NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

Assessment of the EDQ-Rank Link Analysis Algorithm

by

Simon Mc Cann, BSc.

Masters Thesis

Submitted to the National University of Ireland, Maynooth

Department of Computer Science

Hamilton Institute

National University of Ireland.

Maynooth

Co. Kildare,

November 2005

Research Supervisor: Prof. Barak Pearlmutter

Abstract

The expansion and use of the web has proceeded at an exceptional rate since its conception in 1990, with current estimates of over 11.5 billion documents and nearly 1 billion users (14.9% of the world's population). As this growth continues, so too does the pivotal role of search engines, with the majority of users selecting a search engine as their gateway to the Internet. A problem with current search-engine results is that often a page important in the context of the entire web is returned in preference to a page that is important in relation to the user query. To counteract this deficiency, we propose 'EQD-Rank' to refine the result-sets generated using Google's PageRank algorithm. The premise behind EQD-Rank is that a hyperlink from a topically-equivalent page, is more important than a hyperlink from a topically disparate page. EQD-Rank is based on local-graph traversal and implementable at runtime, manipulating a PageRank vector computed "*a-priori*". Comprehensive evaluation of a link analysis ranking algorithm is a non-trivial matter and within this thesis we provide a testing-environment framework involving dataset compilation, efficient corpus representation, and an evaluation of the EQD-Rank algorithm.

Acknowledgements

The research performed during this thesis would not have been possible without the combined patience and help of many colleagues, friends, and family alike. In particular I would like to thank my research supervisor Barak Perlmutter for providing the opportunity to perform this research, under his diligent supervision. Also I would like to thank Prof. Douglas Leith for allowing me to perform this research at the Hamilton Institute, NUIM Maynooth.

My time at the Hamilton Institute has been a truly enriching and enlightening period, aided by the rich ambiance and assiduous approach to research. In particular, I would like to thank my office colleagues: Yoshio Konno, Ross O'Neill, Santiago Jaramillo, Paul O'Grady, Steven Strachan, and Yunong Zhang, for their friendship and support.

I would also like to thank the members of Hamilton F.C.: Dr. Ollie Mason, Mark Verwoerd, Eric Bullinger, Peter Clifford, Dimitris Kalamatianos, Selim Solmaz and Rick Middleton. It was through the brave efforts and dedication of these fine sportsmen, and hours of hard word on the training field that resulted in the epiphany of an 8-1 defeat of the Computer Science department, in a recent interdepartmental challenge match. Let's hope there are many more such triumphant occasions.

Also I would like to thank Diarmuid O' Donohue with whom I began this epic voyage of discovery during my BSc. Thesis. I would like also to thank all the other people of the Hamilton Institute not mentioned specifically above, for their support and friendship and last but not least I would like to thank Rosemary Hunt and Kate Moriarty for providing support and smiles (also muffins!) throughout the period.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Today the web is the largest repository of human information, with an estimated 11.5 billion publicly accessible pages [GS05]. The information contained within this hypertext repository lacks the formal structure present in previous Information Retrieval document collections. In the case of the web, a diverse range of information is available for human digestion in a semi-structured environment. Documents available not only differ in quality but also in language, range of vocabulary and type (.asp, .HTML, jsp, etc). To add to the unpredictability of this environment, it is estimated that over 1 million pages are added to the web daily, not uniformly at random but with greater probability to existing pages that are already highly linked to [BA99].

The task facing modern day search engines involves exploiting this dynamic environment, to process over 250 million web searches daily and therefore attempt to satisfy every user information need. An additional problem facing search engines is the very nature of the user query involved. The complexities involved are aptly highlighted through an example of a user information need.

“ Find all pages containing information on the Java programming language which: (1) are non-commercial and (2) contain working examples.” To be relevant, all pages returned must also include e-mail and phone number for contact purposes along with a link to the Sun website.”

This query cannot be used directly to return relevant pages and must be translated into the language of the information system. In this case, in the language of the search engine, by specifying words that adequately convey the semantic representation of this information need. The problem being the search engine uses natural language and is therefore ambiguous and imprecise. Also, another major problem is that the user need may be poorly defined or broad in nature. This is particularly true when you consider that a user query typically contains 2.35 terms. The fact that most IR users are prepared to examine at most the first 10 or 20 documents [BWB02, SHMM98], again adds to the complexity involved.

Taking all of these factors into consideration, the relevancy achieved by the leading search engines (Google, Yahoo, MSN) could be considered relatively adequate. This success has centered around exploitation of the World Wide Web link topology, with edges representing links between pages and nodes representing pages. Explicitly this exploitation is achieved through link analysis ranking algorithms. Current ranking algorithms exploit this inherent context information, through the premise that a link from page A to page B denotes an endorsement of the quality of B. A few of the most cited link analysis ranking algorithms are Google’s PageRank algorithm, Kleinberg’s HITS algorithm [Kle99], and Lempel and Moran’s SALSA algorithm [LM00].

In this thesis we focus on the PageRank algorithm and its query independent nature. We suggest a modification of the result set at runtime using a post-ranking refinement of the result set, to increase the relevance returned to a typical user. This modification enables the reproduction of a query dependent PageRank result set at runtime and is therefore the focus of this thesis.

1.2 Information Retrieval

The goal of Information Retrieval (IR) involves satisfying the information need of an individual. This problem may initially appear straightforward until we consider the specificity of the user query and structure of the data collection.

The type of user query input to a data collection covers an entire spectrum [CPKT92]. At one end, a narrowly specified search for a particular document, may be implemented through a query such as the document title. At the opposite end is a browsing session with no well defined goal, satisfying a need to learn more about a topic. It is also common for a user to move across this spectrum from browsing to search, with the user beginning with a partially defined information need that is refined as they learn more about the topic. The objective of an IR system is to extract the semantic meaning inherent in these user queries, regardless of the specificity.

Information Retrieval is also concerned with representation, storage, organization of, and access to information items with a view to satisfying a user's information need. With regards to the storage of information, documents may be represented using a set of index terms or keywords. These keywords may be extracted directly from the text or specified by human subjects. Also due to the availability of large disk space, storage of a full representation of a document is possible through its words. In practice, however, a full representation is rarely stored and methods such as stemming, elimination of stop words, and identification of noun phrases are all implemented. Furthermore this data is often compressed further using modern compression techniques. Currently the two main evaluation techniques implemented for IR are recall and precision [Cle97], with recall defined as the proportion of relevant documents retrieved by the system and precision described as the proportion of retrieved documents that are relevant. The overall goal of an IR system is to retrieve all the documents relevant to a user query, while retrieving as few of the non-relevant documents as possible. In the following subsections we trace the development of information retrieval from ancient times to web information retrieval.

1.2.1 Early Information Retrieval (pre 1945)

The storage and retrieval of information dates from 700Bc and a letter from the Assyrian king, referring to the storage of books. During ancient times, books were stored in libraries with the most famous being the library of Alexandria. Callimachus (305-240 BC) devised a classification system for this library, divided into six poetic genres and five prosaic areas (history, rhetoric, philosophy, medicine and law, and varia). For each category, the books were arranged alphabetically by author and for each book an incipit (the first lines of the book) was stored with author dates.

Between the time of these ancient libraries and the libraries of the middle ages, papyrus was replaced by parchment and the scroll was replaced with the bounded book, making texts more compact and manageable. Generally, books pre 19th century were classified by subject and assigned rooms and shelves based on the method suggested by Callimachus. Catalogues were kept by most libraries containing three parts consisting of a list of titles containing the volume, number of leaves and number of separate treatise contained within the book, a shelf list, and an author list in alphabetical order of the entire collection.

The invention of the printing press and the rapid growth of literacy made the retrieval of information more difficult and therefore centred on book subject rather than the author. In 1876 Melvil Dewey introduced the idea of relative, as opposed to, absolute location and the assignment of numbers to books, rather than shelves, with the subject identity being attached to the book, instead of the shelf. Books were placed relative to each other using a linear numbering scheme, hierarchically divided in 10 classes by 10 divisions by 10 sections. The Universal Decimal Classification by Paul Otlet and Henri-Marie Lafontaine, used in many libraries is a descendent of Dewy's work. S.R. Ranganathan followed this work with facet analysis, the technique of dividing a subject into its complex parts by relating them to a set of five fundamental categories named personality, energy, matter, space and time, in his Colon Classification system (1933). Controlled dictionaries and thesauri were also introduced around this time to instill some semantic order [Fos82].

1.2.2 Modern Information Retrieval

The popularization of the idea of information Retrieval began in 1945, with Vannevar Bush's article [Bus45]. Bush envisaged a system that would store the individual notes and memories of scientists in his Memex machine. "A Memex is a device in which an individual stores his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged supplement to his memory". This type of associative linking was behind the idea of automatic access to vast amounts of stored information.

The first information systems built in the 1950's included the invention of Key Word In Context (KWIC) indexes. These indices were a computer generated concordance, with a concordance being an alphabetical list of important words of a book or author, with reference to the passages in which they occur. The program searches for keywords and prints them surrounded by the context in which they occur. The WRU Searching Selector program was also introduced at this time by Allen Kent [RK58] to perform literature searches based on encoded abstracts. Also in 1959, H. P Luhn proposed using words as indexing units for documents and measuring word overlap as a criterion for retrieval [Luh57].

In the early 1960's Cyril Cleverdon developed the mathematics of 'recall' (the fraction of documents retrieved) and 'precision' (the fraction of documents retrieved that are relevant), for the evaluation of retrieval systems and built the first test collections to measure them [Cle97]. In 1965 Ted Neilson coined the term hypertext with it defined as "a combination of natural language text with the computer's capacity for branching, or dynamic display and text which is not constrained to be linear".

Also, during this period, Salton introduced the Vector space model [SWY71]. Using the Vector Space Model (VSM) text is represented as a vector of terms. The terms are usually either words or phrases. If words are represented as terms, every word becomes a dimension in a high dimensional vector space and any text document can be represented as a vector. As any text contains a limited number of terms, most of these vectors are sparse. The VSM is used to assign a ranking to a document based on the potential similarities of the document to the query. The document score is based on the similarity between the query vector and the document vector. The angle between two vectors is used as a measure of divergence, with the cosine of the angle or dot product used in most instances.

Similar to the VSM, the Probabilistic model (PR) represents documents and queries as vectors. However, the PR model is founded on the premise that documents in a dataset should be ranked by decreasing probability of their relevance to the query. This premise is known as the probabilistic ranking principle [Rob75]. The notion of probabilistic retrieval was introduced by Maron and Kuhns [MK60] with term weights calculated based on term distribution within the documents under evaluation. The probability of document relevance is calculated by summing individual term relevance weights which are estimations of probabilities that query terms will appear in relevant and not irrelevant documents.

Numerous methods exist to assign term weight in the VSM and PR model. Most of these are variants of inverse document frequency (idf) weight [Jon72]. This weighting scheme is based on the inverse of the number of documents in which the term appears. Often this is coupled with term frequency (tf) to assign a ranking, where term frequency is simply a term's frequency within a document. Also taken into consideration is the length of the document, with longer documents tending to score higher, due to more word occurrences and word repetitions.

J. J. Rocchio introduced Relevance feedback in 1965 to help refine the user query [Roc71]. In this instance results are returned to the user ranked by some relevance measure. Following examination of the results, a further subset of results may be selected which generate a higher level of interest. The selected pages are then used to further refine the user query with the resulting method therefore called a ‘user feedback cycle’.

With the advent of computerized databases, the idea of ‘free-text searching arose’, where complete retrieval of any document, using a particular word, could be achieved with no cost for manual indexing. Multi-lingual experiments were conducted at this time, using a bilingual thesaurus and mapping words from both languages into the same concepts. An example of the existing multi-lingual thesaurus is EuroWordNet. Constructed by many European countries, EuroWordNet uses a structure called ‘interlingual index’ to represent the semantic structure of multilingual words [Pie98].

During the 1980’s online information retrieval expanded in two main ways. The first being the availability of full text instead of just abstracts and indexing and the other was the use of online retrieval by non-specialists, as libraries replaced or supplemented their card catalogues with online public access catalogues. The CD-ROM was introduced at this stage, with most libraries having at least one CD-ROM drive. The probabilistic and vector space approaches that have been refined over the past number of years cannot claim to imply real understanding of the documents and as Cleverdon observed human indexing is not consistent enough to guarantee acceptable recall and precision over sizable databases. In the 1990’s frequency-based, word orientated models were again adopted and the large scale corpora of machine readable text, such as that provided by the TREC collection provided a suitable testing environment, with the first TREC evaluation conference established in 1992 [Har93]. The development of the World Wide Web by Tim Berners Lee at this time revolutionized Information Retrieval.

1.2.3 Information Retrieval and the Web

Until the early 1980's Information Retrieval was viewed as a narrow area of interest by librarians and information experts. The World Wide Web was introduced in 1990 by the European Laboratory for Particle Physics (or CERN), as a way for physicists to track the progress of other individuals. The concept involved the ability of people working in different locations to learn what each other were working on, through examination of a hypertextual document, accessible on the Internet. Creation of the World Wide Web is credited to Tim Berners Lee. His original vision for the web was the following: "My definition of the Web is a universe of network-accessible information, a means of human-to-human communication, and a space in which software agents can, through access to a vast amount of everything which is society, science and its problems, become tools to work with us." The modern web has evolved from this vision into a vast repository of human knowledge, thus fulfilling Bush's previously discussed vision. However this repository of information adds a new dimension of difficulty to the traditional information retrieval problem.

In traditional information retrieval, content based metrics are used to retrieve relevant documents from a structured environment. In the case of Information Retrieval on the Web, however, the collection of documents is contained in what can be described as a semi-structured environment. Documents are constantly added and removed from the Web, with it suggested in [NCO04] that every week 25% new links are created, and that after a year 80% of the hyperlinks are replaced with new ones. It is further suggested that new pages are created at a rate of 8% per week and after a year roughly 50% of the web content is new. The content of information present in the web is diverse not only with respect to quality but also in language, range of vocabulary and type (.asp, .HTML, .jsp, etc).

Content-based metrics such as TF*IDF, used in traditional information retrieval, assume the integrity of data in the document collection. However, this assumption is not applicable with Information Retrieval on the web. Numerous methods have evolved for the commercial exploitation of the web. Invisible text, page cloaking, and page redirects are a few of the more popular methods used to mislead search engines and increase the ranking of a website. Also in the case of context, the ranking mechanism must take into consideration reciprocal link programs and other local link structures, manufactured for the sole purpose of improving a website's ranking.

As a result, search engines use a combination of content and context score when assigning a ranking to a page. Through analysis of the structure of the web, we demonstrate how context information is effectively exploited by the leading search engines.

1.3 Graph Structure of the Web

The web can be viewed as a directed graph, with every page represented as a node, and the hyperlinks between pages represented as edges. Therefore the web graph of today is comprised of 11.5 billion nodes and approximately 92 billion edges [GS05]. The web is constantly evolving and as a result the structure is dynamic rather than static. Nodes and edges are constantly being removed from and added to this hyperlink structure, with it suggested in [KRRT99] that the half-life of a web page is in the order of a few weeks.

The evolution of this structure has been the focus of much research in the past and suggested in [BA99] is that the addition of pages is not random and follows a model of preferential attachment. Under this model a link is created from a new node to an existing node not uniformly at random, but with a higher probability to existing nodes that already have a large in-degree. This is a 'winner takes all' situation, present in a large number of social networks. This model, however, does not account for the distributions present in the low connectivity regions, observed in [BKM+00] as possibly Poisson or a combination of Poisson and power law. Suggested in [PFL+02] is a combination of preferential attachment and uniform attachment, to better account for this distribution in the low connectivity regions. Also suggested is another model in which "A new page adds links by picking an existing page, and copying some links from that page to itself" [KRRT99]. This is based on the notion that a user will select pages to link to, based on pages that the user has a preference for.

A hyperlink pointing from a page is said to be an out-link of that page. With regards to the page that it points to, the link is referred to as an in-link. The number of in-links to a page is subsequentially referred to as the In-Degree and the number of out-links from a page is referred to as the Out-Degree. The In-Degree and Out-Degree distributions are conjectured to follow a power-law distribution, with best fit exhibited for exponents of 2.1 and 2.4 respectively [BA99, BKM$^+$00, Wan]. These distributions are also inverse polynomial distributions and can be referred to as Zipfian distributions [Zip49].

Initially the structure of the web may appear totally haphazard, but further investigation reveals an underlying structure. This structure, exposed by Broder et al. [BKM$^+$00], during experimentation on a web dataset of over 200 million pages, is likened to that of a 'bow-tie'. Revealed is a central Strongly Connected Component; a subgraph (IN) with directed paths leading to the SCC, a component (OUT) leading away from the SCC and isolated tendrils attached to one of the subgraphs. These four regions are roughly a quarter of the total size of the web each. Later work by Dill et al. [DKM$^+$01] showed that in subgraphs of the web this structure is maintained, although the ratio of component sizes differs somewhat.

Overall the structure of the web may be viewed as sparse (the average out-degree is eight [KRRT99]), but at a local level the structure is dense. The hyperlink structure at this level maintains different properties from those associated with the global hyperlink properties. According to [EM], locality with respect to hyperlinks in the web can be defined through the premise that "links tend to be correlated to links that are nearby in some measure". This locality among links is reinforced by Adamic for a "small world" graph of the World Wide Web [Ada99], and is further observed in [SY01]. In the latter it is suggested that "almost three quarters of links point to pages on the same hosts, and often to pages that are only a short distance from the source in the sorted order of URLs". Also a form of locality is highlighted by Davison [Dav00] and Menczer [Men02], with it suggested that pages linked to or from the same page are usually topically related.

It is at the local level that we believe the structure of the web may be further exploited to identify collections of topically-related pages, and therefore increase the relevance of the results returned to a search engine user. Bi-partie cores are identified in a large dataset of the web [KRRT99], with it suggested that a large fraction are in fact topically coherent. Community structures are also identified within the web by [FLGC02]. In this instance, a community is defined as a collection of web pages such that each member has more hyperlinks within this community, than outside of the community. Later experimentation, discussed within this thesis, focuses on topically-related subgraphs and attempts to fully exploit the inherent information.

1.4 Link Analysis Ranking Algorithms

The web as a document collection is unique in comparison to traditional document collections, with the assignment of a ranking value to every hypermedia webpage being a non-trivial matter. Firstly, this is due to the previously discussed diversity of documents and constant addition of new pages to the environment. Consider also, the tremendous size of the web, that the web contains redundant documents and broken links, and that certain local linkage-structures may be created for the commercial exploitation of the Web. These attributes highlight the dynamic instability inherent in this hypertext corpus.

Other problems facing link analysis ranking algorithms spawn from over-simplistic user queries and possible cognitive overload resulting from queries of broad specificity.

Dealing with queries sufficiently broad in nature is defined as the Abundance Problem by Kleinberg: “The number of pages that could be reasonably returned as relevant is far too large for a human user to digest” [Kle99]. Also considering the fact that a user query typically contains 2.35 terms [Hen03] and the fact that most IR users are prepared to examine at most the first 10 or 20 documents [SHMM98, BWB02], the critical nature of this problem becomes apparent.

As a result traditional information retrieval techniques are insufficient and a social network view of the web is adopted along with the associated analytical-techniques. Using a directed graph to model a social network is not unique to the web, with in-degree intuitively a good indicator of status. In 1949, J. R. Seely [See49] realized the recursiveness present in a social network, with web pages being replaced by actors: “we are involved in an infinite regress:[the actor’s status] is a function of those who choose him; and their status is a function of those who choose them, and so *ad infinitum”*. Such social network and therefore web properties form the foundations on which the majority of todays link analysis ranking algorithms are based.

Current link analysis algorithms exploit the context information inherent in the hyperlink structure of the Web, with the premise underlying all link analysis ranking algorithms, being that a link from page A to page B denotes an endorsement of the quality of B. Following on from social network analysis, an initial indication of page importance is the number of in-links. This is the premise behind the RankDex algorithm [Ran49], with pages assigned ranking values based on their number of inlinks. The PageRank algorithm [PBMW98] used by the Google search engine, is a descendant of RankDex. PageRank is based on the premise that the importance of a page is determined by the importance of the pages linking to it, and is a manifestation of Seely’s observation.

The hyperlink structure of the web is exploited by three other important link analysis ranking algorithms. Kleinberg’s HITS algorithm [Kle99] attempts to exploit this structure through the use of hubs and authorities over a root set, K. Bharat and G. Mihaila exploit the link structure between expert and target documents, through their Hilltop algorithm [BM01], and Lempel and Moran [LM00] exploit this structure through their SALSA algorithm. In the following subsections we briefly discuss the PageRank algorithm, some query dependent algorithms, and give an overview of other algorithms of interest.

1.4.1 The PageRank Algorithm

As previously discussed, the Rankdex algorithm [Ran49] is based on the hypothesis that the 'importance' of a page should be determined by its number of inlinks and is essentially a form of simple citation counting. The problem with Rankdex is that it is highly susceptible to spamming and does not favour important pages with only a few in-links. PageRank evolved from this simple idea into an iterative ranking process over the entire web graph. The hypothesis behind the PageRank algorithm is that a page with a large number of in-links, or a link from an important page, should be deemed important. The PageRank of a page is therefore based on its In-Degree and the importance of pages linking to it.

The PageRank Algorithm:

$$\mathit{Pr}\left(i\right) = \left(1 - D\right) + D \sum_{j \in A_i} \mathit{Pr}\left(j\right) / \textit{out-links}\left(j\right) \tag{1.1}$$

The PageRank Pr of a page i is the sum of the PageRank Pr of its in-links Ai, divided by the number of their corresponding out-links.

The PageRank algorithm is used to model the behaviour of a typical person browsing the Web. This person, often referred to as a 'random surfer', will choose to follow one of the links from a page with equal probability, and occasionally get bored and jump to another page completely at random. The computation of the PageRank vector involves an iterative process over a square adjacency matrix A and is equivalent to extracting the principal eigenvector of the matrix A. A detailed description of the PageRank algorithm is given in chapter 4.

1.4.2 The HyperLink Induced Topic Search algorithm (HITS)

Kleinberg [Kle99] suggests a different method for ranking web pages through the introduction of hubs and authorities. The HITS technique is based on the premise that a sufficiently broad topic contains communities of pages. In order to rank pages each relevant page is assigned a hub and authority score. An authoritive page is a highly referenced topic-relevant page and a hub page is a page pointing to many authoritive pages.

The HITS algorithm:

$$\begin{aligned} a\left(p\right) &= \sum_{q \to p} h\left(q\right) \\ h\left(p\right) &= \sum_{p \to q} a\left(q\right) \end{aligned} \tag{1.2}$$

The authority value of page p ($a\left(p\right)$) is the sum of hub scores of all pages pointing to p and the hub value of page p ($h\left(p\right)$) is the sum of authority scores of all pages that p points to.

