

UA



Learning Lab Lower Saxony [L3S]

TEA September, 2003  
Universidad Autónoma de Madrid

# Finding Hubs for Personalised Web Search

Different ranks to different users

Daniel Olmedilla  
olmedilla@learninglab.de  
September 8, 2003

---

Copyright © 2005 Daniel Olmedilla.

Typeset by the author with the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Documentation System.

All rights reserved. No part of this work may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior permission.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>State of the Art</b>	<b>8</b>
2.1	The World Wide Web . . . . .	10
2.2	Crawlers and Search Engines . . . . .	11
2.2.1	Information Retrieval . . . . .	12
2.2.2	Webbase . . . . .	15
2.3	Ranking algorithms . . . . .	19
2.3.1	Notation . . . . .	19
2.3.2	PageRank . . . . .	20
2.3.3	HITS . . . . .	24
2.3.4	SALSA . . . . .	28
2.3.5	Randomized HITS . . . . .	35
2.3.6	Subspace HITS . . . . .	36
2.3.7	SimRank . . . . .	37
2.3.8	Other algorithms . . . . .	39
2.4	Personalization . . . . .	43
2.4.1	Personalized PageRank . . . . .	44
2.4.2	Topic-sensitive PageRank . . . . .	44
2.4.3	Scaling Personalized Web Search . . . . .	46
<b>3</b>	<b>Our work</b>	<b>52</b>
3.1	Webbase . . . . .	53
3.2	Ranking algorithms . . . . .	54
3.3	HubRank . . . . .	55
3.4	Proxy . . . . .	56
3.5	HubFinder . . . . .	56
3.6	Personalized Ranks . . . . .	57
3.7	Summary of the process . . . . .	58

<b>4</b>	<b>Results</b>	<b>60</b>
4.1	Webbase . . . . .	60
4.2	HubRank . . . . .	61
4.2.1	Small Graph . . . . .	61
4.2.2	Big crawl . . . . .	64
4.2.3	Conclusion . . . . .	67
4.3	HubFinder . . . . .	67
4.4	Personalized Ranks . . . . .	69
<b>5</b>	<b>Conclusions and Further work</b>	<b>71</b>

# List of Figures

2.1	Web's macroscopic structure . . . . .	10
2.2	Typical Search Engine Structure . . . . .	13
2.3	Typical Crawler Structure . . . . .	14
2.4	Webbase Architecture . . . . .	16
2.5	WebBase index factory diagram . . . . .	18
2.6	Small Subgraph . . . . .	21
2.7	PageRank Example . . . . .	23
2.8	Search with keywords "airline company" in Google . . . . .	25
2.9	Algorithm to construct a focused subgraph of the WWW . . . . .	27
2.10	HITS algorithm . . . . .	29
2.11	Filter algorithm . . . . .	29
2.12	HITS example . . . . .	30
2.13	SALSA example . . . . .	34
2.14	Randomized HITS example . . . . .	36
2.15	Small Web graph $G$ . . . . .	39
2.16	Node-pairs graph $G^2$ . . . . .	41
2.17	Construction of hub vectors . . . . .	49
3.1	Algorithm to find important pages to a initial subset . . . . .	57
4.1	HubRank Example . . . . .	61

## List of Tables

2.1	PageRank Ranking . . . . .	24
2.2	HITS Ranking . . . . .	31
2.3	SALSA Ranking . . . . .	33
2.4	Randomized HITS Ranking . . . . .	37
2.5	SimRank scores . . . . .	40
2.6	Topics used in Topic-Sensitive PageRank . . . . .	45
4.1	Score comparison among algorithms for the graph of figure 2.6	62
4.2	Rank comparison between algorithms for the graph of figure 2.6	63
4.3	Scores from the ranking algorithms over the whole graph . . .	65
4.4	Scores from the ranking algorithms over the whole graph . . .	66
4.5	Bookmarks set . . . . .	68
4.6	HubFinder comparison with different algorithms . . . . .	68
4.7	Personalized Ranks . . . . .	70

# Chapter 1

## Introduction

The last years, the web has evolved rapidly. It is getting bigger and bigger really fast. The amount of information that Internet provides to users is becoming unmanageable. People who search for information waste a lot of time in the process. Which are the reasons? The reasons are that there is too much information interesting for a user and the current systems are not good enough selecting it according to the user's needs.

