.com/`.

[212] `http://www.archive.org/`.

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

[214] `http://www.atomenabled.org/`.

[215] http://www.bbcworld.com/.

[216] http://www.betanews.com/article/Microsoft_Tests_Search_Clustering/1106319504.

[217] http://www.cpan.org/.

[218] http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html.

[219] http://www.e-marketing-news.co.uk/Aug04-Greg.html.

[220] http://www.epinions.com/.

[221] http://www.eurekster.com/.

[222] http://www.findory.com/.

[223] http://www.google.com/.

[224] http://www.google.com/searchhistory/.

[225] http://www.internetnews.com/stats/article.php/1363881.

[226] http://www.neci.nj.nec.com/homepages/lawrence/.

[227] http://www.newscientist.com/channel/info-tech/mg18624975.900.

[228] http://www.newsinessence.com/.

[229] http://www.nielsen-netratings.com/.

[230] http://www.nutch.org/.

[231] http://www.pewinternet.org/pdfs/PIP_SearchData_1105.pdf.

[232] http://www.rednova.com/.

[233] http://www.reuters.com/.

[234] http://www.searchenginelowdown.com/2004/10/Web-20-exclusive-demonstration-of.html.

[235] http://www.teoma.com/.

[236] http://vivisimo.com/docs/personalization.pdf.

[237] http://www.yahoo.com/.