Initially all hub and authority values are set to 1. In a single iteration the authority score of page p is replaced by the sum of the hub score of pages pointing to p, and the hub score of page p, is replaced by the sum of the authority score of pages pointed to by p. As a result hubs and authorities exhibit a mutually reinforcing relationship, where a good hub points to many good authorities and a good authority is pointed to by many good hubs. Convergence of the hub and authority values is guaranteed [Kle99] and typically occurs within 10 iterations [BH98].

HITS is inclined to favor the most dense community of hubs and authorities and is referred to as the 'Tightly Knit Community effect'. This preference can lead to an irrelevant result set and is due to query-irrelevant pages in the initial root set. A second drawback of HITS is a mutually reinforcing relationship occurring between hosts, with many pages on one host linking to a single page on another host. The resulting authority score of the single page is inflated and subsequently so too are the hub scores of the pages pointing to it.

Two other drawbacks of HITS are identified by Chakrabarti et al. [CDR+98], with the first occurring when a page discusses a number of topics. A link will point to different topics depending on their location within the page. If the page has a large out-degree, it will receive a large hub weight, which flows on as high authority weights for referenced pages, regardless of relevance to the initial query topic. The second drawback is topic generalization. If the search topic is specific, the algorithm often returns good sources for more general topics. This behavior can also be reversed with pages more specific than the general search topic, dominating the results set.

1.4.3 The Topic Sensitive PageRank algorithm

With Topic Sensitive PageRank [Hav02] a set of ranking vectors are computed, as opposed to the single PageRank vector generated using standard PageRank. These vectors are biased using a set of representative topics, to capture the notion of importance with respect to a topic, indirectly specified through a user query and if available through user context also.

During offline processing of 120 million pages, indexed during a web crawl in January 2001, 16 topic-sensitive PageRank vectors are generated, each biased using URLs from a top-level category of the Open Directory Project. At query time a linear combination of the topic-sensitive vectors, weighted using the similarities of the query (and context if available) is used instead of a single ranking vector.

This is an extension of the idea of a personalization vector suggested in [BMPW98]. In terms of the random-walk model, each topic vector represents the addition of a complete set of transition edges, and with probability D the random surfer follows an out-link from the current page, occasionally jumping to a URL in the ODP category. The influence exerted by each topic, over the final rank value, is proportional to their affinity with the query or query context.

Topic Sensitive PageRank lacks practical application as 16 topic categories is insufficient to cover all topics on the web and therefore a query may be likened to a category topic, with which it has little affinity. A more comprehensive implementation involving a vast number of category topics would be required with the use of a finer-grained set of topics, such as the second or third level of the Open Directory hierarchy, as suggested within the paper. However such an implementation would have a profound effect on the computation and storage constraints.

1.4.4 The Hilltop Algorithm

The Hilltop algorithm [BM01] works on the same hypothesis as most connectivity-based algorithms, namely that the number and 'quality' of pages pointing to a particular page are an indication of the page's quality. The Hilltop algorithm is unique through the fact that only pages considered 'experts' for a user specified query are considered. An expert page is defined as a page about a certain topic having links to many other non-affiliated pages on that topic. Initially, a list of the most relevant experts on the topic is computed, based on a user query. The relevant links contained within these experts are then selected and followed, leading to the target pages. The rank value of target pages is determined by the number and relevance of non-affiliated experts pointing to them. Target pages must be linked to by at least two non-affiliated expert pages. The ranking value of a target page therefore reflects the collective opinion of the best independent experts on the query topic. When no such experts are available, no results are returned by Hilltop.

A common consensus in the search engine world is that some form of the Hilltop algorithm is currently implemented by the Google search engine, for broad queries. The shortcoming of this method is that computation must be performed "*a-priori*". Therefore if the user query was not previously considered, standard PageRank algorithm is implemented, leaving the query susceptible to results that are authoritative in general, but not in relation to the user query.

1.4.5 The Intelligent Surfer Algorithm

The Intelligent Surfer algorithm [RD02] attempts to improve upon the standard PageRank algorithm by introducing a more intelligent random surfer. This surfer is guided by a probabilistic model of relevance to the query, with the probability distribution given by:

The Intelligent Surfer Algorithm:

$$P_q(j) = (1-\beta)\, P'_q(j) + \beta \sum_{i \in B_j} P_q(i \rightarrow j) \tag{1.3}$$

where β is a damping factor $0 \leq \beta \leq 1$ and $P_q(i \rightarrow j)$ is the probability the surfer moves to page j given that he is on page i. Where the surfer jumps when not following links is specified by $(1-\beta)P'_q(j)$. The resulting probability distribution over pages is given by $P_q(j)$ and both $P'_q(j)$ and $P_q(j)$ are derived from a measure of relevance of page j to query q.

The above equation is best explained using the 'random surfer' analogy. The web surfer probabilistically hops from page to page, depending on the page contents and the user query. When choosing between multiple out-links from a page, the intelligent surfer will choose a random link from the set of pages deemed query relevant, instead of one at random from the entire set of page out-links. When a page has no out-links or none deemed relevant, links are added from the page to all pages in the dataset. To assign a measure of relevance to a page, a basic content-based function is suggested, where the relevancy of page j for query q ($rq\,(j)$) is equal to 1, if the query term appears in the page, and 0 otherwise. Only pages related to the user query ($rq\,(j) = 1$) are considered.

The bottleneck for the Intelligent Surfer method is that each query cannot be processed at runtime within a reasonable time frame. Suggested as a solution, is the calculation of the Query Dependent PageRank (QD-PageRank) vector for hundreds of thousands of query words "*a-priori*". At runtime, provided the user query matches one of these words, the corresponding QD-PageRank vector is then used to populate the results set. Computation and storage is stated as requiring roughly 100-200 times that of standard PageRank, for hundreds of thousands of words. The drawback being that provision of a QD-PageRank value for every possible query word, "*a-priori*", is not realistic.

1.4.6 Other Algorithms and Ranking Methods

Numerous variants of the PageRank algorithm currently exit. Richardson and Domingos [RD02] and Haveliwala [Hav02] create two variations previously highlighted. Other PageRank Personalization variants are discussed by Jen and Widom [JW03] and Brin et al. [BMPW98]. Rafiefi and Mendelzon [RM] consider a variant of HITS, incorporating random jumps to determine the topic associated with a page. Chakrabarti et al. [CDR$^+$98] and Bharat and Henzinger [BH98] come up with improvements to the HITS algorithm by adding weights to nodes and links, based on page relevance using a cosine similarity measure.

Cohn and Chang [CC00] provide a probabilistic analogue of the HITS algorithm through their PHITS approach, with the model attempting to explain documents and citations in terms of common community-factors. Modifications to Kleinbergs HITS algorithm based on a Bayesian statistical approach are suggested in [BRRT01]. Cohn and Hofmann propose an extension of their PLSA and PHITS algorithms by creating a mixture based on a probabilistic factor to allow the identification of authoritive documents. Methods to overcome the effects of local aggregation through hyperlink evaluation and evaluation-based web ranking are suggested in [Wan03]. The idea involves the weighting of links between different domains, more strongly than links within a single domain, and weighting them based on the distance between pages.

1.5 Overview

Highlighted in the link analysis ranking algorithms discussed todate, is the inability to incorporate the user query at runtime, within a reasonable time bound. We therefore propose the introduction of the EQD-Rank algorithm to combine content and context information in a post ranking refinement of the PageRank results-set. The aim of EQD-Rank is to approximate the query dependent PageRank vector, through a manipulation of the PageRank vector, based on a local-graph traversal. Initially the PageRank vector is computed "*a-priori*" and is then adjusted based on the relevance of each of the page in-links.

This ranking refinement is necessary due to the fact that the topic subgraph for each query, cannot be reproduced within a reasonable time bound, in an environment where the user is prepared to wait at most between 10 and 20 seconds. Following implementation and testing of the algorithm, we show through a user study that EQD-Rank can be used to achieve a relevance increase of over 20%.

During subsequent chapters the elements involved in analysis of a link-based algorithm are discussed along the results obtained. The crawler used to download the dataset and its various components are highlighted in *chapter 2*. Also examined are the various challenges faced by modern day crawlers and comparisons are made with other crawlers of interest. In *chapter 3* we examine the various aspects associated with the storage of a large-scale dataset, creation of the inverted index, creation of the transpose matrix and the memory saving techniques implemented. *Chapter four* describes the implementation of the standard PageRank algorithm, QD-PageRank and our EQD-Rank refinement, along with the various algorithmic strengths and weaknesses. In *chapter 5* we examine the ranking distributions and certain result-set properties. Also examined are the convergence rates and some content-based document selection techniques. In *chapter 6* we reach our conclusions based on experimental results and discuss potential future work.

Chapter 2

The ViperBot Crawler

2.1 Introduction

A modern search engine must index the huge repository of information known as the web, as quickly as possible. A program or collection of programs capable of achieving this, is referred to as a crawler, spider, or bot. Shkapenyuk and Suel [SS02] note that: “While it is fairly easy to build a slow crawler that downloads a few pages per second for a short period of time, building a high-performance system that can download hundreds of millions of pages over several weeks presents a number of challenges in system design, I/O and network efficiency, and robustness and manageability.”

A typical crawler cycle begins with a request for a page and the parsing of the body of the fetched page to extract links. These links are added to a datastructure containing the URLs of pages to be crawled, and the cycle continues until some termination condition is met or until the list of uncrawled URLs is exhausted. One important issue to be addressed when designing a modern-day crawler, is the crawl strategy to implement. One common strategy involves a *breadth-first* crawl. In this instance a seed page is parsed and the links are extracted. Every link on that page is then crawled until all links have been extracted. Each link on the page, represented by the first page's first link, is then processed, with the cycle continuing until the list of links is exhausted. Najork and Weiner [NW01] use *breadth-first* crawling to perform a crawl of 328 million pages and suggest that a *breadth-first* crawl discovers high PageRank pages early in the crawl. Boldi et al. [BCSV02] also test a *breadth-first* strategy against a random ordering and an omniscient strategy. Intuitively one would expect the omniscient strategy to perform the best, although it is suggested that "many strategies that accumulate PageRank quickly explore subgraphs with bad correlated ranks, and vica versa". Also an omniscient strategy requires previous knowledge of the region of the web to be crawled in advance, and it is therefore suggested that a *breath-first* strategy performs best. A *depth-first* strategy is also applicable, although less widely used. Using this strategy the first link on the first page is crawled. Following this, the first link on that page is crawled and the cycle continues until no more links can be obtained. Another possible method involves ordering pages based on the PageRank metric [CGMP98]. Numerous ordering methods for discovering important pages early during a crawl are also recently discussed, with a method using the PageRank from a previous crawl determining the ordering that performs the best [BYCMR05].

Crawlers are predominantly used to create a large repository of web pages. This is the function of crawlers such as Googlebot, Yahoo! Slurp and MSNBot. However, crawlers may also be used for numerous other tasks. A crawler may be used to re-crawl pages for updates. This is particularly important for constantly changing pages such as news webpages. In this instance, indexed page freshness is essential with Cho and Garcia-Molina [CGM00] previously considering methods to re-crawl a page based on its update history. Crawlers may also be used to perform some form of focused crawling such as downloading topic-specific pages [CvdBD99], images, mp3 files, or pdf files. Examination of a site for broken links and email address extraction are some other functions carried out by crawlers. M. Henzinger et al. [HHMN99] also use a crawler to perform random walks on the web, measuring search-engine index quality. To perform these functions, however, the crawler must avoid numerous pitfalls.

One of the main difficulties faced by a crawler involves avoiding a crawler trap. In this instance, a URL or multiple URLs cause the crawler to get stuck in an infinite loop. An example of this is a CGI script that dynamically generates an infinite number of web documents. No automatic method to detect a trap exists, although a site returning an extraordinary large number of documents can be an indication and can be blocked from consideration with future crawls. Other pitfalls are largely related to duplicate pages, resulting from a host containing multiple links to the same page. Host-name aliases are another area of concern, occurring when multiple host-names correspond to the same IP address. Another instance of the duplication problem involves same page replication across multiple hosts, with mirror websites and multiple web servers accessing the same file system, being prime examples.

The difficulties facing crawlers are compounded by the lack of work published to date, due to the highly competitive nature of the search engine industry. However there are some notable exceptions and these are discussed briefly in the next section.

2.2 Previous Work

Widely regarded as the first crawler, the World Wide Web Worm [McB94] was created in September 1994 by Oliver Mc Bryan at the University of Colorado. The crawler was designed to build an index of document titles and URLs and led to the creation of a dataset of 300,000 multimedia objects. Other Java crawlers of interest are the Internet Archive Crawler, GoogleBot, WebSphinx and the Mercator crawler.

The Internet Archive Crawler [Bur97] is distributed across a number of machines, with each process single threaded and using asynchronous I/O. The inter-process exchange of URLs is carried out in batch to minimize cost, with the overall aim of downloading periodic snapshots of the web.

Page and Brin [BP98] give a description of the initial Google crawler known as Googlebot, described as a fast distributed crawling system, containing multiple crawlers, with each keeping approximately 300 connections open at once. Each crawler process runs on an individual machine and contains a single thread using asynchronous I/O, to fetch data from up to 300 servers in parallel.

The WebSphinx crawler [MB98] from Compaq Systems Research Center, is a Java-based toolkit and interactive development suite for crawlers. Designed for site-specific, personal and relocatable crawlers, WebSphinx is multi-threaded providing library support for crawling in Java, using HTML parsing, pattern matching, and Web transformations.

The Mercator search engine [HN99] is used by the AltaVista search engine. Designed exclusively in the Java programming language scalability is achieved through its datastructures and through a modular design, it is also extensible. The Mercator search engine is used to create a snapshot of web pages on a corporate intranet, collect a variety of web statistics, and perform a series of random walks in the web. A detailed description of the system architecture is provided also, ideally for systems comparison.

2.3 The ViperBot System Architecture

2.3.1 Overview of the System

ViperBot is written entirely in Java. Java's object-orientated nature facilitates the addition of new components through subclassing or by method overriding. Java's API (Application Programming Interface) includes classes related to objects, threads, exception handling, and garbage collection, therefore aiding the design of an extensible and scalable crawler.

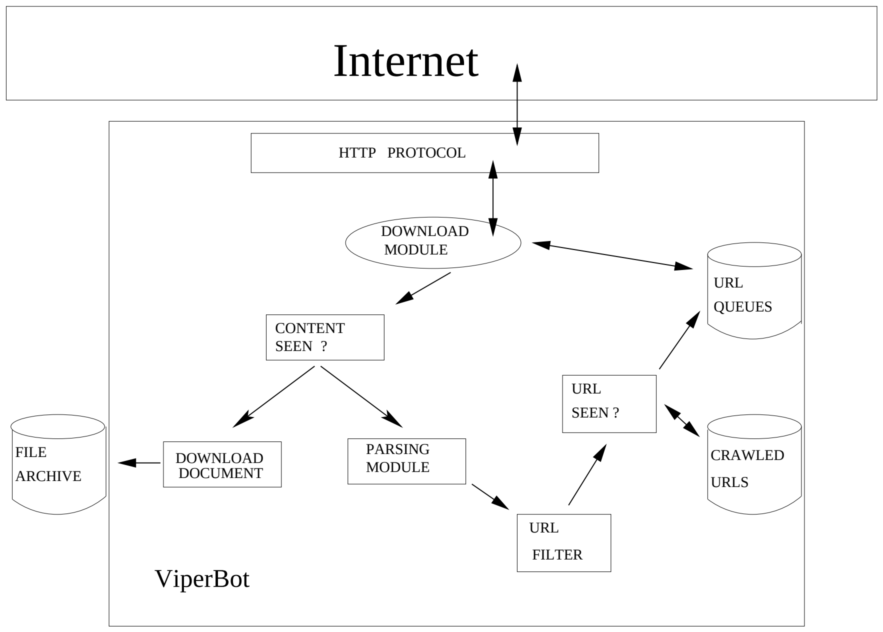

Figure 2.1: The ViperBot Architecture

The crawler is composed of 100 independent threads with thread management handled by the *DOWNLOAD MODULE*, as displayed in figure 2.1. The pre-defined maximum number of threads is one hundred, due to an excessive amount of threads resulting in the potential for more CPU time being spent swapping between threads, than actually processing them. Initially 100 seed URLs are selected and used to initialize the crawl. Seed pages may be randomly selected but to maximize the subsequent dataset diversity, the URLs are selected from the top-level categories of the Open Source Directory and Yahoo Directories. Each thread operates independently and once the crawl is initialized the threads are managed via a ThreadPool. A Random Access File (RAF) is initially created for each thread containing a seed URL and is represented in figure 2.1 by *URL QUEUES*.

Each thread continually processes URLs provided that the list of uncrawled URLs has not been exhausted. The URLs are selected sequentially from a thread specific

RAF, and extracted in a First In First Out (FIFO) manner. To establish a connection and download page contents within a specified time-frame a separate thread is created. Should the contents be successfully downloaded, the *CONTENT SEEN ?* module is invoked and used to prevent the download of duplicate pages. Qualifying pages are passed to the *DOWNLOAD DOCUMENT* module, where they are written to disk and stored in the *FILE ARCHIVE* for further offline processing. At this stage the *PARSING* MODULE is also invoked writing MetaData to disk and extracting fully qualified URLs. These URLs are subsequently passed to the *URL FILTER*, with URLs containing certain unwanted characters (i.e ? symbol) discarded.

A check to see if the URLs were previously crawled, occurs in the *URL SEEN ?* module, in the next phase. The check involves a comparison with the *CRAWLED URLS* datastructure. Should the URL surmount this condition, it is added to *CRAWLED URLS*, preventing a recrawl and is assigned to the appropriate RAF for further processing, with the cycle beginning once more. This cycle, performed by each thread, continues indefinitely until the list of uncrawled URLs is exhausted, or some global limit on the number of pages to be downloaded is reached.

2.3.2 The Download thread

The download thread is implemented via the networking facilities provided by java.net. Specifically we create a URLConnection through the URL protocol. The connection object is established by invoking the 'openConnection' method on the URL, with the actual connection to the remote object provided through the 'connect' method. Once the object becomes available page contents are downloaded. A problem with creating a 'URLConnection' is that no provision is made for the specification of a timeout value. A connection request can therefore hang indefinitely, should the remote server fail to close the connection. To resolve this problem we initially attempted to interrupt the thread, in which the connection is created. However, after much suffering it was discovered that a thread blocked on an I/O operation cannot be interrupted. As of Java 1.4 support is added for a socket timeout in the URLConnection class by setting the sun.net.client.defaultConnectTimeout and sun.net.client.defaultReadTimeout parameters. These parameters are therefore set as follows, with a timeout value of $8,000$ milliseconds specified:

System.setProperty("sun.net.client.defaultConnectTimeout","8000");

System.setProperty("sun.net.client.defaultReadTimeout", "8000");

If the connection is successful, page contents are downloaded to a string buffer, with a pre-specified capacity to prevent memory strain. The StringBuffer is doubled in size when more memory is required and therefore selection of an appropriate default size is essential. The download thread is allocated 10 seconds to connect to the page and download the contents, after which it is killed.

Following termination of the download thread, a String representation of the page contents is created and subsequently written to hard disk. The page contents are downloaded using an inputStreamReader wrapped in a BufferReader, therefore avoiding costly conversions. The InputStreamReader is a bridge from byte streams to character streams, reading bytes and translating them into character encoding. The bufferedReader reads the text from the character-input stream and buffers characters to provide efficient reading of characters, arrays and lines. During the next stage of processing, the downloaded page is parsed and URLs are extracted.

2.3.3 The Parsing module

This module parses page contents and extracts relevant links, making use of the Java regular expression API library (java.util.regex). In particular a compiled representation of a regular expression, known as a Pattern is used. A Pattern is represented as a sequence of characters describing a character sequence and can be used to find matches in other character sequences. In this case the regular expression is specified using the following pattern:

```
Pattern p = Pattern.compile("http(s?)\:\/\/[a-zA-Z0-9\-\._]+(\.[a-zA-Z0-9\-
\._]+){2,}(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+=&amp;%\$#_]*)?")
```

Using the above regular expression, URLs are extracted for further processing. This expression is then compared against another sequence known as a Matcher. This engine performs match operations on a character sequence by interpreting a Pattern. The Matcher then scans the page contents looking for the next subsequence that matches the Pattern, until the end of the document is reached. When a match is found, it is returned as a string, provided certain constraints are not violated. These constraints are used to specify unwanted page types. Example pages, currently not processed by ViperBot include .gif, .jpeg, and .cgi. Currently these page types number twenty three, with new page types to be excluded, added regularly. During this stage some link processing occurs with absolute and relative URLs prefixed and anchors removed from links. All of the constraints mentioned above are in place to maximise crawling speeds and to prevent crawler crashes. All relevant links returned from the parsing module are added to the relevant URL queue, provided that the URL has not been previously processed.

2.3.4 Duplication Detection

To avoid downloading multiple copies of the same page, a module ensures each URL is only processed once. The string representation of each URL is input and the HashCode is extracted. The hash code for a String object is:

$$\sum_{0 \le i < n} s\left[i\right] 31^{i} \qquad (2.1)$$

where $s\left[i\right]$ is the ith character of the string and n is the length of the string (commonly referred to as mod 2^{32}).

This HashCode is then compared with the HashCode of previously processed URLs. If the HashCode is already contained in the HashSet, the URL is discarded and the next URL is examined. If not, the HashCode of the URL is stored and the URL is processed. A more complex alternative would involve implementing a hash function known as MD5. The MD5 algorithm takes an input of arbitrary length and produces a 128-bit fingerprint of the input. URLs may be collapsed into anything between 32 and 128 bits, based on the number of distinct URLs to be supported.

Despite this measure, duplicate pages may still be downloaded. This can occur in the case where different URLs point to the same page. Duplicate detection is critical when it is considered that 29.2% of webpages are very similar to other pages and 22.2% are virtually identical [FMN03]. To detect if the actual page is a duplicate, a number of methods may be implemented. One primitive method indicating a duplicate page is length. It is a distinct possibility that two pages are identical if they have the same number of characters. Therefore omitting a page with the same length as a page already downloaded will prevent duplication, although some non-duplicate pages of the same length may be overlooked. Other methods that could be implemented are Rabin's fingerprint algorithm [Rab81] or Shingling [BGMZ97]. In the later, Broder et al. present a syntactic technique to measure the degree of similarity between two documents, with a shingle defined as k-word sequences of adjacent words, and documents declared equal if they have the same set of shingles.