I think all of us who have worked with a computer have used one of the current existing search engines. Many times they seem to be really useful (and they are) but many other times we do not find what we were searching for. Even in those times where we find something interesting we have had to pass through a slow and time costly process. This process is not to think about the query we want to submit. This process is not the time needed by the search engine to give us an answer. This process is the filtering and selection of the pages returned by a search engine as a result of our query in order to find those pages that are really interesting to us. Many times this is a really slow process and many times it requires several iterations where we refine our query and submit it again to the search engine and we start again to check all the results returned.

We do not think this process is not useful. Of course it works most of the times but we think it can be improved. How can it be improved? Let me describe roughly how the process works. A search engine receives our query, it processes the query, it search into its index for relevant documents (documents that are likely related to our query and supposed to be interesting to us), then the search engine ranks the documents found relevant and it shows us the results. Where is the big weakness of this process? In our honest opinion we think that there two points where this process can be improved:

- First, how do we define the relevance between documents? Current approaches use basically text matching as a solution for this task. However, these approaches have many disadvantages. Let me give you an example. If someone gives you a query based on some keywords, do you think you will be able to infer what is he talking about? If I told you I want information about “windows” would you be able to infer if I am talking about the operating system Windows or if I am talking about the windows of a house? If I asked you about “java” would you be able to know if I am referring to the programming language, to an island or to coffee? Those are just examples and we think that the best approach to solve them is giving semantic information to the content, so then I would give some keywords and I would say what I mean with those keywords. However, the *Semantic Web* is still too far to be a valid approach for the whole World Wide Web.
- Second, we think that the current methods and algorithms to rank the results given by a search engine are not the very best. We think that they can be optimised in many ways. We claim that the best way to optimise them is to know and include information about the user. If we know the user interests we can filter the results for him automatically instead of letting him do it and waste his time. Many different ways exist to get information about the user like taking the history of the queries submitted by the user into account (so if we know the user has search for information related to the computer science field we can infer that with “windows” and “java” he is probably referring to the operating system and the programming language) or asking the user to give information about him (called user profile) where we can find useful information (if he is a computer science engineer for instance).

In this document we will focus on the second point discussed above and mainly we will try to get some basic information about the user analysing his user behaviour when he surfs the internet and asking him to provide some pages he thinks are the most important pages for him (like his bookmarks). With this information we claim we can provide better ranking for each user (a personalised ranking what means a different ranking for each user based on the user’s preferences and interests) that the one current search engines provide. Our approach claims to reduce dramatically the time spent by the user when he searches for information.

In chapter 2 I will introduce the reader into the existing and current work on these topics. A basic introduction about the Information Retrieval problem is given as well as some of the most important algorithms used to

rank the web results in current web search engines. In chapter 3 I present our work and how we claim to improve the existing work. Chapter 4 presents some of the experiments we realized with our new ideas and algorithms and the results we received. Finally, in chapter 5 I present the conclusions and talk about our next steps.

## Chapter 2

# State of the Art

In this section previous work involved in data mining and information retrieval is described. This work can be considered as the basis of our research. Sometimes because we are using directly some of the theorems or algorithms presented on it, because we use its results to compare ours or simply because it introduced some new ideas which help us to inspire and decide our own work.

Talk about data mining or information retrieval in the context of the World Wide Web (WWW) means mainly talk about crawlers and web search engines. Due the huge amount of information that exists and its distributed nature, these web search engines have become essential for users in order to find what they search. Currently, all the people who has ever navigated in Internet is familiarized with the most common and known web search engines like Google [Goo], Altavista [Alt], InfoSeek [Inf], Excite [Exc], Lycos [Lyc] or Yahoo! [Yah]. However, almost none of them know about what is hidden behind the user interface this web search engines present to them. There is a long and complicated process that permits a web search engine knows about the information existing in the whole WWW.