2.3.5 System Snapshot Module

Due to the distinct possibility of critical failure due to crawler hazards, a module to take a system snapshot at pre-specified time intervals is implemented. The measure prevents an entire recrawl from seed pages in the event of a crawler crash. In the current implementation, a snapshot of the system is invoked automatically and taken after every one hundred thousand pages are downloaded. With the SnapShot module, vital system statistics are written to disk using java.io.PrintWriter. In particular, the location of the next URL to be processed in each thread specific URL queue is written, along with global properties such as the number of crawled and downloaded pages. In the event of a crawler crash, this information can be accessed and used to restart the crawler from the last recorded system state. The snapshot can also be invoked through a pause button on the GUI during crawling, resulting in a system snapshot followed by an exit of the crawler program.

At startup time a previous crawl can be re-initiated from the saved system position, through the restart button. In this instance the previous system variables are uploaded from disk using java.io.FileReader. The crawler threads then read the next location from their URL queue and crawling recommences.

2.3.6 Robot Exclusion Protocol

Aside from checking if a page has already been processed, it is ethically correct for a crawler to implement the Robots Exclusion Protocol [Kos]. An ethically correct crawler will first look for a 'robots.txt' file before processing a URL on a host, not previously contacted. The crawler must then parse this file to determine pages excluded from crawling by the webmaster. Sample contents from the NUI Maynooth robots.txt file are as follows:

```
User-Agent: *
Disallow: /restricted
Disallow: /srs
Disallow: /srs2
```

In the first line, the User-Agents to which the disallow rules apply, are specified. In this case all User-Agents are specified through an asterisk, meaning all robots are restricted from accessing the areas specified in the subsequent disallow statements. The website in this instance prevents a crawler from accessing three paths.

For each host the crawler must request the ‘robot.txt’ file from the server and process it (if one exists) to determine these paths of restricted access. A drawback of adherence to this protocol can be a decrease in crawling speeds. To inhibit this potential bottleneck, our crawler stores the ‘robots.txt’ file on hard disk for each host, to prevent multiple requests for the same ‘robots.txt’ file. Server requests over a given time interval should also be capped to prevent the server becoming overloaded with requests. This can be achieved by maintaining a queue for each server and requesting pages at a maximum predefined rate. The effect of crawler traps will also be minimized as no matter how large a file is or the number of pages returned, the crawler will only retrieve pages at a maximum rate, spread across a large number of servers.

2.3.7 The URLsToCrawl module

The URLsToCrawl module has the responsibility of processing all uncrawled URLs. Once parsed, a URL is passed to the URLsToCrawl module for further processing. This module is composed of two functions. The first is concerned with writing the URL to the appropriate queue of uncrawled URLs. Each queue is maintained in an RAF to facilitate non-sequential access to file contents. The RAF guarantees that searching through one disk block costs at most two kernal calls (one seek and either one read or write). The URLs to be added and the thread number are passed to the module as input parameters. The file is then accessed, URLs are added sequentially, and a variable containing the number of URLs in the file is updated. Each URL is assigned 512 bytes and when a URL is added to the RAF, the current number of URLs multiplied by 512 is the location to seek to within the file. Each URL string character is then written to the file as a two byte value, starting from the current file pointer position and is followed by a URL termination character, symbolizing the end of the URL and therefore preventing unnecessary read operations.

When a URL is required by a thread, the associated URL queue is consulted. In this instance the 'read' method is implemented, with the thread number input and a seek is made to the current read position within the queue. The RAF 'readChar' method is then implemented, reading unicode characters from file, starting at the current file pointer. Characters will be read from the file until a maximum of 256 is reached, or until the URL termination character is reached, signifying the end of the URL. These characters are then appended to a String until the complete URL is returned.

ViperBot implements breadth-first crawling, maintaining FIFO queues with each element dequeued in the order it was added. All of these queues are currently maintained on hard disk. However, to increase crawling speeds we could maintain fixed-size enqueue and dequeue buffers in memory in a similar fashion to the Mercator crawler [HN99].

2.3.8 Creation of the DataSet and Page MetaData

Currently each retrieved page is downloaded in its entirety and stored on disk, with each an average size 20 Kb. Storage in this manner is currently possible as the dataset is comprised of only one million pages. Should a larger dataset be required, the crawler would be reconfigured to compress each file before writing it to disk. This could be achieved through the compression library 'zlib', with a typical HTML page of size 10 Kb being compressed to between 2 and 4 Kb.

At this stage page MetaData is stored to prevent the reparsing of downloaded pages “*a-postoiri*”. The MetaData facilitates the composition of a dataset web graph offline, with the information stored in a large text file on disk. Due to multiple thread access, each thread must obtain a write Lock, when appending information. When the thread holding the lock is finished writing, the lock is released and becomes available to any waiting thread. The result is a large text file consisting of the URL and associated out-links of each downloaded page. A further optimization of this module would involve replacing the read/write lock system with pipes and a multiplexer process.

2.4 The DataSet Collection

The information repository resulting from a crawl using ViperBot is a collection of 1,023,285 unique Web pages. Due to the fact that the seed pages were selected from two, previously mentioned directory websites, our crawler may be biased towards more popular pages. The total size of the dataset is 21.1 G, with each page having a size ranging from 1 byte to 1.84 Mb, and the average being 20.6 Kb.

The fact that we explicitly avoid dynamic pages means that pages in the collection are predominantly HTML. A total of 66,123 unique hosts are represented, helping to counteract the effects of localized link spamming. The total number of links present in the dataset is 12,495,560, with an average of 21 out-links and 12 in-links per page. We established the in-degree and out-degree distributions to draw comparisons with previous ranking algorithm datasets. Displayed in figure 2.2 and 2.3 are log-log plots of the out-degree and in-degree distributions. The out-degree distribution is power law, with best fit exhibited for an exponent of 2.7. Also in keeping with [BKM$^+$00], the initial segment of the distribution deviates significantly from power law, suggesting that pages with a small number of out-links follow a different distribution. The in-degree distribution also follows power law, this time with best fit exhibiting an exponent of 2.15. These dataset characteristics are in good agreement with those of previous datasets[Hav02, BM01, KRRT99].

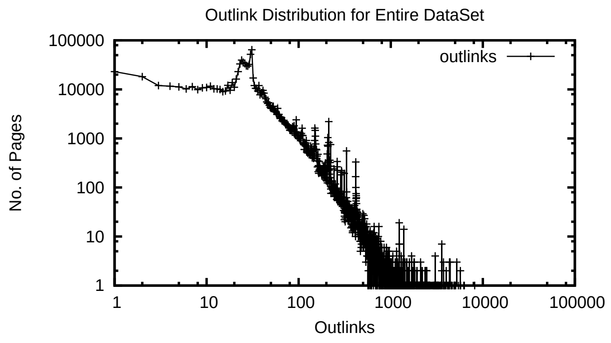

Figure 2.2: Dataset Out-link Distribution

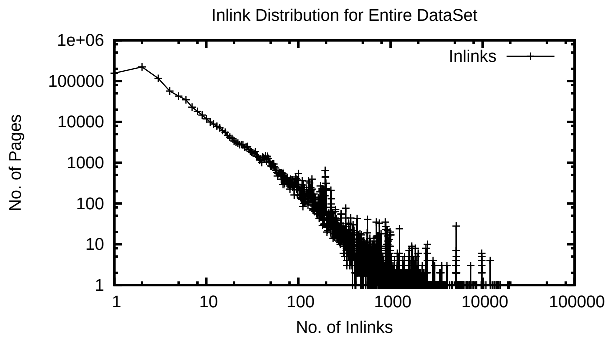

Figure 2.3: Dataset In-link Distribution

2.5 Crawler Optimization

A number of benefits are introduced through the design of ViperBot using Java. Firstly, Java's object orientated support and API class library are ideal for a modular design, the exception handling helps with the design of robust applications, memory leaks are counteracted through garbage collection, and multithreading supports the design of a multithreaded crawler. However certain drawbacks to a Java implementation exist, as identified by Allan Heydon and Marc Najork [HN00].

In many instances, the Java core libraries sacrifice speed for ease of use, with many of the core classes made thread safe by declaring their public methods to be synchronized, resulting in numerous unnecessary lock conditions. The real bottleneck for our implementation and for many Java crawlers, is host name resolution. In the case of the Mercator crawler [HN99] it accounted for 91% of each thread's elapsed time. The fact that many created objects cannot be reused is another drawback. Examples of this include FileInputStream, BufferedInputStream and RAFs. In the later the only way to access a different file is to allocate a new RAF object, with the RAF unbuffered and every read or write requiring a kernal call.

ViperBot's scalable and extensible design means it could be further optimized through numerous techniques. The storage of most ViperBot datastructures on disk reinforces this scalability and means it could easily be reconfigured to handle a crawl of increased magnitude. ViperBot is designed in a modular fashion making it extensible and therefore susceptible to new functionality.

In its current state, ViperBot is well suited to the task of downloading one million pages in less than a day. However to tune ViperBot to industrial standards, numerous upgrades are required. The initial upgrade would address the bottleneck of host-name resolution. Host-name resolution is concerned with taking the host name and resolving it into an IP address, with support provided by the Domain Name Service (DNS). Using the Java API, host name resolution is currently performed by the 'getByName' and 'getAllByName' methods of the InetAddress class. This resolution, as previously discussed, accounts for a large percentage of each thread's processing time. To overcome this, we propose to circumvent the standard name resolution system and use a DNS resolver.

Another upgrade relates to the storage of the document repository. In the current implementation, the full text version of each page is downloaded and stored on disk. One basic optimization would involve storing the downloaded files in compressed format. The ‘zlib’ library could be used to perform this compression, with expected memory saving in the region of between 60 and 80 percent. With the 1 million pages downloaded yielding a dataset of 21.1 G we suggest compression be implemented for any crawl of increased size. Furthermore we suggest the storage of our repository using a storage manager, such as the new version of the Berkley DB Java Edition. This is currently the choice of implementation for the Internet Archive’s Heritix Web crawler, and is used to archive over 50 billion URLs. Should ViperBot become a large scale crawling system the repository could be distributed over a number of storage servers.

Another area of improvement relates to duplication detection. Currently two pages identical except for one additional character may be archived in the dataset. To eliminate this problem, shingling would be implemented for the detection of near-duplicate pages. The crawler speed could also be improved through the storage of URL queue buffers within memory. Should ViperBot be upgraded to a large-scale crawling system we would also implement a load monitor to keep track of system statistics such as the number of open sockets, threads assigned to each server, and the time interval between server requests. Therefore each thread could use load monitor statistics to prevent denial of service attacks, by limiting the number of active requests to a server IP address at any given time.

Chapter 3

Representation of the Transpose Adjacency Matrix and Inverted Index

3.1 Transpose Adjacency Matrix Creation and Storage

3.1.1 Introduction

Creation of a graph representation of the document corpus is necessary in order to run link-based ranking algorithms. When deciding on graph representation two factors must be taken into consideration: speed and storage. An optimal implementation would involve creation of a two-dimensional transpose bit matrix, and storage in virtual memory. However, due to memory constraints, this is not possible and other techniques are considered.

The corpus is composed of over one million nodes, each representing a URL, and over 60 million directed edges representing hyperlinks. Considering that on average, a

URL consists of between 50 [SY01] and 80 [BBH$^+$98] bytes, the extent of the memory requirement becomes apparent. As previously discussed, a dataset graph is composed of nodes and edges and is represented through a square adjacency matrix A. If a page j links to page i then $A_{ij} = 1$, otherwise $A_{ij} = 0$. The result is essentially a very sparse bit matrix, creating two problems. The first regards the adjacency matrix size. The memory requirement is of the order $\mathrm{O}\big(n^2\big)$, with n defined as the number of dataset URLs. This results in a memory requirement of a terabit of storage space and is too excessive to be considered. The second relates to the method adopted for the storage of the square adjacency matrix and has a direct bearing on the efficiency of the retrieval of graph properties and the convergence speeds of link analysis ranking algorithms. Alternately, we create and store the adjacency matrix transpose.

With an adjacency matrix transpose representation, every row represents page in-links as opposed to out-links. A sparse representation of the transpose matrix is stored instead of an $n * n$ bit dataset representation, with each row containing an arbitrary amount of page in-link numbers, instead of a fixed amount of binary digits.

3.1.2 Creation of the Transpose Matrix

The steps involved in converting the connectivity information into the transpose matrix are highlighted in this section. Initially, a brief outline of the contents of the connectivity-information file is provided. Secondly a preprocessing stage, enabling fast URL lookup during the creation of the transpose matrix is highlighted. The prevention of multiple-identical link re-processing is also discussed, along with the problem of dangling-link detection.

The Connectivity-Information File

Creation of the Transpose Matrix involves processing the linkage information compiled during a crawl of the Internet. During the crawl, linkage information is written to a large text file named “Connectivity.txt”. The file is composed of the URL of each page and the URL representation of each out-link. The URLs are not written to the file in sequential order and as a result each page number is stored along with the number of out-links, to provide fast out-link number lookup for any page, during the propagation of a link analysis ranking algorithm.

The Detection of Url Presence

With the corpus currently containing over 1 million URLs, URL lookup can become a severe bottleneck during the creation of the transpose matrix. As a result, during a preprocessing stage all of the downloaded page URLs are extracted and stored in virtual memory. Each of the URLs is stored in one of one thousand HashMaps. The HashMap to which each URL is added is determined based on the numeric value of the last three digits of the HashCode. Therefore if the last three digits are 000 the URL will be added to the first HashMap, if the digits are 999 the URL will be added to the 1000th HashMap, with all intermediate digits used to reference the HashMaps sequentially from the second to the 999th.

Prevention of Multiple Processing of the same Out-link

Creation of the transpose matrix begins with extraction of the page number of each URL, due to the un-sequential nature of URLs in the file. Next the out-links associated with each downloaded URL are processed (if any exist), with the URL string representation of each out-link extracted. The URL is then added to a temporary ArrayList, to prevent identical out-links being processed for a page and subsequently to prevent a page having two or more inlinks from the same page. If the link is already in the ArrayList, it is not processed further and the next link is examined, otherwise the link is added to the ArrayList and becomes available for further processing.

Dangling-Link Detection

Before an out-link is used to update the transpose matrix, another check is made to establish if the page pointed to is contained in the corpus. If this is not the case, the link is dangling and is discarded. This is determined by querying the URLs present in the set of HashMaps. The HashCode of each out-link is computed with the last three digits used to determine the HashMap to query. The Java API HashMap function ‘get()’ is then called, with the HashCode used as the URL key. The return of a null object indicates that the link is dangling and results in the discarding of the link. Should a non-null object be returned, page presence within the corpus is indicated and the transpose matrix is updated accordingly.

3.1.3 Storage of the Transpose Matrix

In this section we discuss how the transpose matrix is written to memory. Initially we look at the structure used for storage of the transpose matrix and the reason for its selection. The steps involved in populating the transpose matrix are discussed next, including how initial markers are written to the RAF, how inlink integer representations are added and finally a discussion is made on the use of overflow RAFs.

Storage Structure

Storage of the Transpose matrix is designated to an RAF, facilitating non-sequential access and as previously mentioned guaranteeing that searching through one disk block costs at most two kernel calls (one seek and either one read or write). A fixed number of bytes is assigned to each page for the storage of its in-links within the RAF. The figure is currently set to four thousand bytes facilitating the storage of one thousand in-links per page. This restriction is implemented due to the fact that 74 % of pages have less than one thousand inlinks, and the provision of storage space based on the maximum number of inlinks received by any page (in excess of 12000), would lead to a considerable waste of storage space. As an alternative, each page could be assigned an individual RAF to contain its inlinks exclusively, greatly reducing storage requirements although to the detriment of link analysis ranking algorithm implementation speeds. In the event of the number of inlinks associated with a page surpassing one thousand, an overflow RAF is created to store the additional inlinks for that page.

Creation of the Transpose Matrix

Creation of the main RAF begins with a seek to the location marking the beginning of the in-links for each page. Each location is determined by multiplying the page number by 4000 and writing a -1. This integer is used to signify the end of in-links for each page and is moved every time a new in-link is added for a page, with a -1 written immediately after the rightmost in-link.

To write an in-link number to the transpose matrix, firstly a seek is made to the beginning of the byte allocation for that page by multiplying the page number by 4000. Integers are then read until a -1 or a -2 is processed. When a -1 is encountered, the -1 is overwritten by the in-link number and proceeded by the -1. In the later case, -2 symbolizes the end of space allocated for the pages in-links and the in-link number is written to the overflow RAF.

OverFlow Storage

An overflow RAF is created for every page with in excess of one thousand in-links. When a -2 is encountered in the main RAF, signifying the end of the byte allocation for that page, an RAF is created for that page, if the file does not already exist. Therefore a check is initially made to see if the file is empty and if this is the case an integer 2 is initially written, followed by the integer representation of the in-link. The integer 2 represents the next write location within the file. When the file is re-accessed, if the file is not empty this integer is read, used to calculate the seek location, and incremented.

The in-link number is then written as a four byte integer (with the high byte written first) using the RAF function ‘writeInt()’ and the connection to the random access file is closed.

3.1.4 Previous Work

Graph representation for a large corpus relies on compression techniques. However decreased storage space often leads to an increase in data retrieval and therefore currently available compression techniques need to be closely scrutinized.

URLs and link structure need to be compressed separately and are therefore two separate problems. K. Bharat et al. [BBH$^+$98] suggest that the average URL is approximately 80 bytes on average and that storage of 100 million URLs is reduced by 70% using delta-encoded text files, with each URL stored as the difference between the current and previous URL.

In relation to link compression, locality and similarity are two features generally exploited. The majority of links on a page are navigational in nature, leading to other pages within the same host, with T. Suel and J. Yuan [SY01] suggesting that three quarters of links on a page, point to pages on the same host. With regards to similarity many pages occurring close to each other tend to share numerous common out-links, with many navigational links the same across numerous host pages, often copied from one page to another. It may also be observed that many links within a page are consecutive in terms of lexicographical order, due to the fact the navigational links on most pages point to a fixed-level hierarchy. Therefore they share a long common prefix which can be exploited. These host structures may also be identified within the global dataset and compressed individually, and blocks within the matrix graph representation may also be stored individually, to decrease storage requirements.

K. Randall et al. [RSWW01] exploit the fact that most links on a page point to other pages on the same host, suggesting that 80 % of links fall into this category.

Also exposed is the fact that many pages on the same host share many of the same hyperlinks. For each adjacency list, it is suggested that the delta gap between URLs is stored, and also introduced the idea of partitioning adjacency lists into groups based on the number of links they contain, to compress pointers to them. Using these technique each link is reduced to roughly 6 bits.

T. Suel and J. Yuan [SY01], store global links (non inter-host links) using Huffman coding for the majority of more popular destinations and Golub coding for the rest. For local links (inter-host links) Huffman coding is again implemented for the more popular, with delta coding introduced for the rest. Each link is reportedly reduced to an average of roughly 14 bits. Suggested also is that the average URL consists of about 50 characters (compared to the 80 suggested by K. Bharet et al.) in uncompressed format and can be compressed to about 13 bytes by exploiting the common prefixes in the sorted list of URLs.

P. Boldi and S. Vignain [BV03a] suggest gaps of web graphs are distributed as power law with an exponent ranging from 1.1 to 1.3 and therefore that codes coming from full text indexing, such as Golumb codes are not useful, as they are based on exponentially decaying distributions. Other dataset graph compression techniques are discussed by P. Boldi and S. Vignain in [BV03b], with inner redundancies of the web exploited, enabling graph storage in a limited space in memory. The compression techniques evolve around referentation and intervalisation, compressing a transpose graph of 118 million nodes to 2.89 bits per link. Suggested are algorithms exploiting gap compression, referentiation, intervalisation and algorithms for accessing a compressed graph without the need for re-compression, resulting in an ideal workbench for web graph manipulation.

3.1.5 Optimization of Technique Adopted

Corpus connectivity information is represented using a sparse transpose matrix, stored in a main RAF with additional overflow files. The structure is adopted to increase algorithmic implementation speeds by storing the vast majority of page inlinks in a single file, although consequently at the cost of additional memory. This implementation is sufficient in our case with the transpose matrix requiring 4 G, a long way from approaching the storage capacity of the PC. In the event of an increase in the size of the dataset, each page's inlinks could be stored in a single RAF utilizing compression techniques. It is suggested that the transpose of the graph is often more "entropic" and therefore more difficult to compress than the graph itself [Kle99], although this is dismissed by P. Boldi and S. Vignain where the transpose is stored using less bits per link than when the original graph matrix is compressed.

Therefore optimization would involve storage of the gap between in-links and adoption of compression techniques, such as delta encoding or those suggested by P. Boldi and S. Vignain. Although this would lead to an increase in the computation time required by the link analysis ranking algorithms, storage requirements would be reduced considerably and facilitate the storage of a web graph comprised from many tens of millions of pages.

3.2 Inverted Index Creation and Storage

3.2.1 Introduction

To facilitate efficient user-query processing a search engine maintains an inverted index file. An inverted index is a datastructure mapping a word to the set of documents containing the word, and is sometimes referred to as a postings file. Creation of an inverted index requires the presence of a list of all words appearing in the corpus, known as a lexicon.

A few important factors should be taken into consideration when creating an inverted index. These include, the time taken to index the documents, to access the inverted index, to access the postings for a particular word and the index updated rate. Of importance also, is the amount of storage memory required, where the inverted index size should be proportional to the number of postings [CP90], with the indexing overhead defined as the index size expressed as a percentage of the entire dataset.

A typical inverted index is composed of the word document identification, the number of word occurrences and the position of the word within the document, for each word contained in the lexicon. The word position may, however, be excluded from the inverted index, should word proximity not be considered when processing a query. Storage of word to document IDs can be provided by a storage manager such as the Berkley DB storage manager and is ideal for a dynamic index, where documents are constantly added, removed, and deleted. For the static collection used in our experimentation, we implement a custom made datastructure, providing the potential for more space-efficient implementations.

3.2.2 Creation of the Inverted Index

Once downloaded from the Internet, the hypertext corpus requires further processing before a user query can be addressed. Corpus documents are parsed using a HTML parser, extracting words and the number of word occurrences. These extracted words exclude links, HTML tags or meta-data of any kind. Essentially only text visible when viewing the document with a browser is extracted. A word file is created for every word in the lexicon, containing document IDs and word occurrences, with the necessary random access provided through a Java RAF. Once a document is parsed, the number of word occurrences and document ID are written to a word file.

When deciding on what document content to index, case folding, stemming and stop words should be considered. Case folding is concerned with the conversion of characters to their lowercase equivalent. Instead of creating an individual posting for three strings such as ‘car’, ‘CAR’ and ‘Car’, the strings may be converted to their lowercase equivalent, with all document IDs stored in a single postings file for the word ‘car’.

In the case of stemming [Por], every word is stripped to the root and stored accordingly. For example the words ‘acceptance’ and ‘accepting’ are stripped to the root and stored using ‘accept’. The storage of the indexed words, therefore requires less space, although some accuracy is removed from the query matching process and is therefore not appropriate for every document collection. However a combination of case folding and stemming can be used to reduce the number of documents indexed by 40 percent [WMB99], with stop word removal another technique capable of reducing index size. A stop word is a word occurring frequently within most documents, with its removal having minimal impact. The set of stop words is referred to as a stop list with an example stop list given below:

the, and, it, was, of, for which, be, a, are, have, or, its, but, be, on

The removal of stop words from the inverted index, can result in a space saving of 25 percent, for large document collections [WMB99]. However stop list words are often the shortest words in the lexicon, requiring the least amount of storage space and as a result, the benefits of a stop list may not be as great as they appear initially.

3.2.3 Storage of the Inverted Index

For each word in the corpus, a list of document IDs and word occurrences are stored within a RAF, containing an arbitrary number of integers. An initial integer is used to specify the number of document IDs present and therefore is used to specify the seek position when writing to file. For every document the word appears in, two four byte integers are written to the file, representing the document ID and number of word occurrences. A sample corpus consisting of three documents and the resulting inverted-index is displayed in figure 3.1.

d1: The fat cat

d2: The fat cat slept

d3: The cat is on the mat

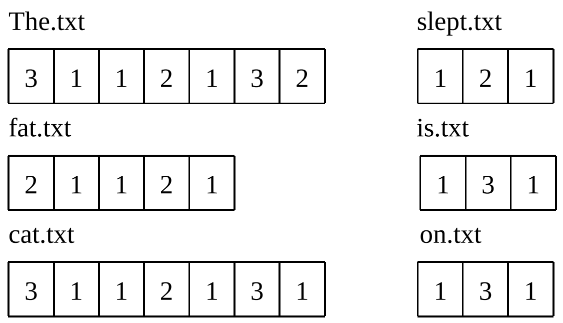

Figure 3.1: Example of Inverted Index Creation

The corpus contains the three documents d1, d2 and d3. The three documents are parsed sequentially, with the list of integers displayed in figure 3.1 written to each word file. For the word file "The.txt" the integer values are explained. The initial integer specifies the number of documents in which the term appears. In this instance the word is present in three documents and therefore the initial integer is a three. Each of the document IDs are then written followed by the number of times the word appeared in the document. Looking at the second integer we can see that the word appeared in document one and by looking at the third we see that it occurred once. The remaining document IDs and word occurrences are written to "The.txt" in the same manner, with the procedure repeated for the five remaining word files.

When writing to a word file initially the first integer is read and used as the location to seek to within the file. This value is calculated as $(x * 8) + 4$, with x defined as the initial integer . Therefore it is the number of documents represented in the file multiplied by 8, taking into account that 2 four byte integers will be added during each write with the final four taking into account the initial integer. Once the initial integer sets the pointer position it is incremented and rewritten to file. The document ID and word occurrences are then appended to the file and the file connection is closed. Following the parsing and processing of every corpus document we are left with an inverted index where every word in the lexicon is represented through a word file. Therefore if a query contains a word in the index, the word file is simply accessed and every document present can be assigned a weighting, based on the number of word occurrences within the text.

3.2.4 Further Inverted Index Refinements

Numerous inverted-index compression techniques are currently available, with it is suggested that compression techniques can reduce an index size to between 10 and 15 percent of the uncompressed index [SWYZ02].

A large portion of the inverted index is composed of document IDs and can be exploited though a delta coding compression technique. Document IDs are sorted in increasing order, storing the first document ID in full and subsequently only the difference from the previous ID, called the gap. This method is simplistic, efficient, and best described through an example. Consider a word *w* appearing in the following 10 documents numbered $7, 11, 20, 29, 36, 51, 71, 77, 82, 96$. The word *w* will be represented through the following list:

$\langle 10; 7, 11, 20, 29, 36, 51, 71, 77, 82, 96\rangle$

Therefore the list for word *w* contains the number of documents *dn* in which the term appears (10), followed by the document IDs. As the list is in ascending order, it can be compressed by storing the initial complete document ID, followed by a list of gaps. The list above therefore translates to the following:

$\langle 10; 7, 4, 9, 9, 7, 15, 20, 6, 5, 14 \rangle$

As a result, the original document IDs can always be retrieved by obtaining the cumulative sum of the gaps and allows the storage of word lists using substantially fewer bits on average. Many models are available to describe the probability distribution of d-gap sizes. One method available is Unary Coding and involves coding an integer $x \geq 1$, as $x - 1$ one bits followed by a zero. For example the integer 4 would be coded as 1110. Unary coding favors short gaps and is equivalent to assigning a probability of $Pr\left[x\right] = 2^{-x}$ to gaps of length x. This is excessively small and may not be suitable for the compression of an inverted index.

Another efficient compression technique available is Golumb Coding. Related to Unary Coding, exponentially decreasing probabilities are assigned with the exponential decay often close to one. Golumb coding is more efficient when documents not containing the word are stored, rather than those that do. An inverted-index compression technique based on Golumb coding is therefore suggested as the ideal implementation for most large-scale document corpora.

Chapter 4

Link Analysis Ranking Algorithms

4.1 PageRank

4.1.1 Introduction

L. Page and S. Brin [PBMW98] state that “The original goal of PageRank was to sort backlinks so if there were a large number of backlinks for a document the ‘best’ backlinks could be displayed first”. PageRank is currently the major influence behind the Google search engine and is an extension of simple citation counting. Citation counting simply ranks pages based on the their number of in-links. However this ranking technique is susceptible to link spamming and can overlook important pages with only a few in-links. Through the necessity to overcome this limitation, PageRank emerged from social network analysis into an iterative ranking process over the entire web graph. The hypothesis underlying the PageRank algorithm is that a page with a large number of in-links or a link from an important page should be deemed important. Using the

PageRank paradigm, if a page A links to page B the importance conferred on B by A, is proportional to the importance of A and inversely proportional to the number of pages pointed to by A.

PageRank Algorithm:

$$\mathit{Pr}\left(i\right) = \left(1 - D\right) + D \sum_{j \in A_i} \mathit{Pr}\left(j\right) / \mathit{out\text{-}links}\left(j\right) \tag{4.1}$$

The PageRank (Pr) of a page i, is the sum of the PageRank (Pr) of its in-links A_i, divided by the number of corresponding out-links. D is the damping factor (typically 0.85) and ensures every page receives a minimum ranking of $(1 - D)$.

The PageRank algorithm can be described using the ‘random surfer’ analogy, where the behaviour of a archetypical person browsing the web is modeled. Starting from a random page, the surfer will choose to follow one of the page out-links with equal probability, occasionally getting bored and jumping to another page completely at random. The algorithm propagates index linkage information, represented in a non-negative adjacency matrix with row sums equal to one or zero, often called the Google matrix. The resulting PageRank vector is used to optimize the layout of the inverted-index structure accordingly and while PageRank is query independent and computed offline, it is typically aggregated with other content-based scores.

4.1.2 PageRank Implementation

One important issue to address before implementing PageRank, is dangling-node management. A dangling node occurs when a page is linked to, but has no out-link and a dangling link refers to a link pointing to one of these pages. Considering that no search engine indexes more than approximately 16% of the web [LG00], it is not surprising to find that many links on a page are dangling, and can be due to a number of reasons. Possible reasons for this is that there may be restricted access to the page or due to the majority of PDF and Postscript pages having no embedded links, to name but a couple.

In most datasets the number of dangling links greatly outnumber the crawled links, with numerous methods available to counterbalance their effect [EMT04]. Simple removal of the dangling links will skew the results, as the number of out-links associated with non-dangling links will have to be adjusted accordingly. Instead of completely removing these pages, one can simulate a random-surfer jump to a random node with probability 1, by adding a link from the dangling page to every other page, or remove the dangling links and reinsert them for the last few iterations [KHMG03, Hav99, BMPW98].

Once the dangling links issue is resolved PageRank computations begin with an iterative process over the non-negative square adjacency matrix A. Initially every page is assigned a PageRank value of 1 or 1 divided by the total number of pages in the dataset, with initial values not affecting the final values and only affecting the convergence rate [GL83]. If a page j links to page i then $A_{ij} = 1/N_j$, otherwise $A_{ij} = 0$. The PageRank vector is computed by repeatedly multiplying matrix A, with the vector of the current estimate of page importance until the estimate is stable, and is equivalent to extracting the principal eigenvector from the Markov Matrix A.

The PageRank of a page is therefore acquired through an iterative fixed point computation, with the Power method [FKS03] used to perform the computation in most instances. Using the Power Method, the number of iterations required to achieve convergence grows with the damping factor, requiring increased numerical precision as D gets closer to 1.

Single precision computations are sufficient for the majority of PageRank implementations, but across an index on the scale of modern-day search engines (i.e. Google, Yahoo and MSN), double precision becomes necessary [EMT04]. The PageRank computation terminates in logarithmic time (time logarithmic in the size of the graph [**?**]) with potential for the reduction of total computation times available through improved I/O management [Hav99, CGS].

4.1.3 Damping Factor

The web is modeled using a directed graph and the assumption is that it is strongly connected, meaning every page can be reached by following links from any other page. In reality this is not the case and as a result a damping factor D is incorporated in the PageRank equation. In terms of the random surfer analogy, this can be viewed as the occasion when the surfer gets bored and jumps to a page within the dataset completely at random. The damping factor is described by L. Page and S. Brin [PBMW98] as "a vector over the Web pages which is used as a source of rank to make up for the rank sinks such as cycles with no outedges". This rank sink effect is exhibited during iteration when PageRank gets concentrated in recurrent states, namely loops of pages accumulating rank but never distributing any. The value selected for D can be any value in the range $0 \leq D \leq 1$, although as the damping factor goes to 1, the rank of all important nodes goes to 0. A value of 0.85 suggested by L. Page and S. Brin is more commonly used, striking a balance between achieving rapid convergence with minimal perturbation to the rankings.

The damping factor limits the effects of rank sink, ensuring the web graph is strongly connected and consequently guarantees convergence. As the damping factor corresponds to the distribution of web pages jumped to periodically by a web surfer, it results in the assignment of a PageRank value of $(1 - D)$ uniformly to all pages, simply because they exist. The rank is passed from each page through the out-links, with one common form of link spamming involving the passing of this rank to a single page, therefore leading to an inflated PageRank value. The damping factor can be used to to increase or decrease a site's PageRank value, perhaps to punish a suspected link-farmer or to reward a favored client. Instead of the random surfer occasionally jumping to a page completely at random, personalization PageRank may be implemented. In this case, using the ‘random surfer’ analogy the jump may be to a particular page such as an individual’s home page or with equal probability to a page from a set of topic-related pages [BMPW98, Hav02].

4.1.4 Convergence

PageRank convergence is guaranteed provided the web graph is irreducible and aperiodic [Hav99], with the latter guaranteed for the web and the former true with the introduction of the previously discussed damping factor.

Numerous factors influence the rate of convergence, not least of which are the damping factor and the matrix size. An increase in damping value decreases the convergence rate with a low damping value providing the opportunity for spam pages to accumulate an excessive PageRank value. The rate of convergence can be measured through examination of the PageRank-induced orderings, using the Kendall tau distance [Ram98], or through the more traditional L1 norm method [KHG03, KHMG03, KHMG].

The rate of convergence and overall implementation times are critical for PageRank implementations on web datasets where regular updates are required, due to the inherent dynamic nature of the web. Using the Power Method the rate of convergence for individual pages is nonuniform, with many pages converging quickly and a few pages taking much longer to converge. On a dataset of $280,000$ nodes and 3 million links, Sepandar D. Kamvar et al. [KHG03] observe that the majority of pages converge within 15 iterations, with a few (those with higher PageRank values) requiring over 40 iterations and exploit this observation to decrease the PageRank computational overhead by up to 30%.

Using Quadratic Extrapolation, Sepandar D. Kamvar et al. [KHMG03] again speed up PageRank computations by between 25 and 300% and introduce their BlockRank algorithm in [KHMG], to exploit host structures. A local PageRank vector is computed for each host, giving the relative importance of pages within a host, and is then used to provide an approximation of the standard PageRank vector, with most hosts converging to an L1 residual < 10^{-1} in less than 12 iterations. No correlation between the convergence of a host and the host's size is observed and hosts with strong nested block structures appear slower to converge, when compared to those with a more random connectivity pattern.

4.1.5 PageRank Merits and Demerits

One of the main benefits of PageRank is its resistance to link spamming. This is due to the fact that it is very difficult for a web page author to obtain a link from an important page, unless the page contains information of interest to the important page author. Through the introduction of link farms and reciprocal link programs, PageRank is not immune to spamming. The query independent nature of PageRank however, combined with the fact that the PageRank vector is computed over the entire web graph makes the algorithm less susceptible to localized link spam than other link analysis ranking algorithms such as HITS.

Another benefit of the PageRank algorithm is that the damping factor is not fixed and may be used to bias the algorithm to a particular page or pages related to a particular topic (Personalization PageRank). For example, as previously mentioned, the damping factor may be set to a user home page or bookmark list to bias the algorithm towards pages of a similar topic [Hav02, JW03]. However the drawback of such an implementation is the computation time involved in generating the multiple PageRank vectors, that correspond to various teleportation vectors for different topics [KHMG03].

Overall, the main weakness of the PageRank algorithm is its inability to differentiate between pages that are authoritive in general and pages that are authoritive in relation to a particular query. As a result, a page with a high PageRank value that contains a query word, but is not relevant to the user query, may be returned in a prominent result-set position. This is due to the fact that the PageRank algorithm is query independent and therefore not influenced by the user query.

For example consider a hypothetical situation involving two web pages and the query word ‘Jaguar’. Examination of the page contents reveals that both the Yahoo homepage and Jaguar.com contain the query word. In this case the Yahoo homepage occupies a more central position in the web graph than the Jaguar homepage, and will therefore be returned in a more prominent result-set position for the query word ‘Jaguar’. From figure 4.1 we can see that the Yahoo site has over twice as many links as the Jaguar site and is more likely to obtain a higher PageRank value. This is due to the fact that PageRank is assigned based on global importance, rather than local importance and as a result the Yahoo site will obtain a higher PageRank value. It is suggested, however that the Jaguar page is more relevant to the query due to the fact that it’s inlinks are all topic related. In an attempt to reduce this effect the PageRank score is usually combined with a content score (and additional scores) at runtime.

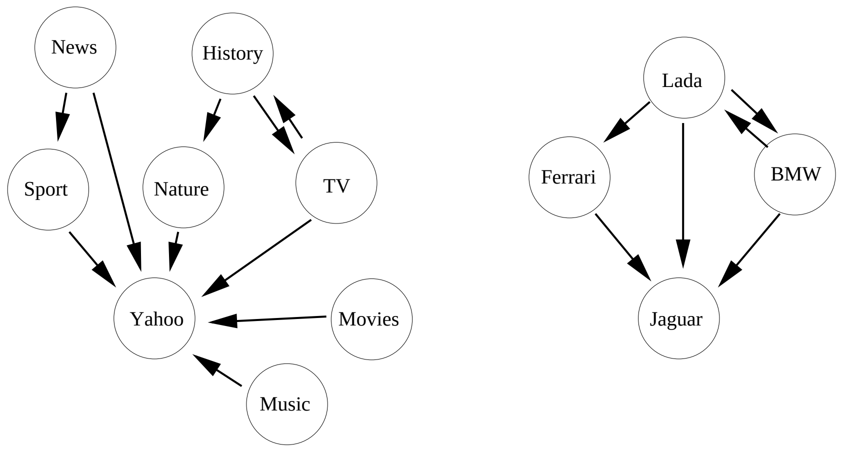

Figure 4.1: Example of Global Importance Vs. Local Importance

No bias towards topic-related links exists for the PageRank algorithm and it is suggested that a link from a topic-related page should carry more weight. This intuition is based on the idea that a link to a page from one of its peer authors, should be viewed as more important than a link from an author whose page is unrelated to the topic of the page in question. This is the fundamental principal behind the QD-PageRank and subsequently the EQD-Rank algorithm.

4.1.6 PageRank Example

To demonstrate the computations involved in a PageRank implementation, we calculate the PageRank vector for the web graph consisting of twelve nodes and twenty eight links, displayed below:

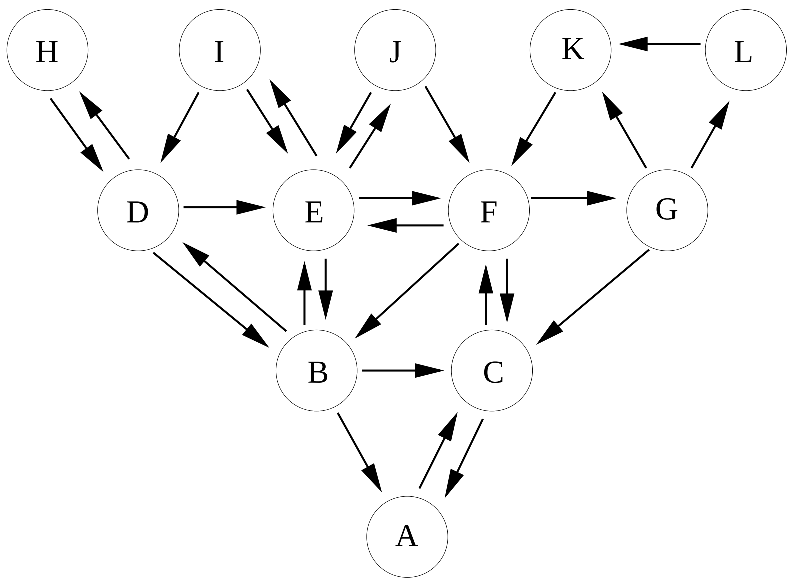

Figure 4.2: Sample Dataset Graph for PageRank

The square adjacency matrix A is thus created, based on the connectivity information displayed in figure 4.2. Row 0 of the matrix corresponds to the page A, row

11 corresponds to page L, with all intermediate rows representing the pages from B to K sequentially. Instead of creating a sparse bit matrix, with each row entry (A_{ij}) represented by either a 0 or 1 divided by the number of out-links, only out-links are represented, divided by the total number of out-links.

$$A = \begin{bmatrix} 2 & & & \\ 0/3 & 2/3 & 3/3 & 4/3 \\ 0/3 & 5 & & \\ 1/3 & 4/3 & 7/3 & \\ 1/4 & 5/4 & 8/4 & 9/4 \\ 1/4 & 2/4 & 4/4 & 6/4 \\ 2/3 & 11/3 & 12/3 & \\ 3 & & & \\ 3/2 & 4/2 & & \\ 4/2 & 5/2 & & \\ 5 & & & \\ 10 & & & \end{bmatrix}$$

From the matrix A above we can see that the out-links for each page are represented in rows 0 to 11, divided by the total number of out-links for that row. Taking page E as an example we can see that its four out-links, pages B, F, I, and J are all represented in row four, divided by the total number of out-links associated with page E. In its current state matrix A is stochastic, due to the fact that each node has at least one out-link. However, consider a situation where a page with no out-links is introduced and represented by row 13. This would result in A becoming non-stochastic and would result in the necessity for every zero entry in row 13 being replaced with $1/13$. Matrix A is however reducible and results in the introduction of the damping factor D. As a result, matrix A is now stochastic and irreducible and following iteration of the PageRank algorithm (equation 4.1), we obtain the following PageRank vector:

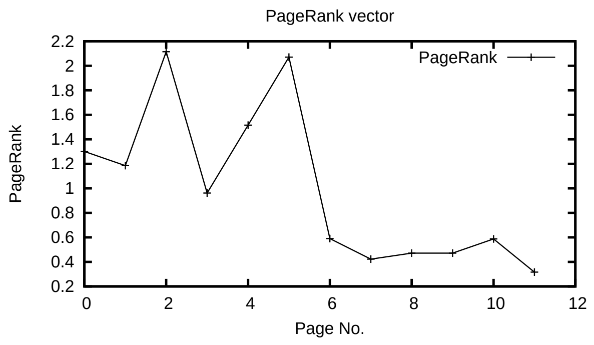

Figure 4.3: PageRank Vector

The result is a PageRank vector with values ranging from a maximum of 2.07 to a minimun of 0.32. In this case page C received the maximum value with pages F and E next in order. This can be attributed to their central location within the web graph. Page L receives the lowest ranking and can be attributed to the solitary link from 'unimportant' page G.

4.2 Query Dependent PageRank

The fundamental premise underlying link analysis ranking algorithms is that a link from page A to page B, is a recommendation of page B, by the author of A. Therefore should page A be considered query relevant, intuitively the link from page A should be weighted more heavily than a link from a query-irrelevant page. A query-dependent ranking algorithm can focus specifically on the topic community and disregard irrelevant links. As a result, a page deemed important within this topic community and thus by its peers, is more relevant than a page deemed important across the entire web. Consequently, M. Richardson and P. Domingos [RD02] introduce the user query to the previously query independent PageRank algorithm. The hypothesis underlying Query Dependent PageRank (QD-PageRank) is that a page with numerous in-links or a link from an important page within the topic community, is more important than a page with numerous in-links or a link from an important page within the entire web. The QD-PageRank algorithm is therefore an augmentation of standard PageRank.

QD-PageRank algorithm:

$$\textit{QD-PR}\,(i) = (1 - D) + D \sum_{j \in A_i} \textit{QD-PR}\,(j)\,/\textit{Outlinks}\,(j) \tag{4.2}$$

The QD-PageRank (QD-PR) of page i is the sum of the Query Dependent PageRank (QD-PR) of its in-links A_i, **where both i and j are query relevant**, divided by the number of their corresponding out-links. The damping factor D is typically 0.85 ensuring every query-related page receives a minimum ranking of 0.15.

Using the 'random surfer' analogy, when the random surfer is choosing from multiple page out-links, only those deemed query relevant are considered. When the surfer makes a random jump, it is random only within the set of query-relevant pages. Query irrelevant pages therefore exert no influence over the final ranking values and are assigned a ranking of zero.

4.2.1 Implementation

A preprocessing stage to determine page relevance is required before QD-PageRank can be implemented. Every page in the index is assigned a relevance score based on the user query, with numerous ranking metrics available to determine this relevance (e.g. TFIDF [SL68], Latent Semantic Indexing [DFL+88], or Probabilistic Latent Semantic Indexing [Hof99]. Pages with a relevance score above some predetermined threshold are included in the subgraph, through which the QD-PageRank algorithm will iterate. Every relevant page is assigned an initial QD-PageRank value of 1, with all other pages assigned a value of 0. Therefore in the square adjacency matrix A representation of the subgraph, if a page j links to page i, and both are relevant then $Aij = 1$, otherwise

$Aij = 0$.