This process can be divided in the following tasks:

1. **Crawling:**

A crawler is in charge of visiting as many pages it can and retrieve the information needed from them. The idea is that this information is stored for the use by the search engine afterwards.

2. **Indexing:**

The information provided by a crawler has to be stored in order to be accessed by the search engine. As the user will be in front of his computer waiting for the answer of the search engine, time response

becomes an important issue. That is why this information is indexed in order to decrease the time needed to look into it.

### 3. **Searching:**

The web search engine represents the user interface needed to permit the user to query the information. It is the connection between the user and the information repository.

### 4. **Sorting/Ranking**

Due to the huge amount of information existing in the web when a user sends a query about a general topic (e.g. java course), there will exist an incredible number of pages related to this query (you can try to do so in Google for example). Of course only a small part of such amount of information will be really interesting for the user. That is why current search engines incorporate ranking algorithms in order to sort the results. That way the user will have the most important results first.

In the following sections I will try to give an overview of the work done over the topics related to our work. In section 2.1 I describe several properties of the World Wide Web. Section 2.2 gives a basic introduction to crawlers and web search engines as well as the Information Retrieval process in general. I will present the project that is developing the crawler we have modified in our work (WebBase Project [ACGM<sup>+</sup>00, HRGMP00]). In section 2.3 I will present the most important ranking algorithms developed. I will focus on:

1. **PageRank:** because of its proved importance and usefulness in Google.
2. **HITS:** shown to be the other most important ranking algorithm. The first one is PageRank.
3. **SALSA:** improvement over HITS. It behaves better against some properties of web graphs.
4. **Randomized HITS:** several problems of HITS make it unstable and Randomized HITS is a new version that makes use of the stability of PageRank to improve it.
5. **Subspace HITS:** another possibility to solve the unstability of HITS generalizing the result.
6. **SimRank:** a new algorithm to measure the similarity of pages based on their link structure.

In section 2.4 I present some existing approaches to personalise the ranking algorithms in order to give different ranks to different users.