The issue of dangling nodes must be addressed for the subgraph, with a link from the dangling node to every other node introduced. Also to ensure that the matrix is irreducible and to prevent rank sink, the damping factor is applied. The QD-PageRank values are computed next by simulating the repeated multiplication of the adjacency matrix for the query-related subgraph, with the vector of the current estimate of page importance, until the estimate is stable and is equivalent to extracting the principal eigenvector from matrix A.

Each QD-PageRank value is computed through an iterative fixed point computation using the Power method [GL83], with the number of iterations required to achieve convergence dependent on query broadness and subsequently the size of the subgraph. Single-precision computations are sufficient for the computation of the QD-PageRank vector for all words in our lexicon. The computations terminate in logarithmic time (time logarithmic in the size of the graph) with potential for the reduction of computation times available through various algorithmic speed-up techniques [KHG03, KHMG03, KHMG, Hav99].

4.2.2 Strengths & Weaknesses

The primary benefit of QD-PageRank is that pages that are irrelevant but authoritative in general no longer dominate. However, as typical search engine users are prepared to wait at most between 10 and 20 seconds, calculation of the ranking values for pages in the query-related subgraph is not feasible within this time bound. M. Richardson and P. Domingos suggest the computation and storage of QD-PageRank values "*a-priori*", with computation and storage requirements stated as requiring approximately 100 times that of standard PageRank. Despite the somewhat conservative nature of these figures, the infinite number of possible words across the different languages means QD-PageRank cannot be implemented by modern search engines. One possible method to address this shortcoming is the computation of QD-PageRank values for a limited number of popular queries, in a similar manner adopted for the Hilltop algorithm [BM01].

When the user query is very specific the resulting topic community may be sparse and the ranking algorithm becomes less effective. For such a sparse community the algorithm will be highly susceptible to localized link spam, resulting in a preference for a standard PageRank implementation.

The main drawback of a QD-PageRank implementation is the computation time. These times, however are minimized through the fact that only query-relevant pages need to be considered and results in a decrease in the QD-PageRank computations. The number of query relevant pages can be reduced through the constraint that if the word does not appear in the document, the document is automatically deemed irrelevant and through a threshold on the content-based retrieval score.

The overall computation times are proportional to the number of documents in the subgraph. The overall time will require, a factor of S/N times the computation of the query independent PageRank vector, with S defined as the number of relevant pages and N defined as the number of corpus pages. On average the QD-Rank computations require $0.75 * S/N$, with potential for further improvements arising through the ability of some of the smaller sub graphs to now fit into memory, eliminating costly disk seeks.

4.2.3 QD-PageRank Example

To demonstrate the computations involved in a QD-PageRank implementation, the QD-PageRank vector is computed for the web graph consisting of twelve nodes and twenty eight links, displayed below:

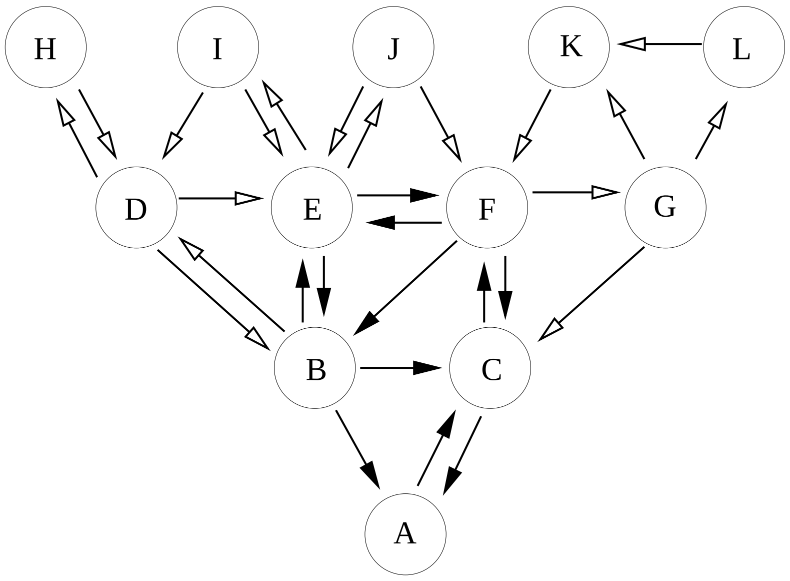

Figure 4.4: Sample Dataset Graph for QD-PageRank

This is the same as the web graph from the PageRank example (figure 4.2) with the addition of node-relevance classification. In figure 4.4 nodes A, B, C, E, and F are deemed relevant, with interconnectivity displayed using black arrows. All other links are extraneous, therefore excluded, and represented using white arrows. Once the relevant nodes and links are identified, the next stage involves the creation of a square adjacency matrix for the sub graph. The matrix A therefore consists of five rows:

$$A = \begin{bmatrix} 2 & & \\ 0/3 & 2/3 & 3/3 \\ 0/2 & 4/2 & \\ 1/2 & 4/2 & \\ 1/3 & 2/3 & 3/3 \end{bmatrix}$$

From matrix A we can see that each row displays the out-links associated with each query-relevant page divided by the number of out-links. Again for this example, if a hypothetical sixth row existed containing no out-links, every zero entry in the row would be replaced by $1/6$. As with the standard PageRank example the damping factor of $D = 0.85$ is introduced, eliminating reducibility. The matrix A is stochastic and irreducible and QD-Pagerank algorithm (equation 4.2) implementation results in the following QD-PageRank vector:

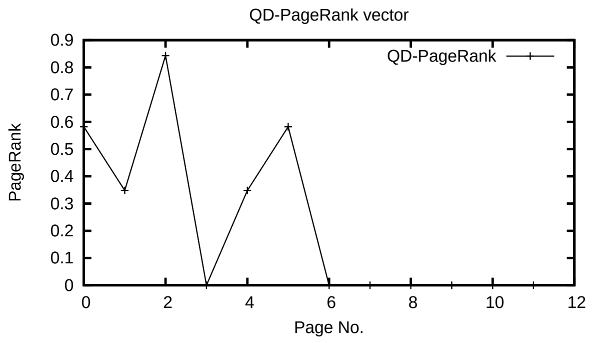

Figure 4.5: QD-PageRank Vector

The result is a QD-PageRank distribution ranging from a maximum rank value of 0.84 to a minimum value of 0.348. In this case only five pages are deemed relevant to the query and as a result all other seven pages have a rank value of zero. Pages B and E obtain the minimum ranking value in this instance, with page C obtaining the maximum value, attributed to the fact that three out of it's original four in-links are query relevant.

4.3 Estimated Query Dependent PageRank

4.3.1 Introduction

The problem with QD-PageRank is its inability to be performed at runtime, due to the necessity of creation of the query-related subgraph. To return an estimation of the QD-PageRank vector and a similar rank ordering, we introduce a result-set post-ranking refinement, based on local graph-traversal. EQD-Rank avoids the creation of the query-related subgraph through a manipulation of the PageRank values. The difference between QD-PageRank and EQD-Rank is that the ranking vector is created at runtime instead of "*a-priori*", with experimentation revealing an average relevance increase of 21.26%.

As discussed previously, the PageRank of a page is based on the number and importance of the pages linking to it. QD-PageRank is assigned in a similar manner, with the additional constraint that pages are also query relevant. The PageRank vector is therefore representative of the PageRank contributions (The PageRank value divided by the number of out-links, with the result multiplied by 0.85) of relevant and non-relevant in-links. Through EQD-Rank a type of PageRank reverse-engineering is suggested, with the PageRank contribution of non-relevant in-links removed, providing an estimation of the QD-PageRank values at runtime. The EQD-Rank algorithm is therefore:

EQD-PR Algorithm:

$$\textit{EQD-PR}\left(i\right) = \textit{PR}\left(i\right) - D\sum_{j \notin A_i} \textit{PR}\left(j\right)/\textit{Outlinks}\left(j\right) \tag{4.3}$$

Where A_i is the set of relevant pages, the EQD-PR of a page i is the PageRank (PR) of page i minus the PageRank contribution of non relevant pages j, with the damping factor (D) applied.

Implementation

The EQD-PageRank process begins with the computation of the PageRank vector "*a-priori*". Subsequently a query-relevance vector is computed through a content-based retrieval metric such as TFIDF, Latent Semantic Indexing, or Probabilistic Latent Semantic Indexing.

All pages falling below some relevance threshold are assigned an EQD-Rank of 0, requiring no further processing. A non-zero EQD-Rank value is then assigned exclusively to all relevant pages, based on a traversal of the local graph associated with each page. The depth to which this local graph is traversed is arbitrary and dependent on the maximum time a user is prepared to wait for result-set compilation (between 10 and 20 seconds).

The EQD-Rank (equation 4.3) is implemented across the set of query-related pages. Computation of the EQD-Rank score begins with an examination of the immediate in-links to a page. For example, consider a dataset with the three nodes A, B and C.

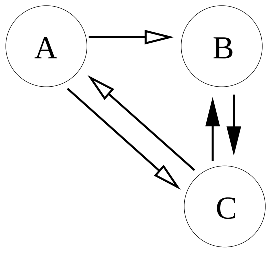

Figure 4.6: A Simple 3 Node Graph

In figure 4.6, black arrows link relevant-pages and white arrows symbolize a link either to or from a non-relevant page. When determining an EQD-PageRank score for page B we examine it's in-links, pages A and C. From the graph it is therefore evident that in-link C is relevant and A is irrelevant. As a result the PageRank contribution of page A ($0.85\,(Pr(A)/Outlinks\,(A))$) is removed from the PageRank of page B. As C is also query relevant it's rank contribution is not removed from page B.

In the event that all in-links to a page are relevant, the EQD-Rank and standard PageRank value will be equivalent. At the other extreme, a page with no query-relevant in-link will obtain an EQD-PageRank of $(1 - D)$. A page with only query-relevant in-links, may however have EQD-Rank and QD-PageRank values that differ significantly. This occurs when the PageRank value of an in-link is composed from the rank contribution of a majority of non-relevant pages. This prompts further examination of the relevance of the pages linking to each in-link, to increase estimation accuracy.

N-Depth Traversal

Each time the in-links to a page are examined, the depth of the traversal is increased by 1. Therefore when the initial in-links of a page are examined, as with the example from figure 4.6, the depth of the traversal (N) is one. To increase the accuracy achieved, further in-links may also be examined. When computing the EQD-Rank of page B, the inlinks of pages A and C may also be examined, resulting in a traversal with a depth of two.

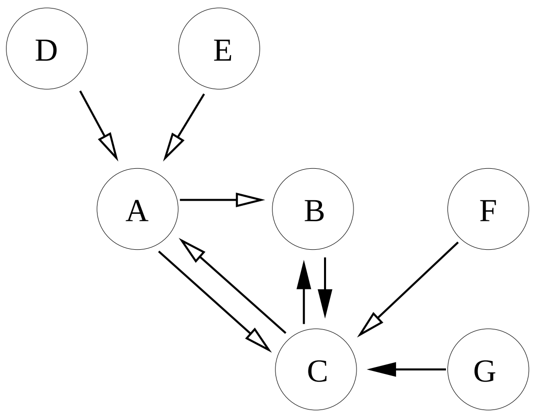

Figure 4.7: Example of N-Depth Traversal

From figure 4.7 we can see that page A is linked to by pages D, E, and C and that page C is linked to by B, F and G. As page A is deemed irrelevant there is no need to examine it's in-links, due to the fact that the total PageRank contribution of A is removed from B. Page C is deemed relevant therefore with $N = 2$, the in-links of C are also examined. In this case pages B and G are relevant, with F classified as irrelevant. Therefore the PageRank contribution of F is removed from the PageRank of C, and subsequently leads to the removal of further rank from page B.

Therefore when $N = 2$, not only the relevance of the in-links to a page, but also the relevance of the in-links to the in-links, affect the EQD-Rank of a page. The resulting distance between the EQD-Rank and QD-PageRank vectors is reduced. This is due to the fact that the local-graph traversed is closer in size to the query-relevant subgraph used to calculate the QD-PageRank vector. The depth of the traversal can continually be increased in this manner until the EQD-Rank vector is composed exclusively from the PageRank contribution of relevant pages. It is therefore suggested that N be increased until a sufficient level of accuracy is achieved, within a reasonable time-bound.

4.3.2 Characteristics

EDQ-Rank exploits the large number of community structures present in the global haphazard structure of the web. The connectivity information within these communities is currently not fully exploited by the PageRank algorithm, due to the equal treatment of query relevant and irrelevant links. To demonstrate this notion it is therefore suggested that if a page is related to science, examination of the link structure between the science community pages will return a better measure of page importance, than examination of the science page in relation to the global structure.

The effectiveness of EQD-Rank is dependent on the breadth of the user query and therefore on the size of the subgraph. The accuracy of the estimation does not decrease for a more specific query, but if the query is too specific, the size of the relevant subgraph can be very small and potentially result in the inclusion of less relevant pages in the result set. When the query is highly specific, EQD-Rank is also susceptible to localised link spamming. For example if a webmaster creates a set of web pages, all related to a specific topic, and points all out-links to one relevant page, using QD-PageRank and subsequently EQD-Rank, the page receives an over-inflated value. This effect is present also for standard PageRank but is not as prevalent due to the global nature of the algorithm. To counterbalance this effect, it is suggested that when the number of relevant pages falls below a predefined threshold, the standard PageRank algorithm be implemented.

A strength of EQD-Rank is that the closer the traversal depth to one, the greater the affect of the in-link examination. In other words, the difference between the rank orderings at a depth of five and four will be much less significant than the difference achieved at depths of one and two. This characteristic is significant, due to the fact that the computational cost increases considerably with each further layer of inlinks examined.

4.3.3 EQD-Rank Example

The calculation of the EQD-Rank vector is best described through an example. For consistency we select the same web graph used in the PageRank and QD-PageRank examples. The graph therefore consists of twelve nodes ranging from A to L and twenty eight links.

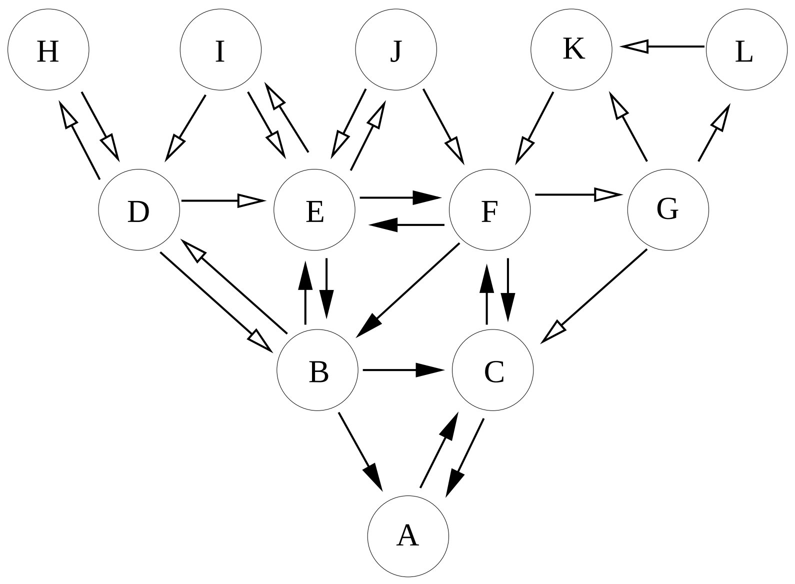

Figure 4.8: Sample Dataset Graph for EQD-Rank

After a user query is input and some relevance technique is implemented, we state that the pages A, B, C, E and F are relevant and that pages D, G, I, J, K and L are irrelevant. Once more, links to relevant pages are displayed using black arrows and links to or from non-relevant pages are displayed using white arrows. Initially the set of irrelevant pages are automatically given an EQD-Rank of 0. The PageRank values for each relevant page are then manipulated using the EQD-Rank paradigm, estimating the query-dependent values, and therefore avoiding creation of the topic-related subgraph. From the standard PageRank example in section 4.1.6, we know that the PageRank vector is:

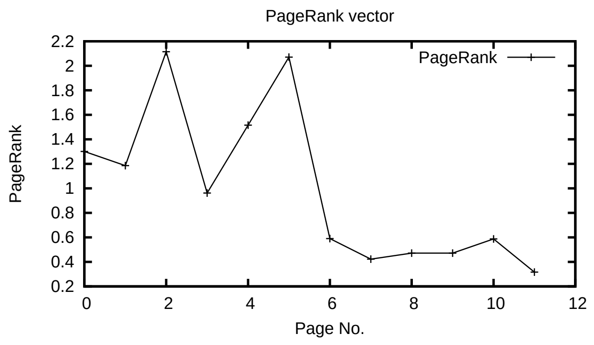

Figure 4.9: PageRank Vector

Using EQD-Rank for $N = 1$ with a depth traversal of 1, the relevance of the in-links of pages A , B, C, E, and F are examined. Taking page A as the example the relevance of in-links B and C are examined. In this case both pages are relevant and therefore no further computation is required. Therefore the EQD-Rank value for page A, with $N = 1$, is the same as the standard PageRank value. With the depth equal to one the EQD-Rank vector is:

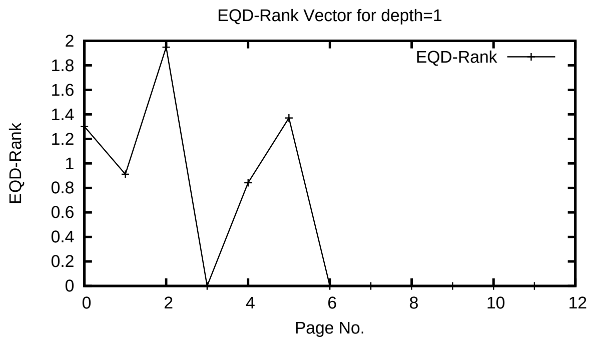

Figure 4.10: EQD-Rank at a Depth of One

For $N = 2$, again inlinks of pages A, B, C, E, and F are examined, with the additional constraint that the inlinks of their inlinks are also examined. Taking page A as the example again, the relevance of inlinks B and C is examined. Both in-links are deemed relevant and therefore the relevance of the in-links to both these pages is also examined.

Page B has an in-link from pages D, E, and F. In this instance pages E and F are relevant and page D is irrelevant. Therefore the PageRank contribution of page D is removed from page B. The ranking contribution removed is $0.85\,(0.962/3)$ resulting in a total removal of 0.2725 from the PageRank of B. This drop in rank of page B will in turn be passed on to page A. To determine the total rank to be removed from A we divide this drop in rank by the number of out-links associated with B and multiply the result by 0.85 to take into account the damping factor. This results the removal of 0.0579 from the rank of A.

Next the in-links of page C are examined, namely page F and page G. In this case page F is relevant and page G is irrelevant. Therefore the PageRank contribution of page G will be removed from page C. In this instance the contribution to be removed is $0.85\,(0.590/3)$, resulting in the removal of 0.167 from the PageRank value of page C. Therefore this drop in rank will in turn be passed on to page A. To determine the total rank to be removed from A, we divide this drop in rank by the number of out-links associated with C and multiply the result by 0.85 to take into account the damping factor. This results in a total removal of 0.071 from A.

The result of a depth 2 traversal for page A, is that a total of 0.0579+0.071 must be removed from its PageRank value resulting in an EQD-Rank value of 1.1722. The resulting EQD-Rank vector is displayed in figure 4.11:

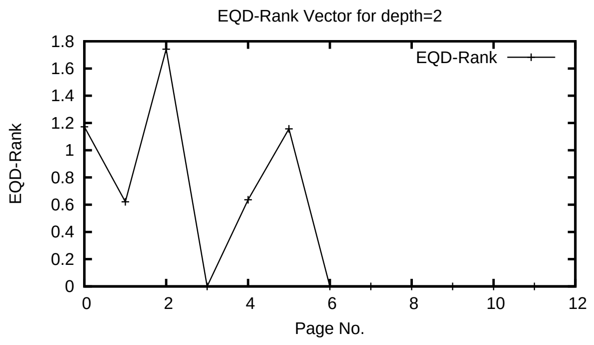

Figure 4.11: EQD-Rank at a Depth of Two

Chapter 5

Experimental Analysis

5.1 Introduction

The distribution of rank within the dataset for the three link analysis ranking algorithms is examined in this chapter, along with a justification for rank accumulation within certain areas of the dataset graph. Convergence rates are the subject of the first section, focusing on the PageRank and QD-PageRank algorithms, and the relationship between the convergence rate and the number of dataset pages. Also examined are the EQD-Rank computation speeds and the potential tradeoff between the accuracy of the induced orderings and the computation time.

In the second section the distribution of PageRank as the number of dataset pages increases is analyzed, along with the global dataset-distribution. Also observed is the effect a hierarchical structure has on the distribution of PageRank within an individual host.

The rank distribution induced using the QD-PageRank and EQD-Rank algorithms is considered in the third section. For QD-PageRank we examine if the self-similarity of the web [DKM$^+$01] is exhibited in the dataset and also measure the correlation between PageRank and the In-Degree. The distribution of rank generated using EQD-Rank within the dataset, the affect cohesive-community interconnectivity has on the amount of rank removed, and the correlation between the rank removed and the depth of the EQD-Rank traversal, are also analyzed.

The different content-based document selection metrics available to assign a relevance ranking, such as TF*IDF and Latent Semantic Indexing (LSI), are highlighted in section four with a discussion of the merits and demerits of both. The final section presents a discussion of the rank-ordering similarity measure selected to compare the induced result-sets.

5.2 Rates of Convergence

The rate of convergence for PageRank is ascertained within subgraphs, cohesive communities, and on a global scale in the following section, with the following test for convergence:

$|X_i^{(k+1)} - X_i^{(k)}|/|X_i|^{(k)} < 10^{-3}$

The total rank in the dataset is measured following each iteration and a comparison is made with the total rank from the previous iteration, with convergence achieved when the difference is less than three significant decimal points.

5.2.1 PageRank Convergence

The global PageRank vector is computed in 25 iterations, a graph of which is displayed below:

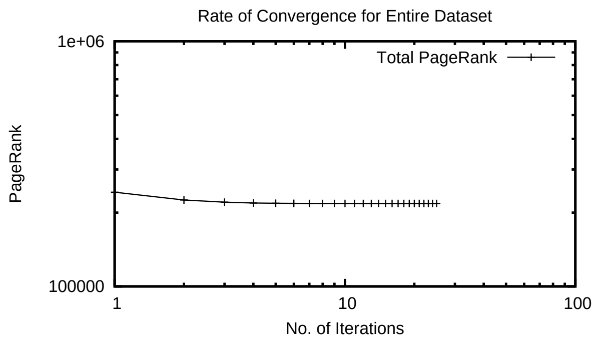

Figure 5.1: PageRank Convergence

Analysis reveals that after 12 iterations the majority of pages have converged and that following 18 iterations rank ordering is established. The last seven iterations do not effect the rank orderings and it is therefore suggested that an optimal test for convergence would measure the rank orderings (Kendall Tau), instead of the rank values. The rate of convergence achieved is comparable with that of Brin and Page [BP98], where convergence on a 322 million page dataset is achieved within 52 iterations.

5.2.2 QD-Rank Convergence

The rates of convergence within cohesive communities, generated for queries ranging from broad to narrow specificity, are also evaluated. The convergence rates are highlighted for the communities generated using the queries ‘shopping’, ‘football’ and ‘physics’. The number of iterations required is small in comparison to the number required to achieve convergence across the entire dataset, due to the decreased interconnectivity. The rate of convergence for these three queries is displayed in the following three successive graphs:

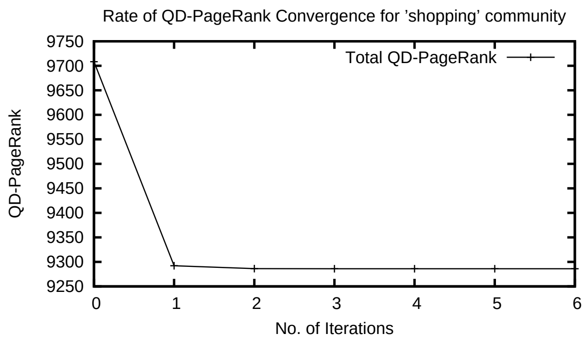

Figure 5.2: QD-PageRank Convergence for ‘shopping’ community

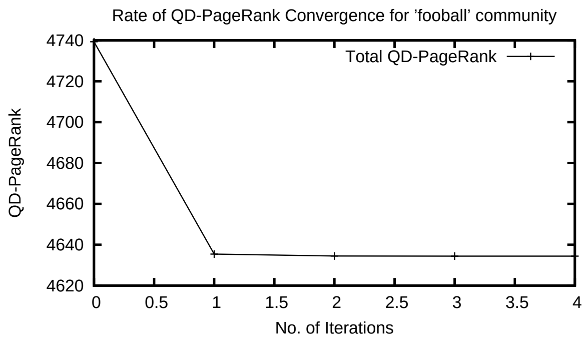

Figure 5.3: QD-PageRank Convergence for ‘football’ community

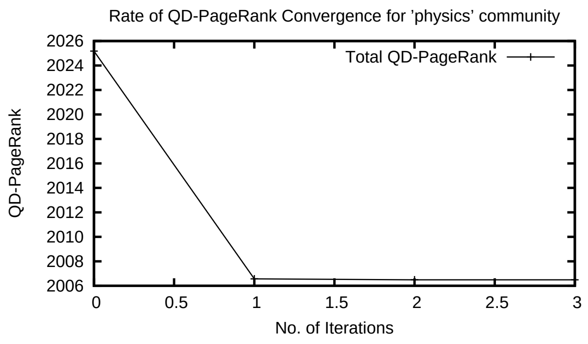

Figure 5.4: QD-PageRank convergence for ‘Physics’ community

The rate of convergence within the ’shopping’ community is displayed in figure 5.2 and illustrates that the final QD-PageRank vector is computed within six iterations. This expeditious rate of convergence, for a community containing $60,886$ pages, can be attributed to the fact that a relatively moderate amount of links are present, totaling of $69,781$. As the specificity of the query increases the rate of convergence also increases, as illustrated in figures 5.3 and 5.4. The convergence within the ’football’ and ’shopping’ communities is achieved within four and three iterations respectively and can be attributed to the low level of interconnectivity within these cohesive communities.

5.2.3 EQD-Rank Computation times

The “Achilles heel” of the QD-PageRank algorithm is the inability to compute the QD-PageRank vector within real-time, following the obtainment of the user query. Subsequently the EQD-Rank approximation algorithm is introduced, providing the relevance obtainable using QD-PageRank within real time speeds. It is a common consensus that a typical search engine user is prepared to wait at most between 10 and 20 seconds for the result set and therefore EQD-Rank must be implementable within this time span. The current average computation times for thirty query words are displayed in the table below:

Algorithm	QD-PR	EQD-L1	EQD-L2	EQD-L3
Computation Time (seconds)	217	24	30	36

Table 5.1: Computation Times Using EQD-Rank

From the graph above it can be observed that an 83 % decrease in the average computation time is introduced using EQD-Rank at a depth of three with the figure increasing to 89 % at a depth equal to one. The average computation times currently range between 24 and 36 seconds, using a naive implementation strategy. It is suggested that the implementation of algorithmic optimization techniques would reduced these figures drastically. A refinement of the number of pages qualifying for cohesive-community selection could be introduced, using a threshold based on the content-based relevance score (see section 5.4) and consequently lead to a further reduction in the computation speeds. This is an area of future research and it is suggested that using the techniques discussed above, real-time computation speeds can be achieved.

5.3 Rank Distributions

5.3.1 PageRank Distribution

In this section we evaluate the change in PageRank distribution as dataset variables are varied. Initially the number of dataset pages is varied to facilitate the characterization of the degree of distribution change. Individual host distributions are also analyzed to ascertain if any abnormal deviation is introduced.

PageRank Distribution within Corpus Subsets

A subset containing 10,000 pages is examined initially, to evaluate the PageRank distribution. Construction of the subset involves the selection of one out of every hundred dataset pages, therefore preventing the presence of a cluster of pages from one particular site or host. The inherent distribution is displayed below:

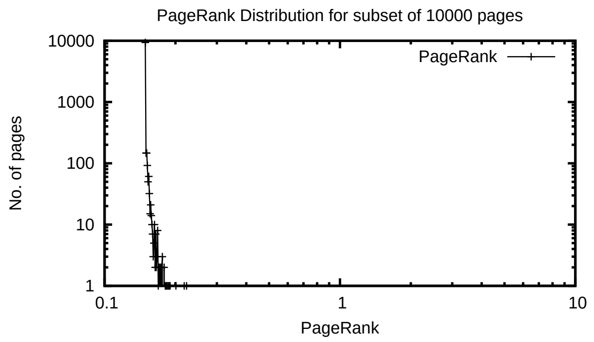

Figure 5.5: PageRank Distribution for 10,000 Pages

This subset contains an interconnectivity of 2426 links, with only 659 of the pages receiving an inlink and therefore explaining the small range in the distribution of PageRank. Despite this low interconnectivity the distribution is power law with an exponent of 2.1, with the vast majority of pages receiving a small PageRank value and a few receiving a high value (with respect to the majority). This distribution is symptomatic of power-law scaling and can be attributed to a "rich get richer" mechanism called preferential attachment. As the graph grows the probability that a node will receive an increase in PageRank is proportional to the amount of PageRank it already has. In order to examine if the distribution pattern is consistent across datasets of varied size we, also analyze the PageRank distribution within a subset of $100,000$ pages.

The subset of pages is generated by selecting one out of every ten pages to minimize the influence of one or two individual hosts. The distribution within this dataset is displayed below:

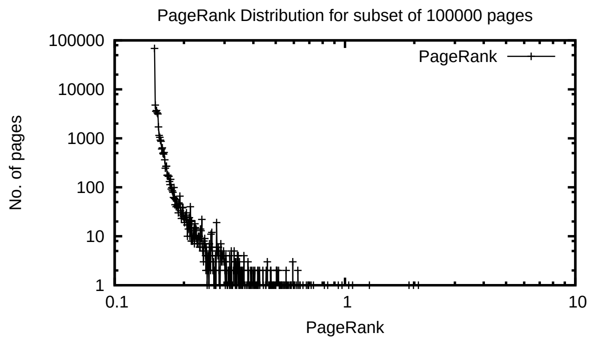

Figure 5.6: PageRank Distribution for 100,000 Pages

The interconnectivity in this instance is significantly greater, consisting of $130,984$ links. From figure 5.6 it can be observed that the distribution is power law with best fit exhibited for an exponent of 2.1 once more and is due to the "rich get richer" behaviour.

Global Distribution

The global PageRank distribution within the total dataset consisting of $1,023,285$ pages and $12,496,560$ links is displayed in the log-log plot below:

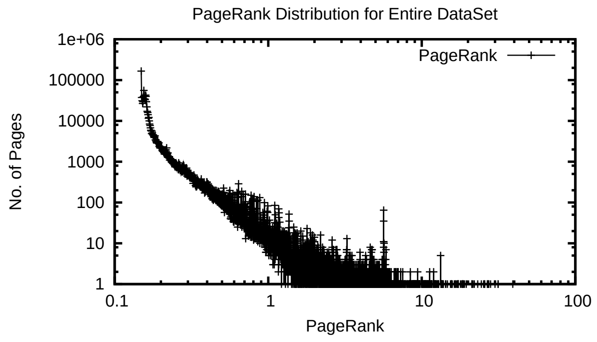

Figure 5.7: PageRank Distribution for Entire DataSet

Analysis of figure 5.7 reveals that the the distribution of rank ranges from a minimum of 0.15 to a maximum of 39.13, with the minimum value representative of pages not linked to from other pages within the dataset. Using the 'random surfer' analogy, these are the pages that the random surfer can reach through a random jump, but not by following the page out-links. The power law tail illustrated in this graph characterizes not only the PageRank distribution but also the connectivity of many naturally occurring network distributions, research paper citations, movie actor collaborations, and United States power grid connections [PFL$^+$02]. The distribution of PageRank is once more power law with best fit exhibited for an exponent of 2.1.

We therefore conjecture that irrespective of the dataset size, the PageRank distribution is very well approximated by the function $p(\text{PR}(x) = k)) \propto k^{-2.1}$.

PageRank Distribution within a particular Host

To examine the influence of a hierarchical structure on the PageRank distribution, the distribution among pages downloaded from *amazon.com* is measured. The dataset consists of 9092 pages and figure 5.8 shows a significant deviation from the global PageRank distribution. This is due to a structure that is prevalent with many internet websites. The structure is comprised of a home page that is heavily linked to and subsequent pages that are linked to less heavily as the depth of the site increases. In the case of *amazon.com*, the interconnectivity consists of $273,544$ links with an average of 30 links per page. PageRank values range between 0.15 and 5.83, with the maximum value assigned to a *contact us* page that receives a total of 937 inlinks. Considering the relatively small dataset size, the diversity among the rank values is significant, and is affiliated with the large amount of interconnectivity and to a larger extent the hierarchical structure of the site. The PageRank distribution in this instance is power law and concurs with [PFL+02], where it is suggested that within competing pages of the same type, this rich get richer distribution is not as prevalent.

`java/hostDistribution.ps not found!`

Figure 5.8: PageRank Distribution within subset of Amazon.com

5.3.2 Inlink Distribution

The dynamics of link accumulation can strongly influence competition on the web and therefore leads to the necessity of global dataset In-Degree characterization. The In-Degree distribution is therefore computed and displayed below in a log-log plot:

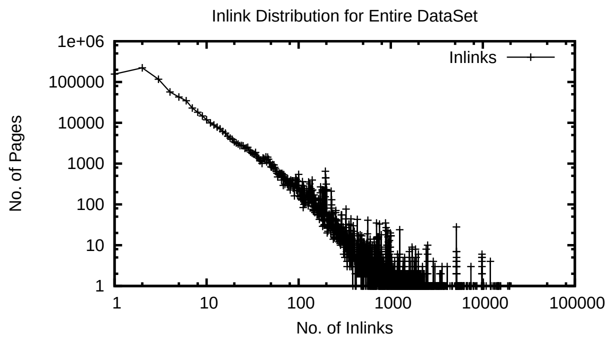

Figure 5.9: IN-Degree Distribution for the Entire DataSet

This distribution is in strong agreement with previous web analysis experimentation [KRRT99, BM01, Hav02] and similar to PageRank distribution, is very well approximated by the function $p(\text{In-Degree}(x) = k)) \propto k^{-2.1}$. This distribution is another example of the "rich get richer" mechanism, with the majority of pages receiving a few inlinks and a small percentage of pages receiving the majority, enough to skew the mean well above the median. The number of inlinks to a page, while being insufficient as a measure of relevance due to a high susceptibility to link spamming, can be used as an indication of page importance.

5.3.3 QD-PageRank

In this section we analyze the distribution of PageRank within cohesive communities of web pages as opposed to the global distribution across a number of pages related to a diverse set of topics. Initially the distribution is examined for six queries of varied specificity to provide a cross section of the PageRank distribution. The distribution of In-Degree is also examined in the following section. Then in the third section the correlation between PageRank and In-Degree is evaluated. In the final section we look at the amount of PageRank obtained by the top k percent, as it may have implications for the compression of the inverted index.

QD-PageRank Distribution

Cohesive collections of web pages (query-related subgraphs) are examined in this section, for numerous queries ranging from broad to narrow specificity. The communities induced for two queries of broad, medium and narrow specificity are analyzed in this instance. Various community attributes are also examined and we attempt to ascertain if cohesive-collection PageRank distributions mirror the global distribution.

Induced distributions for queries of broad specificity are examined initially with the query words ‘shopping’ and ‘games’ selected. The corresponding PageRank distributions are displayed below:

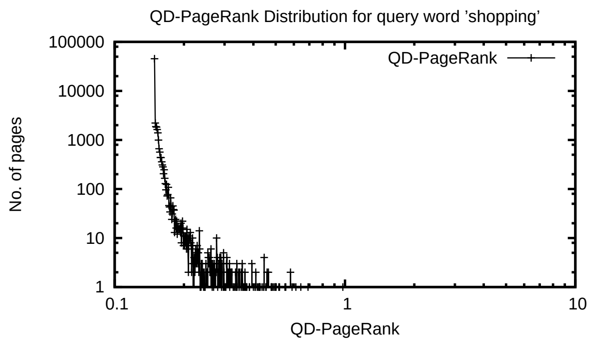

Figure 5.10: QD-PageRank Distribution for the ‘shopping’ Community

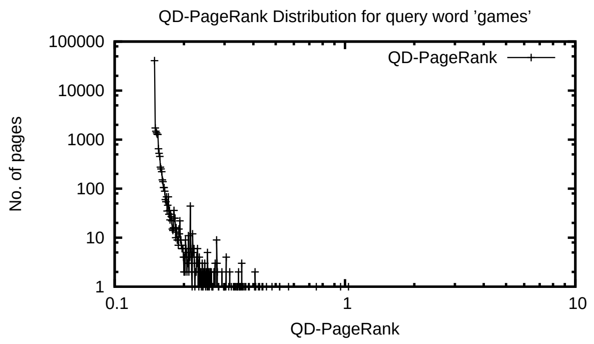

Figure 5.11: QD-PageRank Distribution for the ‘games’ Community

The distribution of QD-PageRank for a ‘shopping’ cohesive community consisting of $60,886$ pages, is displayed in figure 5.10. The community represents 5.95% of the dataset pages and contains an interconnectivity of $69,781$ links. The total amount of QD-PageRank distributed within this community is 9285.94 with values ranging from a minimum of 0.15 to a maximum of 0.979. The distribution of QD-Rank within the ‘games’ cohesive community is displayed within Figure 5.11, with the number of community pages in this instance equal to $52,023$ and the number of intercommunity links equals $40,237$. The distribution of QD-PageRank is power law with exponent 2.1 for both queries of broad specificity.

A significant reduction in the community interconnectivity is introduced through queries of medium specificity to evaluate if the power law distribution is preserved. The query words selected in this instance are ‘footaball’ and ‘computers’ with the respective distributions displayed below:

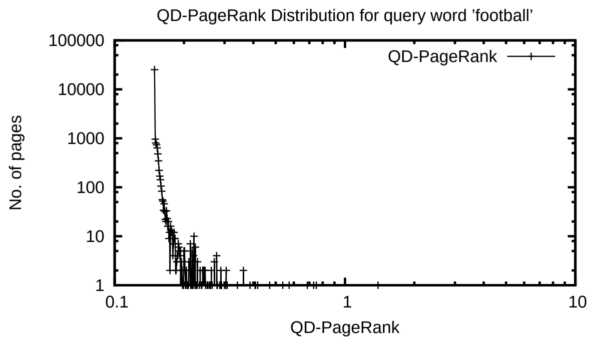

Figure 5.12: QD-PageRank Distribution for the ‘football’ Community

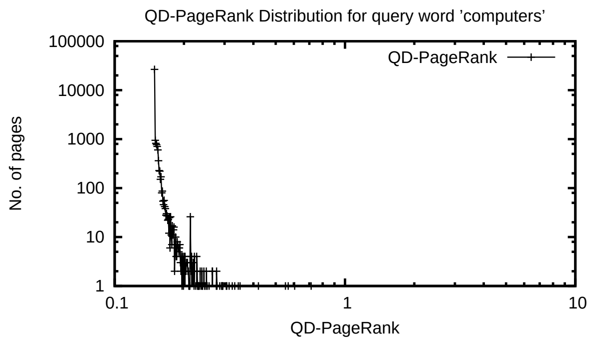

Figure 5.13: QD-PageRank Distribution for the 'computers' Community

For the 'football' and 'computers' communities it can be observed that the distribution remains consistent, although on a somewhat smaller scale. The community sizes in this instance are $30,622$ and $32,440$ with inter-connectivities of $18,229$ and $21,712$ respectively. This distribution is again power law with exponent 2.1 despite a reduction in the community size of an average of over 44%. It is therefore suggested that the PageRank distribution is immune to query specificity and subsequently community-interconnectivity.

To reinforce this hypothesis we evaluate the distribution within cohesive communities of low connectivity. The communities are generated with the narrow specificity query words 'physics' and 'biology', with the PageRank distributions displayed below in the log-log plots:

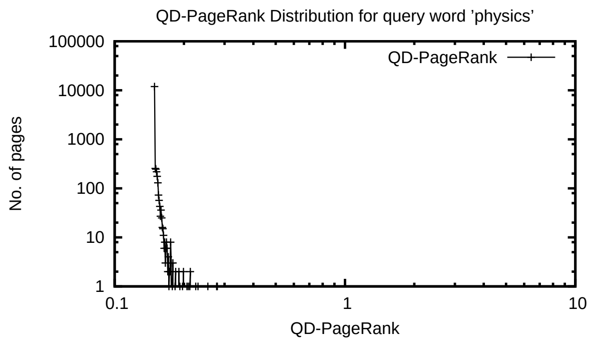

Figure 5.14: QD-PageRank Distribution for the ‘physics’ community

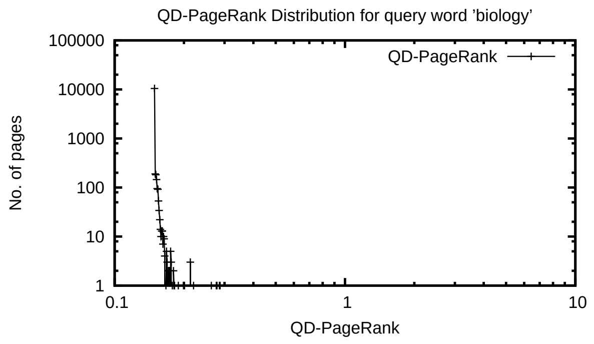

Figure 5.15: QD-PageRank Distribution for the ‘biology’ community

The ‘physics’ and ‘biology’ communities in this case, consists of $13,329$ and $11,382$ pages and $4,582$ and $3,165$ links respectively. In these cases the connectivity is extremely low with a large proportion of the nodes isolated, receiving no inlink and therefore receiving a PageRank value of 0.15. The tail in these distributions is steeper than with previous distributions and it is conjectured that this is related to the significant decrease in the number of community in-links, although the correlation between the number of inlinks and the QD-PageRank of a page has yet to be assessed.

PageRank distribution analysis within cohesive communities of web pages of varying size, results in the following premise. Within communities of varied interconnectivity the distribution can be characterized though the function $p(\text{QD-PR}(x) = k))$

$\propto k^{-2.1}$. In extremely low connectivity cohesive communities a slight deviation from this premise is experienced and the rich get richer effect is not as prevalent.

Distribution of IN-Degree

The distribution of In-Degree within cohesive communities is evaluated in this section. Suggested by [DKM$^+$01] is that global dataset properties are preserved at community level and we ascertain if this is the case for In-Degree distribution. The global distribution of In-Degree is power law with best fit exhibited for an exponent of 2.1. If the self-similarity premise applies at community level, the inherent distributions should follow the same distribution pattern. The cohesive communities that we focus on are the two broad, narrow and specific query communities from the previous experimentation, the In-Degree graphs of which are displayed below:

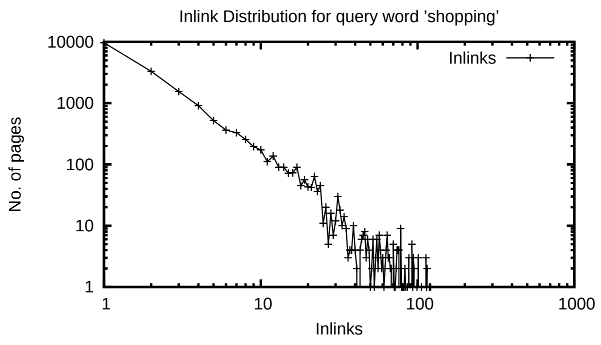

Figure 5.16: In-Degree for the ‘shopping’ Community

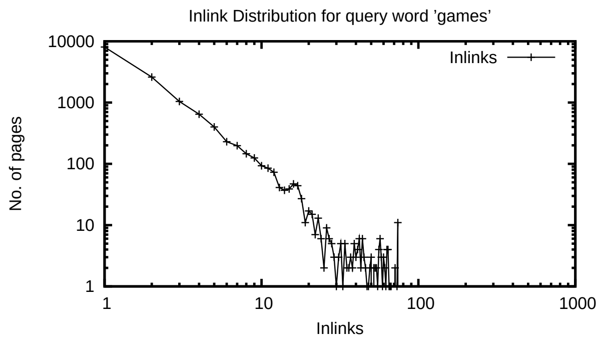

Figure 5.17: In-Degree for the ‘games’ Community

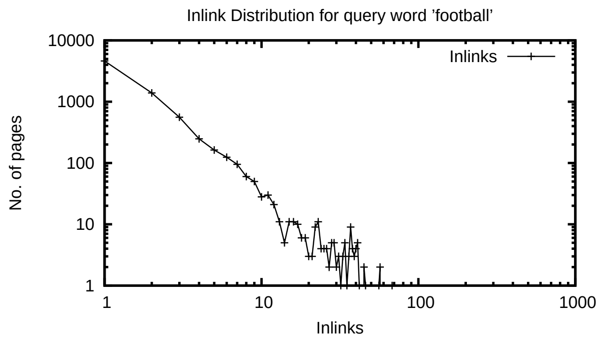

Figure 5.18: In-Degree for the ‘football’ community

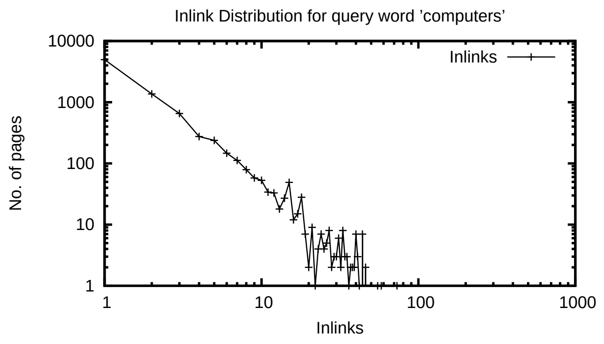

Figure 5.19: In-Degree for the ‘computers’ community

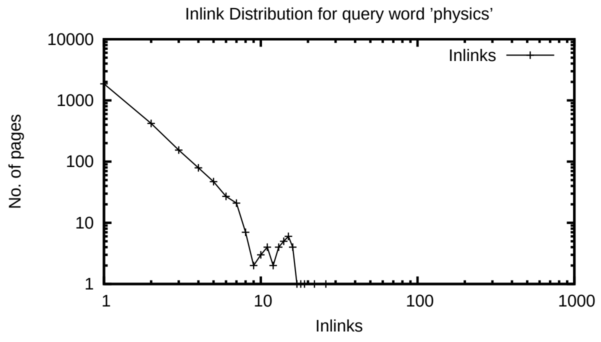

Figure 5.20: In-Degree for the ‘physics’ community

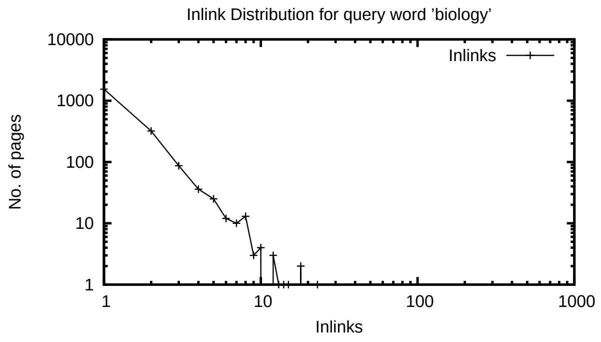

Figure 5.21: In-Degree for the ‘biology’ community

Analysis of figures 5.16 to 5.21 reveals that the In-Degree distribution at community level is equivalent to the global distribution and is therefore best characterized as power law with an exponent of 2.1. Slight deviations from power law scaling are exhibited for the narrow queries ‘physics’ and ‘biology’, although overall we conjecture the self-similarity premise is maintained within cohesive community-structures of the web dataset. These results conflict slightly with those of Pennock et al. [PFL$^+$02], where it is suggested that the In-Degree distribution for sub categories of pages such as university home pages deviate from power law scaling, with the magnitude of the deviation varying depending on the community category.