---

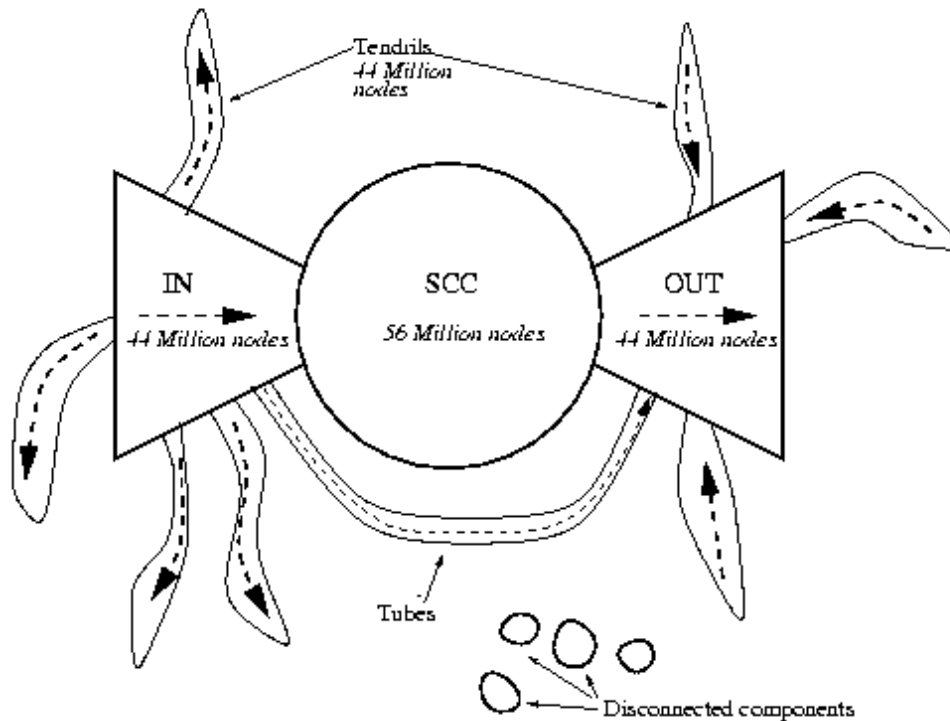


Figure 2.1: Web's macroscopic structure

## 2.1 The World Wide Web

Before we start to talk about anything else I think it would be really valuable to talk a little bit about the properties of the source of information we are dealing with: the World Wide Web. As we already know it is the union of a large amount of sources distributed and interconnected around the whole world. What not all the people know is that this apparently heterogeneous source of information has some properties which I will explain in this section.

In the figure 2.1 is depicted the current structure of the World Wide Web and its different components. This figure has been extracted from the experiments developed in [BKM<sup>+</sup>00] in May 1999. Let me describe each of the four components of this connected web:

- **Strongly Connected Component (SCC):** all the pages in this set can reach one another along directed links.
- **IN:** pages that can reach the SCC but cannot be reached from it (for example new sites that people have not yet discovered and linked to).

- **OUT:** pages that are accessible from the SCC but do not link back to it (for example corporate websites that contain only internal links).
- **TENDRILS:** pages that cannot reach the SCC, and cannot be reached from the SCC.

Perhaps the most surprising fact is that the size of the SCC is relatively small – it comprises about 56 million pages. Each of the other three sets contains about 44 million pages (thus, all four sets have roughly the same size).

Some other properties are that the directed diameter of SCC is at least 28 and likewise the maximum finite shortest path length is at least 503.

## 2.2 Crawlers and Search Engines

First of all, let me remember that the size of the whole World Wide Web is very large. In July 2000, it was estimated to contain about 2.1 billions vertices (pages) and 15 billions edges (links between pages) [MM00, LW01]. Moreover, about 7.3 millions pages are added every day, and many others are modified or removed [GL]. In another source I found in [LV00] (some others can be found in [Wha]) the following: *the world produces between 1 and 2 exabytes of unique information per year, which is roughly 250 megabytes for every man, woman, and child on earth. An exabyte is a billion gigabytes, or  $10^{18}$  bytes. Printed documents of all kinds comprise only .03% of the total.* This rapid evolution and exponential increase present a real challenge to web search engines because of several reasons:

### Accessibility

Not all the pages and all the information are accessible. We need a start point where we search for new pages. However, not all the pages are interconnected so our crawlers will never visit some of them.

### Crawling

A crawler is only a program that retrieves Web pages, commonly for use by a search engine [Pin94]. It navigates in the World Wide Web and jumps from one page to another retrieving information. This process takes time and periodically the crawler has to revisit the already crawled pages in order to maintain updated the information. Even if we have many crawlers working in parallel, we will not be able to visit all the available pages because we do not have unlimited resources (disk space, memory, time, bandwidth, etc.).

### Storage

Crawlers retrieve information from the pages they visit but this information has to be stored in our own repository. This repository has limited size.

### Information Management

Even if we reach to store all the information of the World Wide Web, manage and manipulate this information is a big challenge.

In the following sections I will try to give some basic knowledge about this topic and present some basic features of the World Wide Web.

## 2.2.1 Information Retrieval

In this section I describe the basis of the Information Retrieval. Most of the concepts here described have been extracted from [VR79].

The problem of the information retrieval has always existed. In the past, vast amount of information were stored and to access this information accurately and speedily was really difficult. After the appearance of the computers this problem remains unsolved.

Imagine a store of documents. Imagine a user who creates a query and sends it to the system. He could retrieve the right answer reading all the documents in the store, filtering the documents he is interested in and discarding the rest. This is the “perfect” retrieval but of course this is impracticable. As it is stated in [VR79] “The purpose of an automatic retrieval strategy is to retrieve all the *relevant* documents at the same time retrieving as few of the *non-relevant* as possible”.

After this definition it seems to be necessary to define two new concepts:

**Efficiency:** it is normally defined in terms of computer resources used such as core, backing store and C.P.U. time.

**Effectiveness:** it is usually measured in terms of *precision* and *recall*.

Precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved.

Recall is the ratio of the number of relevant documents retrieved to the total number of relevant documents (both retrieved and not retrieved).

To finish this introduction to the Information Retrieval I would like to show two typical and simple examples of two of the most important components of the current Web Search Engines: a Search Engine and a Crawler.

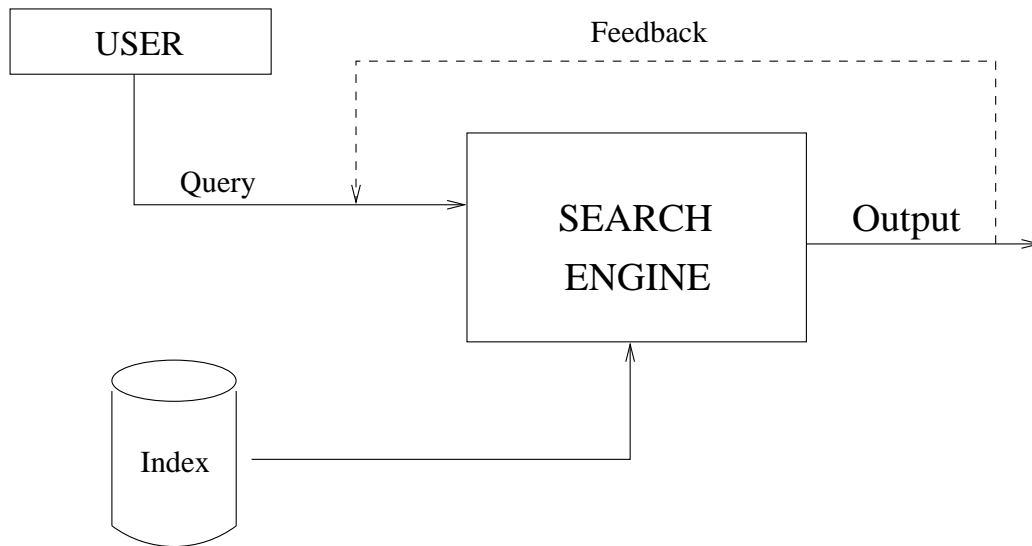


Figure 2.2: Typical Search Engine Structure

In the 2.2 a typical Information Retrieval System is sketched. It is composed of several basic components:

- **User Interface:** needed to take the user query and make it user friendly. The whole and complex process behind should be transparent to the user.
- **Index/Repository:** database/repository with the data to be searched. Here a representation of each document is stored. It is not needed to have the whole document in the repository, many of the current information systems store only part of it (e.g. significant words, metadata, etc.).
- **Search Module:** here is the processor of the system. Some of the tasks it is in charge are:
  - transform query to understandable format (e.g. from natural language to the computer understandable format)
  - do matching with the index. In this case matching means “find relevant documents” because many times the match is not exact but the documents are still relevant to the user query (e.g. synonyms).
  - return the relevant documents as output. How to present the output and the order could be some tasks of this module.