Examination of the Correlation between QD-PageRank and the In-Degree

If a page has a high PageRank value does it also have a high In-Degree or are they unrelated?. Intuitively one might expect that a page with a high In-Degree will have a high PageRank value and vica versa. The correlation coefficient between the two is therefore assessed in order to measure the validity of the previous hypothesis and is displayed below for the six queries of varied specificity.

word	shopping	games	football	computers	physics	biology
correl.	0.297	0.181	0.219	0.177	0.239	0.316

Table 5.2: Correlation Between PageRank and In-Degree

From the results above we perceive a weak correlation between the two, while observing that the strongest relationship exists within the ’biology’ community. We conjecture that the reason for the relatively high correlation in this instance is the low PageRank values resulting from a sparsely-connected community graph. These findings are in strong agreement with [PRU02, CRL$^+$03], where a weak correlation is also reported. It is therefore conjectured that a PageRank value is largely independent of the In-Degree of a page, although it may be weakly indicative of the value.

Examination of the Percentage of QD-PageRank obtained by the Top k percent of Pages

The percentage of PageRank obtained by the top k percent of pages merits further analysis, as it can have implications for the compression of the inverted index and for optimization of the available storage. In this case the PageRank obtained by the top ten percent is highlighted, within the communities generated from the six queries of varied specificity, with the results illustrated below:

word	shopping	games	football	computers	physics	biology
% obtained	12.85	12.27	12.07	12.06	11.17	11.29

Table 5.3: The Percentage of Total PageRank Obtained by Top 10% of Pages

The figures displayed in table 5.2 above exhibit the percentage of PageRank obtained by the top 10 percent of pages, within each of the six cohesive communities. These figures are below the expected values and it is conjectured that the reason for this is the relatively low interconnectivity and smaller range of distribution within these communities. From the table above it can also be observed that the greater the specificity of the query word, the lower the percentage of Pagerank obtained by the top ten percent. We therefore postulate that the greater the interconnectivity within a dataset of webpages the greater the percentage of PageRank.

To measure the validity of this premise, the top ten percent within the global distribution is assessed. In this instance the PageRank obtained by the top ten percent of pages is 29.97 % and is symptomatic of a power law distribution. This culminates in the supposition that given the number of nodes and level of interconnectivity, the percentage of rank obtained by the top k percent of pages could be accurately approximated.

5.3.4 EQD-Rank

The main focus of this section is to validate EDQ-Rank as an optimal algorithmic implementation for the estimation of the QD-PageRank vector while providing numerous indications to fortify this premise. Experimental analysis evolves around the evaluation of cohesive-communities generated using thirty queries of diverse specificity. The distribution of EQD-Rank within a selection of cohesive communities is initially examined. The relationship between the rank removed and the cohesive community query type is examined in section two, to determine if a bias towards communities of a certain specificity exists. In the final section the amount of rank removed using EQD-Rank at a depth ranging from one to three is appraised to measure the affinity between the EQD-Rank and QD-PageRank vectors.

EQD-Rank Distribution

The EQD-Rank algorithm is a QD-PageRank vector approximation and it is conjectured that the similarity between the QD-PageRank and EQD-Rank distributions will provide an indication of the approximation effectiveness. Six queries of varied specificity are therefore focused upon with the resulting EQD-Rank distributions at a depth equal to two displayed in the figures below:

```
java/content/EQDRESULTS/shopping_EQD.ps not found!
```

Figure 5.22: EQD-Rank Distribution for the ‘shopping’ community

```
java/content/EQDRESULTS/games_EQD.ps not found!
```

Figure 5.23: EQD-Rank Distribution for the ‘games’ community

```
java/content/EQDRESULTS/football_EQD.ps not found!
```

Figure 5.24: EQD-Rank Distribution for the ‘football’ community

```
java/content/EQDRESULTS/computers_EQD.ps not found!
```

Figure 5.25: EQD-Rank Distribution for the ‘computers’ community

java/content/EQDRESULTS/biology_EQD.ps not found!

Figure 5.26: EQD-Rank Distribution for the ‘biology’ community

java/content/EQDRESULTS/physics_EQD.ps not found!

Figure 5.27: EQD-Rank Distribution for the ‘physics’ community

Comparison between the figures displayed above and the corresponding QD-PageRank distributions in figures 5.10 to 5.16 reveal that the EDQ-Rank distribution mirrors that of the QD-PageRank vector albeit on a comparatively larger scale. The distributions are power law with best fit exhibited for an exponent of 2.1 and a slight deviation experienced within the cohesive communities induced for query words of narrow specificity. The distributions within these communities are somewhat skewed and may be attributed to the existence of low community-interconnectivity. Overall the average correlation coefficient between the EQD-Rank and QD-PageRank distribution is 0.986. As a result of this high correlation-coefficient and the graphically illustrated similarity between the distributions, it is suggested that EQD-Rank provides a very close approximation of the QD-PageRank values, with further evidence provided in latter sections.

The relationship between the cohesive community connectivity and Rank Removal

The total amount of PageRank removed from each community, using EQD-Rank at a depth of two, is highlighted in this section to ascertain if there is a bias towards communities generated using a query of a certain specificity. Cohesive communities of web pages may contain QD-PageRank and PageRank differing significantly and we therefore measure the rank removed as a percentage of the total PageRank, to assess the relationship between the rank removed and the number of community pages. The rank removed for each of the thirty queries used in the analysis of EQD-Rank is displayed in the table below:

Word	**Size**	**PR**	**QD-PR**	**EQD-PR**	**%**
abortion	4122	871.04	619.05	619.06	28.93
alcoholism	1094	232.17	164.13	164.13	29.31
architecture	14647	3250.72	2205.69	2205.85	32.14
bank	24333	5341.89	3673.73	3674.66	31.21
bicycling	874	178.84	131.11	131.11	26.69
biology	11382	2466.39	1712.09	1712.42	30.57
cheese	6334	1364.18	951.91	951.98	30.22
chemistry	8683	1846.21	1305.05	1305.10	29.31
complexity	4608	978.69	691.86	691.86	29.31
computers	32440	7042.37	4909.90	4912.67	30.24
cruises	2628	554.91	394.48	394.48	28.91
energy	35291	7742.69	5345.46	5347.32	30.94
fitness	18635	3982.99	2809.37	2809.57	29.46
football	30622	6758.18	4634.44	4635.86	31.40
games	52023	11055.85	7898.77	7902.55	28.52
golf	17388	3861.64	2621.90	2622.41	32.09
hiv	5474	1182.04	822.79	822.83	30.39
holiday	16291	3554.14	2454.24	2454.58	30.94
jaguar	1717	364.13	257.76	257.77	29.21
java	15134	3264.89	2279.15	2279.50	30.18
law	61877	13533.23	9440.94	9449.21	30.23
nutrition	17725	3781.94	2670.94	2671.21	29.37
physics	13329	2892.97	2006.49	2006.66	30.64
recreation	9367	1992.74	1407.92	1408.00	29.34
shakespeare	2430	530.12	364.71	364.71	31.20
shopping	60886	13268.12	9285.95	9296.09	29.94
sushi	901	198.99	135.17	135.19	32.07
terrorism	30622	1694.53	1113.65	1113.72	34.28
volcano	32440	568.35	395.74	395.75	30.37
workplace	7418	1610.75	1115.60	1115.68	30.74

Table 5.4: The Percentage of PageRank Removed Within Each Cohesive Community

In the table above the initial column displays the word from which the cohesive community is created. The column labelled ‘size’ indicates the number of pages within the cohesive community, the proceeding three columns indicate the amount of rank within the community for each of the three ranking methods, and the final column displays the percentage of rank removed, using EQD-Rank, as a percentage of the total PageRank.

Examination of table 5.3 reveals that the removal of PageRank from communities of varied type is consistent when viewed as a percentage of the total community PageRank. As the size of the cohesive community varies, the percentage of PageRank removed remains relatively consistent with the total removal ranging between 26.69 and 34.28 percent for each of the queries tested. As a result we postulate that on average 30.27 percent of a page's PageRank value will be removed during the EQD-Rank calculations for a depth equal to two.

The amount of Rank removed in relation to the number of layers examined

As the depth of inlinks examined increases, using EQD-Rank, the accuracy of the estimation increases also, and therefore so too does the amount of PageRank removed from the community pages. In this section we examine the amount of PageRank removed in relation to the number of number of layers examined and measure the result with the QD-PageRank values. In table 5.5 we display the average amount of rank within a community using the PageRank and QD-PageRank algorithms and EDQ-Rank using a depth from one to three.

Pr	**QD-PR**	**EQD-PR**	**EQD-D2**	**EQD-D3**
3120.69	2169.41	2186.94	2170.36	2169.93

Table 5.5: Total Rank assigned using each Algorithm

From table 5.5 above it can be seen that the average amount of QD-Rank (column 2) is 69.51 percent of the total induced using PageRank (column 1). The next three columns show the average amount of rank achieved using EQD-Rank starting a depth of one and arriving at a depth equal to three. It can be observed that as the depth of inlinks examined increases so to does the affinity between the average QD-PageRank and EDQ-Rank values. As a result we conjecture that the amount of rank removed is inversely proportional to the number of layers examined. It is also suggested that the close approximation of the average QD-PageRank vector, using EQD-Rank at a depth of two and three, is a further indication of the degree of accuracy obtainable using EQD-Rank.

5.4 Content-Based Document Selection Techniques

In order to determine the relevance of a document in relation to a user query, it is necessary to implement a content-based document selection technique. The methods available typically provide a content-based representation of a document, based on the term frequency within a document, with the user query finally compared with the document score.

One simplistic approach involves representing a document as a 1 or a 0 and is referred to as *binary judgement*. Using this method a document is represented through a boolean vector, with a 1 indicating term presence and a 0 non-presence. This is exact match text filtering and results in an unranked set of documents. Two other content-based models are the *probabilistic retrieval model* and the *vector space model*. With the former, documents are ranked based on the probability of their relevance in relation to a user query and often rely on term weighting schemes to estimate the relevance. The later technique involves representing every document as a vector of terms and is examined further.

5.4.1 TF*IDF

The Vector-Space Model (VSM) [SWY71] represents every document as a vector of terms, with associated weights describing a term's value as a label for a document. The vector-space effectiveness is improved considerably by transforming the term-frequency vector, to augment the influence of words occurring often in the document, but rarely within the entire document collection. The VSM procedure involves extracting terms representative of the document, weighting the terms, and finally ranking the documents based on a similarity measure (e.g the cosine of the angle between two vectors).

In the VSM term vectors are constructed using a term-selection matrix, measuring the importance of terms within the document. The term frequency of corpus documents is generally used as the weights for the document vector. For our experimentation however, we implement the TF*IDF [SB88] term selection metric and assign a weighting to each page, for every word in the lexicon. This weight is then used at runtime to determine relevant pages and can be combined with a link context-based score (e.g EDQ-Rank) at runtime, to provide a final ranking value.

Using the TF*IDF metric, the importance of a word in a document is based on the number of occurrences of the word within the document and the number of occurrences within all indexed documents. TF (Term Frequency) implies that terms used more frequently are more important and IDF (Inverse Document Frequency) works on the hypothesis that words occurring rarely in the index are highly discriminative.

*TF*IDF equation:*

$w_{ij} = tf_{ij} * \log_2 \frac{N}{n}$

where:

w_{ij}= weight of Term T_j in Document D_j

f_{ij}= frequency of Term Tj in Document D_j

N = number of documents in the collection

n = number of documents where Term T_j occurs at least once

TF*IDF works well for large datasets. However in the case of small text documents there may be insufficient data for a statistical evaluation of the descriptive value of terms. Term weights may also be normalized due to long documents generally having a larger term set than short documents, making larger documents more likely to contain increased occurrences of a word than short documents. The advantage of TF*IDF is that computation is simple and therefore fast to compute. However on the downside TF*IDF does not allow for synonymy, polysemy and noise in documents. To counteract these constraints we therefore suggest that future experiments incorporate Latent Semantic Indexing or Probabilistic Latent Semantic Indexing.

5.4.2 Further Document Selection Techniques

To counteract these problems we could exploit the fact that words with similar meaning tend to occur together and use Latent Semantic Indexing (LSI) to represent documents and queries, not by terms but by the underlying concepts referred to by terms. The objective of LSI is to provide information well beyond the lexical level and reveal relations between the entities of interest. Using singular value decomposition (SVD), LSI transforms a matrix of word vectors, computed using the Vector Space Model (VSM), to discover the semantics of the words and documents present. A high-dimensional word vector is transformed into a lower dimensional space. By reducing the multi-dimensional term space to a small number of dimensions, semantically close keywords get squeezed together. During this collapse noise is lost and information and content words are superimposed on one another. According to [DDL$^+$90], LSI is 30% more effective than other word-matching methods at helping users find relevant information. LSI considers documents which have many words in common to be semantically close even if they do not share a particular word. This way documents that are relevant and do not contain the keyword may be returned.

Other methods could also be introduced taking into account various page properties such as anchor text, text size, page title meta tags, capitalisation bit, font size, and position in document [BP98].

Another method that could be used to improve upon LSI is Probabilistic Latent Semantic Indexing (PLSI) and is based on a mixture decomposition derived from a statistical latent class model. This results in a more principled approach which has a solid foundation in statistics. It is based on the likelihood principle and defines a proper generative model of the data. PLSI can be used to achieve increased recall and precision and works well in cases were LSI fails completely [Hof99].

5.5 Rank Ordering Similarity

As a means of evaluating the algorithms, we measure the degree of result-set similarity. To do this we select a correlation coefficient. A correlation coefficient is intended to measure the strength of a relationship, with different correlation coefficients measuring the strength of the relationship in different ways. The 'strength' discussed in this instance refers to the tendency of the variables to move in the same or opposite direction. Some correlation coefficients available are the *product moment coefficient of correlation, the spearman coefficient,* and the *Kendall Tau coefficient.* For our experimentation we select the Kendall Tau metric to determine the strength in relationship between the rank-orderings induced using the PageRank, QD-PageRank, and the EQD-Rank algorithms.

5.5.1 Kendall Tau Algorithm

To evaluate the rank orderings induced and gauge the strength of the relationships we implement the Kendall Tau correlation coefficient [Ken38], as suggested in [EMT04]. Kendall's Tau is a non-parametric measure of the agreement between two rankings, essentially measuring the strength of the relationship between two paired observations. The values are initially ordered and numbered for each variable separately, with the Kendall Tau coefficient finally applied. Consider the observations (X_i, Y_i) and

(X_j, Y_j). If $X_j - X_i$ and $Y_j - Y_i$ have the same sign, the pair is described as concordant, otherwise if they have opposite signs the pair is referred to as discordant. For a sample of n observations we can form $n(n-1)/2$ pairs corresponding to choices $1 \leq i < j \leq n$. The Kendall Tau coefficient is therefore:

The Kendall Tau coefficient:

$$\tau_a = \frac{\sum \text{sign}(X_i - X_j)\ \text{sign}(Y_{j-} Y_i)}{n(n-1)/2}$$

The numerator is defined as the sum of the concordant pairs $\sum C$ minus the sum of the discordant pairs $\sum D$ with the denominator defining the total number of pairs. Hence, the statistic is the proportion of concordances to the total number of relations.

A positive correlation indicates that the ranks of the two variables increase together and a negative correlation indicates that as the rank of one variable increases, the other one decreases. In the case where all $n(n-1)/2$ pairs are concordant a maximum value of 1 is obtained. Correspondingly the minimum value of -1 is achieved if all the pairs are discordant. Using the Kendall Tau coefficient the odds ratio of P_c/P_D, where c and d are the number of concordant and discordant pairs, is equal to $(1+t)/(1-t)$. Therefore if the tau value is $1/3$ the set of observations $(X_i, Y_i)$and (X_j, Y_j) are twice as likely to be concordant as discordant.

Chapter 6

Conclusions and Future Study

6.1 Experimental Results Analysis

In this section we further justify EQD-Rank as a means of estimating the QD-PageRank vector. Initially the query word selection process is discussed. Following this, a measure of the degree of accuracy achieved with EDQ-Rank in estimating the QD-PageRank vector is provided, with the subsequent level of result-set relevance analyzed through a user study.

Discussion of Query selection

A justification for the selection of the queries used during experimentation is provided in this section, along with reasons for their inclusion. The query set comprising thirty words is displayed below:

abortion, alcoholism, architecture, bank, bicycling, biology, cheese, chemistry, complexity, computers, cruises, energy, fitness, football, games, golf, hiv, holiday, jaguar, java, law, nutrition, physics, recreation, shakespeare, shopping, sushi, terrorism, vol-

cano, workplace.

The queries are predominantly selected to provide a diverse range in query specificity. At one end of the spectrum, narrow specificity queries such as the query words ‘sushi’ and ‘cheese’ are introduced. For these query words the number of relevant pages is small and there may be comparitively little interconnectivity. At the other end of the spectrum, query words such as ‘shopping’ and ‘computers’ are introduced, with it suggested that the number of relevant pages will be significant, therefore providing cohesive communities large in size, with the potential for a large amount of interconnectivity. A number of queries fall between these two query types and are referred to as medium-specificity queries. These queries have a number of relevant pages closer to the average and therefore an intermediate cohesive-community size, with ‘football’ and ‘computers’ examples of two such queries.

Other query words containing unique characteristics are selected to provide a comprehensive evaluation of the link analysis ranking algorithms. One such characteristic is polysemy, a difficulty that must be surmounted by search engines and therefore meriting inclusion in the query-word set. Polysemy refers to a word with multiple meanings, an example of which is the word ‘Jaguar’. In this instance the word may be referring to the animal or to the car. The result of such a query is that the cohesive community will consist of both types of pages, therefore providing a unique link-structure. Another example is the word ‘bank’ where the reference may be to a reliance on somebody or something, a place to store savings, or the side of a river. A bi-polar query is also selected and occurs when the set of query-related pages have a biased stance in relation to the query. An example is the query word ‘abortion’, where the set of query-related pages will predominantly consist of pages that are either pro or anti abortion.

6.1.1 Ranking Similarities

Initially, we compare the PageRank and QD-PageRank orderings to obtain a Kendall-Tau value representative of result-set similarity. For each query, the QD-PageRank orderings are then compared with the result-set induced using EQD-Rank at a depth of one, two and three. Each result set examined is formed using a word from the lexicon, discussed in the previous section, with the similarity measure applied to the top one hundred results for each query. The Kendall-Tau values range from -1 to 1, with a value of -1 indicating no rank ordering similarity and a value of 1 signifying identical orderings.

Query Word	**PR**	**EQD-D1**	**EQD-D2**	**EQD-D3**
abortion	0.1811	0.6040	0.9810	0.9828
alcoholism	0.3777	0.6888	1.0000	1.0000
architecture	0.3194	0.6636	0.9848	0.9873
bank	0.4141	0.7662	0.9821	0.9884
bicycling	0.3349	0.8000	0.8000	0.8000
biology	0.2325	0.6563	0.9852	0.9882
cheese	0.1824	0.6276	0.9839	0.9878
chemistry	0.2222	0.6871	0.9877	0.9925
complexity	0.2199	0.5950	0.9854	0.9871
computers	0.4672	0.7311	0.9808	0.9891
cruises	0.1015	0.6172	0.9821	0.9901
energy	0.4560	0.7484	0.9820	0.9906
fitness	0.3586	0.7260	0.9863	0.9921
football	0.4452	0.7435	0.9820	0.9911
games	0.4784	0.7689	0.9816	0.9926
golf	0.3844	0.7076	0.9836	0.9906
hiv	0.1961	0.6259	0.9870	0.9885
holiday	0.3491	0.6883	0.9859	0.9910
jaguar	0.6618	0.7256	0.9807	0.9862
java	0.3169	0.6720	0.9821	0.9874
law	0.5184	0.7666	0.9774	0.9916
nutrition	0.3733	0.7001	0.9858	0.9914
physics	0.3432	0.6923	0.9856	0.9902
recreation	0.2896	0.6668	0.9879	0.9116
shakespeare	0.3040	0.6260	0.9831	0.9864
shopping	0.5334	0.7830	0.9784	0.9897
sushi	0.4286	0.3571	1.0000	1.0000
terrorism	0.2775	0.6102	0.9851	0.9897
volcano	0.1196	0.6099	0.9731	0.9873
workplace	0.2846	0.6796	0.9863	0.9883
Mean:	0.3333	0.6777	0.9783	0.9837

Table 6.1: The Kendall-Tau Correlation between the Result Sets

From table 6.1, little similarity between QD-PageRank and PageRank orderings (column 2) can be observed. For the majority of queries the Kendall-Tau value is less than 0.5, with an average value of 0.33, revealing a large amount of dissimilarly in the rank orderings. Comparison between EQD-Rank at a depth of one and QD-PageRank (column 3) reveals an increased similarity in rank orderings of on average 112.5%.