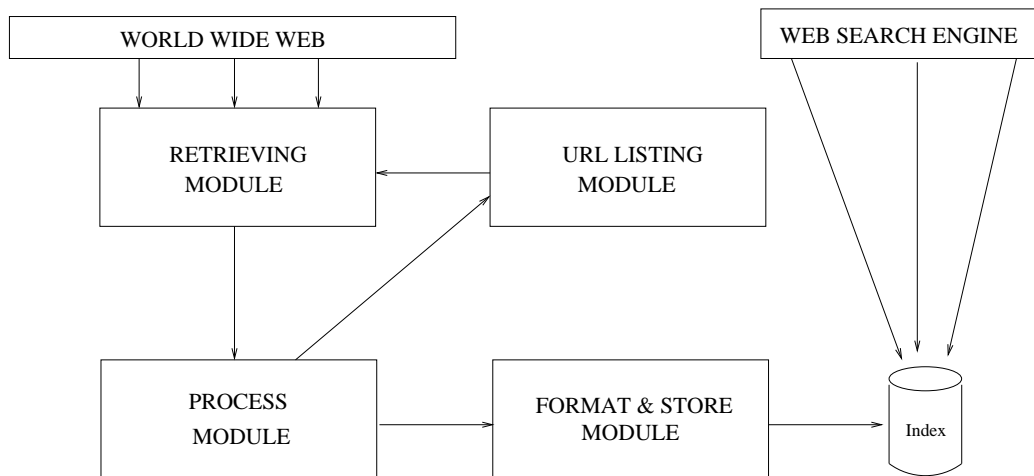


Figure 2.3: Typical Crawler Structure

- **Evaluation Module:** takes care of the user behaviour with the results of the query. Many times systems can track what a user do after a transaction and take it into account in future interactions. For example, if a system shows some results found relevant to a query and the user refines the query and searches it again, the system can suppose that the previous results were unsatisfactory and it can remove them from the next output.

In the figure 2.3 a typical crawler is sketched. Each module has different functions and communicates with each others as follows:

- **Retrieving Module:** retrieves each document from the web and gives it to the Process module
- **URL Listing Module:** has a list (ordered or unordered) of URLs and gives them to the Retrieving module. As it is shown in [CGMP98] a good technique to sort this list become crucial.
- **Process Module:** realizes the following tasks over the document:
  - *Automatic text analysis:* stemming, removal of high frequency words or detecting equivalent stems are some examples.
  - *Indexing:* extract of index terms from the document. An *index language* is the language used to describe documents and requests.
  - *Classification:* use of thesauri, keyword clustering, document clustering ...

- *Filtering*: it is possible to filter the documents we are examining in order to store only some of them (regarding to the topic they belong to, the format they have, etc.).
- *Link extraction*: all the links a document has are extracted and sent to the URL Listing module in order to be crawled in the future.

After these tasks, it gives the result (data) to the Format and Store module.

- **Format and Store Module**: converts data to an improved format and store it into the index.
- **Index/Repository**: database/repository with the useful data retrieved. It will be used by the Search Engine.

A simple implementation of these structures of a search engine and a crawler can be found at [Sim].

### 2.2.2 Webbase

For our work we have modified and used the WebBase crawler [ACGM<sup>+</sup>00, HRGMP00] (our work on this crawler is described in the section 3.1). This crawler is being developed in the context of the Stanford WebBase project [Sta]. The Stanford WebBase project is investigating various issues in crawling, storage, indexing, and querying of large collections of Web pages. The project builds on the previous Google [Goo] activity that was part of the DLI1 [Diga] initiative. The DLI2 WebBase project [Digb] aims to build the necessary infrastructure to facilitate the development and testing of new algorithms for clustering, searching, mining, and classification of Web content.

The WebBase project has the following goals:

- Provide a storage infrastructure for Web-like content
- Store a sizeable portion of the Web
- Enable researchers to easily build indexes of page features across large sets of pages
- Distribute Webbase content via multicast channels
- Support structure and content-based querying over the stored collection

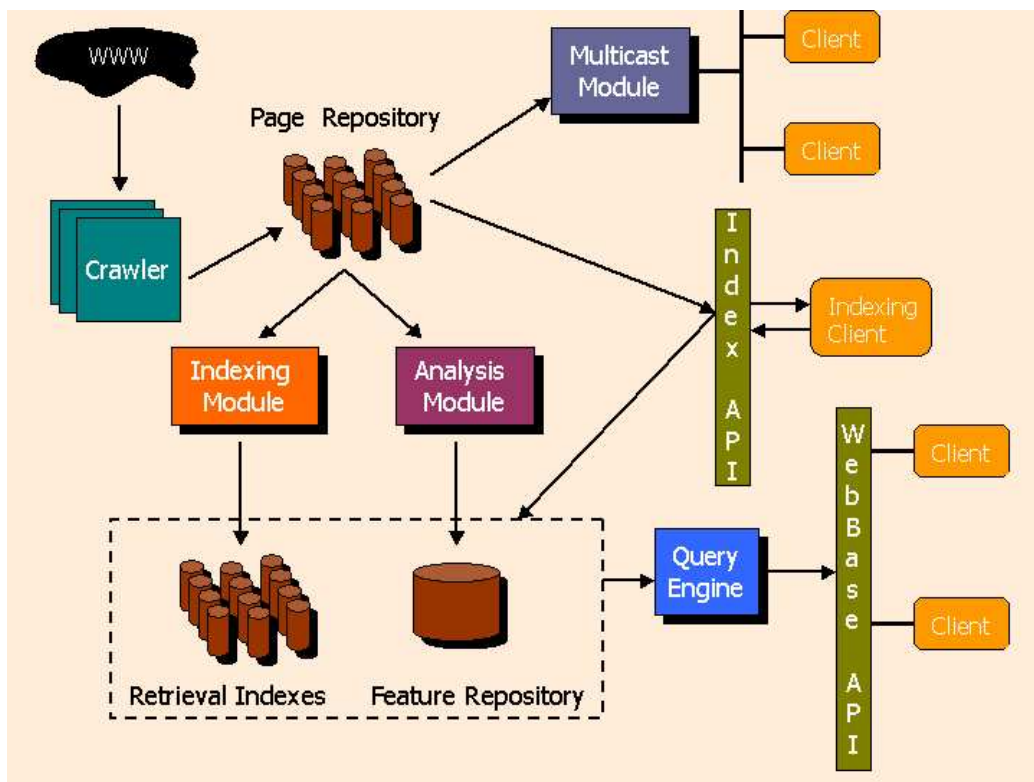


Figure 2.4: Webbase Architecture

The WebBase project employs the architecture shown in Figure 2.4. The important components of this architecture are described below.

- **Crawler:** "Smart" crawling technology is used to crawl "valuable" sites more frequently or more deeply. The measure of "value" is, of course, itself an important research topic. Smart crawling is also used to estimate the rate of change of web pages and adjust the crawling algorithm to maximize the *freshness* of the pages in the repository.
- **Page repository:** a scalable distributed repository is used to store the crawled collection of Web pages. Strategies for physical organization of pages on the storage devices, distribution of pages across machines, and mechanisms to integrate freshly crawled pages, are important issues in the design of this repository. The repository supports both random and stream-based access modes. Random access allows individual pages to be retrieved based on an internal page identifier. Stream-based access allows all or a significant subset of pages to be retrieved as a stream.
- **Indexing and Analysis Modules:** the indexing module uses a stream of pages from the repository to build basic retrieval indexes over the content and structure of the crawled collection. Indexes on the content are akin to standard IR-style text indexes, whereas indexes on the structure represent the hyper linked Web graph. Efficient construction of such indexes over Web-scale collections poses interesting challenges. The analysis module populates the feature repository by using a collection of *feature extraction engines*. Each feature extraction engine computes some property of a page (example: reading level, genre), builds an index over that property, and makes that index available to the rest of WebBase. These indexes will be kept up to date as the archive is refreshed through new crawls. The set of feature extraction engines will grow, as new algorithms are devised to describe and characterize Web pages.

In addition, indexes can also be built off-site by indexing clients that implement the "Index API". These indexing clients receive a page stream, use that to build the index files, and integrate the index into WebBase using the "Index API."