However, an average Kendall-Tau value of 0.677, reveals that a reasonable degree of dissimilarity between the two rank orders, still exists.

This dissimilarity is to a large extent eradicated with EQD-Rank at a depth of two, where the average Kendall Tau value is 0.978. For EQD-Rank at a depth of three, a further increase in the accuracy of the estimation is revealed, with the orderings near identical or identical in all cases and an average Kendall-Tau value of 0.984.

Analysis revealed that the greater the depth of in-links examined, the greater the estimation accuracy. However, as the number of layers examined increases, so does the computational cost and we therefore hypothesis that EQD-Rank at a depth equal to two provides a sufficient level of accuracy, without an extensive computational overhead.

6.1.2 User Study

To compare the relevance achieved using PageRank and EQD-Rank at a depth of two, we perform a user study involving six volunteers. Each volunteer is presented a results set containing the URLs of 20 web pages, composed from the top ten results achieved using these two algorithms, with the results randomly mixed. To return a result-set representative of each algorithm, a combination of context and content scores is used. Pagerank and EQD-Rank values are combined with TF*IDF scores and ranked accordingly. The merging of context and content based scores is an area of research on which little has been published to date. As this paper is, in many ways, a successor of [RD02], we scale each vector to have the same average value in the top ten terms, before adding the two vectors. Following composition of the URL set, volunteers are asked to assign a page rating, ranging from 1 to 4, with 1 signifying no relevance, 2 some relevance, but not a sufficient level, 3 signifying good relevance, and four signifying a very good level of relevance. The scores for each query are displayed in the table below:

Query Word	**PR**	**EQD-PR**	**Query Word**	**PR**	**EQD-PR**
abortion	141	152	*golf*	202	220
alcoholism	182	190	*hiv*	109	136
architecture	190	227	*holiday*	123	139
bank	201	229	*jaguar*	117	109
bicycling	169	182	*java*	211	232
biology	176	231	*law*	147	172
cheese	178	210	*nutrition*	141	181
chemistry	179	213	*physics*	202	231
complexity	195	176	*recreation*	112	162
computers	188	229	*shakespeare*	116	91
cruises	206	198	*shopping*	201	226
energy	161	223	*sushi*	116	105
fitness	189	205	*terrorism*	195	227
football	207	233	*volcano*	145	186
games	179	228	*workplace*	139	156
			Average	156.66	189.66

Table 6.2: User Study Results

From the data above, it can be observed that, on average, every user found EQD-Rank results more relevant for 25 out of the 30 queries. Overall, EQD-Rank led to an increase in relevance of 21.26%. On the few occasions PageRank performed better than EQD-Rank, it should be noted that the induced topic-communities were small in size, with little interconnectivity. In a real world implementation, when the induced community falls below a predefined threshold, it is suggested the standard PageRank values be applied, optionally with intercommunity links assigned more weight.

6.2 Conclusions

During this thesis numerous research areas are covered, leading to the formulation of a number of conclusions. The creation of a webpage corpus through a ‘crawler’ is initially examined. The numerous challenges in system design, I/O and network efficiency, and robustness and manageability imposed during the downloading over a million pages, are highlighted. Difficulties such as crawler traps, hosts containing multiple links to the same page, host-name aliases, and mirror web sites must also be overcome and are discussed. We then demonstrate that these difficulties are surmountable through the extensible and scalable ViperBot crawler and the resulting obtainment of a dataset in excess of one million pages.

Creation and storage of the transpose matrix is the second area of research examined. The dataset graph is represented as a very sparse bit-matrix, resulting in two problems. The naive representation of the matrix requires $O(n)^2$ amount of memory, with n the number of pages in the dataset. The resulting memory-requirement is a a terabit of storage space, resulting in the necessity of an alternate representation. In the area of inverted-index representation, case folding, stemming, and stop words are all considered. The storage of the resulting document IDs and word occurrences are also discussed, along with compression techniques such as Golumb coding.

A discussion of the PageRank, Query Dependent PageRank, and EQD-Rank ensues, highlighting various algorithmic traits and implementation details. As a result a number of important conclusions are reached and discussed. It is conjectured that that the distribution of PageRank and the In-Degree can be characterized as power law, with best fit exhibited for an exponent of 2.1. Despite the fact PageRank and In-Degree distributions exhibit a power law distribution, with an identical exponent, we postulate that a high In-Degree is not indicative of a high PageRank value and vica versa, with experiments revealing a correlation coefficient of 0.29. Also revealed, having implications for inverted index compression and available storage optimization, is that within the entire dataset 30% of the PageRank is obtained by the top 10 percent of pages, despite deviation at community level.

EQD-Rank is based on the premise that QD-PageRank provides an increase in result-set relevant when compared with PageRank, with it suggested in [RD02], that a relevancy increase of up to 34% is obtainable. It is demonstrated that EQD-Rank at a depth of two provides an extremely close approximation of the QD-PageRank vector, with a Kendall-Tau value of on average 0.98. The increase in relevance is also verified through a user study, with the increase greater than 20%.

6.3 Future Study

A rich body of potential research is uncovered during this thesis. The creation of an industrial strength crawling application, using ViperBot as a platform, is one area of future research. Issues to be addressed in this instance involve the bottleneck of host name resolution, the improvement of duplicate-page detection, the introduction of a load monitor, and the storage of URL queue buffers within memory. ViperBot could also be transformed into a distributed crawler with the repository spread over a number of storage servers.

In the area of transpose matrix and inverted-index storage, a number of modifications could also be introduced. With regards to the transpose matrix, delta encoding could be introduced storing the gaps between the inlink numbers, instead of the inlinks themselves. A compression technique based on Golumb coding could also be implemented to compress the inverted index to between 10 and 15% of the uncompressed index.

The current EQD-Rank implementation speed is another area of potential refinement, with much scope for increased speeds currently available through improved I/O and efficient datastructure implementation techniques. As a result it is anticipated that the computation speeds could be reduced to below five seconds, for a typical user query.

The culmination of these optimization techniques would pave the way for a full-scale web search engine. One billion plus pages would initially be downloaded using the new industrial-strength ViperBot. Offline processing would then ensue, through the calculation of the PageRank vector, along with the creation and storage of the compressed inverted index and transpose matrix. In realtime speeds the EQD-Rank algorithm would then be implemented, returning a result-set capable of competing with, and improving upon, the relevance achieved by the leading search engines.

6.4 Final Remarks

During this thesis numerous search-engine techniques are adopted, to facilitate the analysis of the EDQ-Rank result-set post-ranking refinement. The refinement, based on local graph-traversal provides an approximation of the query dependent PageRank vector, while avoiding the computational strain involved in the creation of the query-related subgraph. We therefore conjecture that EQD-Rank could be easily incorporated into the ranking process of a modern search engine and used to return result-sets, with relevance increases in excess of 20%.

Bibliography

[Ada99] Lada A. Adamic, *The small world web*, Proc. 3rd European Conf. Research and Advanced Technology for Digital Libraries, ECDL (S. Abiteboul and A.-M. Vercoustre, eds.), no. 1696, Springer-Verlag, 1999, pp. 443–452.

[BA99] Albert-Laszlo Barabasi and Reka Albert, *Emergence of scaling in random networks*, Science **286** (1999), 509.

[BBH$^+$98] Krishna Bharat, Andrei Broder, Monika Henzinger, Puneet Kumar, and Suresh Venkatasubramanian, *The connectivity server: fast access to linkage information on the web*, WWW7: Proceedings of the seventh international conference on World Wide Web 7 (Amsterdam, The Netherlands), Elsevier Science Publishers B. V., 1998, pp. 469–477.

[BCSV02] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, *Ubicrawler: A scalable fully distributed web crawler*, In Proc. AusWeb02. The Eighth Australian World Wide Web Conference, 2002.

[BGMZ97] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig, *Syntactic clustering of the web*, Selected papers from the sixth international conference on World Wide Web (Essex, UK), Elsevier Science Publishers Ltd., 1997, pp. 1157–1166.

[BH98] Krishna Bharat and Monika R. Henzinger, *Improved algorithms for topic distillation in a hyperlinked environment*, Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval (Melbourne, AU), 1998, pp. 104–111.

[BKM$^+$00] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener, *Graph structure in the Web*, Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking (Amsterdam, The Netherlands), North-Holland Publishing Co., 2000, pp. 309–320.

[BM01] Krishna Bharat and George A. Mihaila, *When experts agree: using non-affiliated experts to rank popular topics*, World Wide Web, 2001, pp. 597–602.

[BMPW98] Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd, *What can you do with a Web in your Pocket?*, Data Engineering Bulletin **21** (1998), no. 2, 37–47.

[BP98] Sergey Brin and Lawrence Page, *The anatomy of a large-scale hypertextual Web search engine*, Computer Networks and ISDN Systems **30** (1998), no. 1–7, 107–117.

[BRRT01] Alan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas, *Finding authorities and hubs from link structures on the World Wide Web*, World Wide Web, 2001, pp. 415–429.

[Bur97] M. Burner, *Crawling towards eternity - building an archive of the world wide web*, Web Techniques, 1997, p. 2(5).

[Bus45] Vannevar Bush, *As we may think*, The Atlantic Monthly **176** (1945), no. 1, 101–108.

[BV03a] P. Boldi and S. Vigna, *The WebGraph framework I: Compression techniques*, Technical Report 293-03, Universit di Milano, Dipartimento di Scienze dell'Informazione, 2003., 2003.

[BV03b] Paolo Boldi and Sebastiano Vigna, *The webgraph framework ii: Codes for the world wide web*, Technical Report 294-03, Universit di Milano, Dipartimento di Scienze dell'Informazione, 2003., 2003.

[BWB02] Michael W. Berry, P. Wang, and J. Bownas, *Website query analysis: trend and behaviour detection*, Second SIAM conference on Data Mining, 2002.

[BYCMR05] Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez, *Crawling a country: better strategies than breadth-first for web page ordering*, WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web (New York, NY, USA), ACM Press, 2005, pp. 864–872.

[CC00] David Cohn and Huan Chang, *Learning to Probabilistically Identify Authoritative Documents*, Proc. 17th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 2000, pp. 167–174.

[CDR$^+$98] Soumen Chakrabarti, Byron Dom, Prabhakar Raghavan, Sridhar Rajagopalan, David Gibson, and Jon Kleinberg, *Automatic resource compilation by analyzing hyperlink structure and associated text*, WWW7: Proceedings of the seventh international conference on World Wide Web 7 (Amsterdam, The Netherlands, The Netherlands), Elsevier Science Publishers B. V., 1998, pp. 65–74.

[CGM00] Junghoo Cho and Hector Garcia-Molina, *The Evolution of the Web and Implications for an Incremental Crawler*, Proceedings of the Twenty-sixth International Conference on Very Large Databases, 2000.

[CGMP98] Junghoo Cho, Hector García-Molina, and Lawrence Page, *Efficient crawling through URL ordering*, Computer Networks and ISDN Systems **30** (1998), no. 1–7, 161–172.

[CGS] Yen-Yu Chen, Qingqing Gan, and Torsten Suel, *I/O-Efficient Techniques for Computing Pagerank*, In Proc. of the 11th International Conf. on Information and Knowledge Management, pp. 549–557.

[Cle97] Cyril Cleverdon, *The cranfield tests on index language devices*, Readings in information retrieval (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1997, pp. 47–59.

[CP90] Doug Cutting and Jan Pedersen, *Optimizations for dynamic inverted index maintenance*, Proceedings of the 13th International ACM SIGIR Conference on Research and Development in Information Retrieval, 1990, pp. 405–411.

[CPKT92] Douglass R. Cutting, Jan O. Pedersen, David Karger, and John W. Tukey, *Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections*, Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1992, pp. 318–329.

[CRL$^+$03] G. Caldarelli, P. De Los Rios, L. Laura, S. Leonardi, and S. Millozzi, *A study of stochastic models for the Web*, Tech. report, dipartimento di Informatica e Sistemistica, Universita' di Roma, Technical Report, 2003.

[CvdBD99] Soumen Chakrabarti, Martin van den Berg, and Byron Dom, *Focused crawling: a new approach to topic-specific Web resource discovery*, Computer Networks (Amsterdam, Netherlands: 1999) **31** (1999), no. 11–16, 1623–1640.

[Dav00] Brian D. Davison, *Topical locality in the web*, SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), ACM Press, 2000, pp. 272–279.

[DDL$^+$90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman, *Indexing by Latent Semantic Analysis*, Journal of the American Society of Information Science **41** (1990), no. 6, 391–407.

[DFL$^+$88] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman, *Using latent semantic analysis to improve access to textual information*, CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems (New York, NY, USA), ACM Press, 1988, pp. 281–285.

[DKM$^+$01] Stephen Dill, S. Ravi Kumar, Kevin S. McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins, *Self-similarity in the Web*, The VLDB Journal, 2001, pp. 69–78.

[EM] Nadav Eiron and Kevin S. Mccurley, *Locality, Hierarchy, and Bidirectionality in the Web*, In Workshop on Algorithms and Models for the Web Graph, Budapest.

[EMT04] N. Eiron, K. McCurley, and J. Tomlin, *Ranking the web frontier*, In Proceedings of the 13th conference on World Wide Web, 2004, pp. 309–318.

[FKS03] R. Fagin, R. Kumar, and D. Sivakumar, *Comparing top k lists*, In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms., 2003.

[FLGC02] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee, *Self-Organization and Identification of Web Communities*, Computer **35** (2002), no. 3, 66–71.

[FMN03] D. Fetterly, M. Manasse, and M. Najork, *The Evolution of Clusters of Near-Duplicate Web Pages*, In 1st Latin American Web Congress, Nov. 2003., 2003.

[Fos82] A. C. Foskett, *The subject approach to information (4th ed.)*, Facet Publishing, 1982.

[GL83] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins University Press, Baltimore, Maryland, 1983.

[GS05] A. Gulli and A. Signorini, *The Indexable Web is More than 11.5 Billion Pages*, 2005.

[Har93] Donna Harman, *Overview of the first trec conference*, SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), ACM Press, 1993, pp. 36–47.

[Hav99] Taher Haveliwala, *Efficient Computation of PageRank*, Tech. report, Stanford University Technical Report, 1999.

[Hav02] T. Haveliwala, *Topic-Sensitive PageRank*, Proceedings of the Eleventh International World Wide Web Conference, Honolulu, Hawaii, 2002.

[Hen03] M. Henzinger, *Hyperlink analysis on the web*, 2003, available online at http://www-cad.eecs.berkeley.edu/ tah/170/Notes/170-google.ppt.

[HHMN99] Monika R. Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork, *Measuring index quality using random walks on the Web*, Computer Networks (Amsterdam, Netherlands: 1999) **31** (1999), no. 11–16, 1291–1303.

[HN99] Allan Heydon and Marc Najork, *Mercator: A scalable, extensible web crawler*, World Wide Web **2** (1999), no. 4, 219–229.

[HN00] ______, *Performance limitations of the Java core libraries*, Concurrency: Practice and Experience **12** (2000), no. 6, 363–373.

[Hof99] Thomas Hofmann, *Probabilistic Latent Semantic Analysis*, Proc. of Uncertainty in Artificial Intelligence, UAI'99 (Stockholm), 1999.

[Jon72] K. S. Jones, *A statistical interpretation of term specificity and its application in retrieval*, Journal of Documentation, 1972, pp. 28:11–21.

[JW03] Glen Jeh and Jennifer Widom, *Scaling personalized web search*, WWW '03: Proceedings of the 12th international conference on World Wide Web (New York, NY, USA), ACM Press, 2003, pp. 271–279.

[Ken38] Maurice G. Kendall, *A new measure of rank correlation*, Biometrika, 1938, pp. 30(1–2):81–93.

[KHG03] S. Kamvar, T. Haveliwala, and G. Golub, *Adaptive methods for the computation of pagerank*, Tech. report, Stanford University Technical report., 2003.

[KHMG] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, *Extrapolation methods for accelerating PageRank computations*.

[KHMG03] ______, *Exploiting the block structure of the web for computing PageRank*, Tech. report, Stanford University Technical Report, 2003., 2003.

[Kle99] Jon M. Kleinberg, *Authoritative sources in a hyperlinked environment*, Journal of the ACM **46** (1999), no. 5, 604–632.

[Kos] M. Koster, *Guidelines for robot writers*, available online at http://www.robotstxt.org/wc/guidelines.html.

[KRRT99] S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins, *Extracting large-scale knowledge bases from the web*, The VLDB Journal, 1999, pp. 639–650.

[LG00] Steve Lawrence and C. Lee Giles, *Accessibility of information on the web*, Intelligence **11** (2000), no. 1, 32–39.

[LM00] R. Lempel and S. Moran, *The stochastic approach for link-structure analysis (SALSA) and the TKC effect*, Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications networking (Amsterdam, The Netherlands, The Netherlands), North-Holland Publishing Co., 2000, pp. 387–401.

[Luh57] H. P. Luhn, *A statistical approach to mechanized encoding and searching of literary information*, IBM Journal of Research and Development, 1957.

[MB98] Robert C. Miller and Krishna Bharat, *Sphinx: a framework for creating personal, site-specific web crawlers*, WWW7: Proceedings of the seventh international conference on World Wide Web 7 (Amsterdam, The Netherlands), Elsevier Science Publishers B. V., 1998, pp. 119–130.

[McB94] O. A. McBryan, *Genvl and wwww: Tools for taming the web*, In First International Conference on the World Wide Web, 1994, pp. 313–323.

[Men02] F. Menczer, *Growing and navigating the small world web by local content*, Proceedings of the National Academy of Sciences, 2002, pp. 99(22):14014–14019.

[MK60] M.E. Maron and J.L. Kuhns, *On relevance, probabilistic indexing and information retrieval*, Journal of the ACM, 1960, pp. 216–244.

[NCO04] A. Ntoulus, 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 Thirteenth International World Wide Web Conference, 2004.

[NW01] Marc Najork and Janet L. Wiener, *Breadth-first crawling yields high-quality pages*, WWW '01: Proceedings of the 10th international conference on World Wide Web (New York, NY, USA), ACM Press, 2001, pp. 114–118.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Tech. report, Stanford Digital Library Technologies Project, 1998.

[PFL$^+$02] D. Pennock, G. Flake, S. Lawrence, E. Glover, and C. Giles, *Winners Don't Take All: Characterizing the Competition for Links on the Web*, Proceedings of the National Academy of Sciences **99** (2002), no. 8, 5207–5211.

[Pie98] V. Piek, *Introduction to eurowordnet*, Computers and the Humanities, 1998, pp. 32(2–3):73–89.

[Por] M. F. Porter, *An algorithm for suffix stripping*, Program: Automated Library and Information Systems.

[PRU02] Gopal Pandurangan, Prabhakara Raghavan, and Eli Upfal, *Using PageRank to Characterize Web Structure*, 8th Annual International Computing and Combinatorics Conference (COCOON), 2002.

[Rab81] M. O. Rabin, *Fingerprinting by random polynomials*, Harvard University Technical Report TR-15-81, 1981.

[Ram98] R. Ramakrishnan, *Database management systems*, MxGraw-Hill, 1998.

[Ran49] Rankdex, *The Rankdex search engine*, 1949, available online at http://rankdex.gari.com/.

[RD02] Mathew Richardson and Pedro Domingos, *The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank*, Advances in Neural Information Processing Systems 14, MIT Press, 2002.

[RK58] J. Rees and A. Kent, *Mechanised searching experiments using the wru seacrhing selector*, American Documentation, 1958, pp. 9(4):277–303.

[RM] Davood Rafiei and Alberto Mendelzon, *What is this Page Known for? Computing Web Page Reputations*, In Proc. 9th World Wide Web Conference, Amsterdam.

[Rob75] S. E. Robertson, *The probabilistic ranking principle in ir*, Journal of Documentation, 1975, pp. 33:294–304.

[Roc71] J. J. Roccio, *Relevance feedback in information retrieval*, In G. Salton, editor, The SMART Retrieval System: Experiments in Automatic Document Processing, 1971, pp. 313–323.

[RSWW01] K. Randall, R. Stata, R. Wickremesinghe, and J. Wiener, *The link database: Fast access to graphs of the Web*, Research Report 175, Compaq Systems Research Center, Palo Alto, CA, 2001., 2001.

[SB88] Gerard Salton and Chris Buckley, *Term-weighting approaches in automatic text retrieval*, Information Processing and Management, 1988, pp. 24(5):5133–523.

[See49] J. R. Seely, *The net of reciprocal influence. a problem in treating sociometric data*, Canadian Jornal of Psychology, 1949, pp. 3:234–240.

[SHMM98] Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael Moricz, *Analysis of a very large altavista query log*, Tech. Report 1998-014, Digital SRC, 1998.

[SL68] G. Salton and M. E. Lesk, *Computer evaluation of indexing and text processing*, J. ACM **15** (1968), no. 1, 8–36.

[SS02] Vladislav Shkapenyuk and Torsten Suel, *Design and Implementation of a High-Performance Distributed Web Crawler*, In Proceedings of the IEEE International Conference on Data Engineering,February 2002, 2002.

[SWY71] Gerard Salton, A. Wong, and C. S. Yang, *A vector space model for automatic indexing*, Communications of the ACM, 1971, pp. 18(11):613–620.

[SWYZ02] F. Scholer, H. Williams, J. Yiannis, and J. Zobel, *Compression of inverted indexes for fast query evaluation*, ACM SIGIR conference on research and development in information retrieval, 2002, pp. 222–229.

[SY01] Torsten Suel and Jun Yuan, *Compressing the Graph Structure of the Web*, Data Compression Conference, 2001, pp. 213–222.

[Wan] Ziyang Wang, *Improved Link-Based Algorithms for Ranking Web Pages*, citeseer.ist.psu.edu/651305.html.

[Wan03] Z. Wang, *Improved link-based algorithms for ranking web pages*, NYU Computer Science Technical Report TR2003-846, 2003.

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell, *Managing gigabytes (2nd ed.): compressing and indexing documents and images*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[Zip49] G. K. Zipf, *Human behavior and the principle of least-effort*, 1949.