- **Multicast module:** the distribution machinery implemented by the multicast module will allow researchers everywhere to take advantage of our collected data. Rather than forcing all index creation and data analysis to run at the site where the data is located, our wide-area data

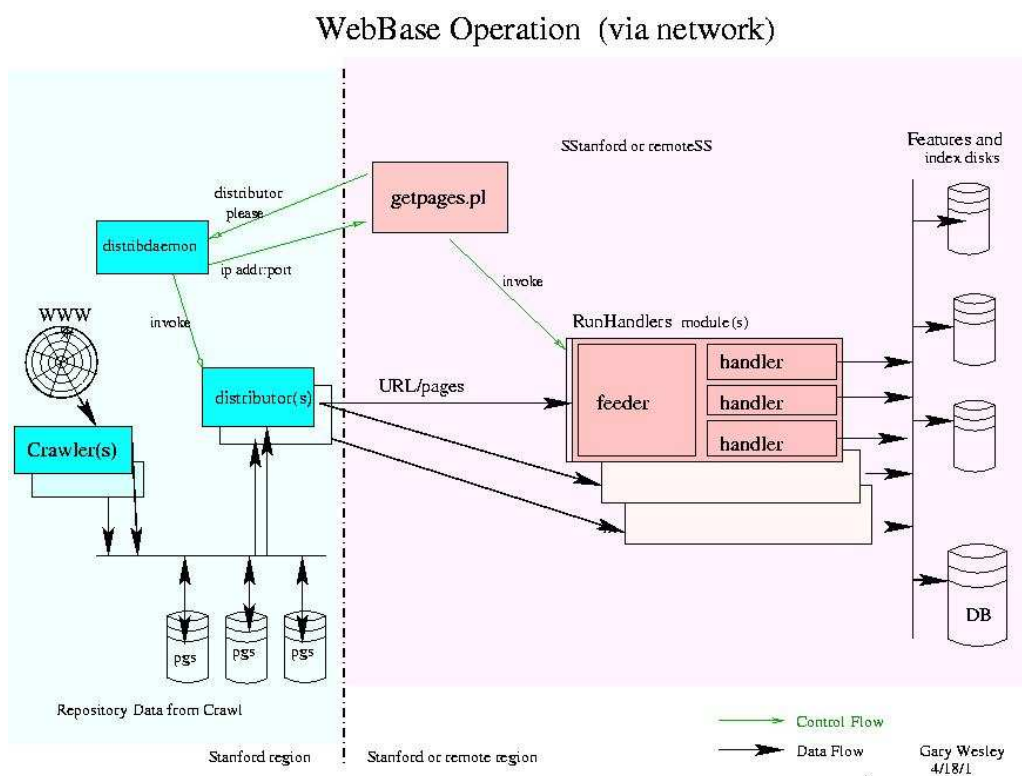


Figure 2.5: WebBase index factory diagram

distribution facility will multicast the WebBase content through multicast channels. Channels may vary in bandwidth and content. Subscribers specify the parameters of their channel, including the subset of pages that are of interest to them.

- **Query Engine:** Query-based access to the pages and the computed features (from the feature repository) is provided via the WebBase query engine. Unlike the traditional keyword-based queries supported by existing search engines, queries to the WebBase query engine can involve predicates on both the content and link structure of the Web pages.

## 2.3 Ranking algorithms

As is it described in [Bro02] there are three different types of queries according to their intent: *navigational* (to get a particular site), *informational* (to retrieve information assumed to be present on one or more web pages) and *transactional* (to perform some web-mediated activity). For all of them, as I said in the previous sections, the web is very large and diverse and many pages could be related to a given query. That is why a method to sort the entire pages subject to be interesting to a user query is needed. To solve this problem we need to introduce a new concept associated to every page: importance. For each web page, a value will be associated measuring its importance (for some algorithms we will define two scores instead of one: *authority* and *hub*. A definition of these scores is given in the section 2.3.3 because they were defined in that context). Built on top of the text of the web pages they hypertext (link structure and link text) provides an important way to retrieve information. That way, once the search engine retrieves the results that match the user query, it will order them according to their importance score presenting first the most important (supposed to be likely more interesting to the user).

In this section the most important and used algorithms to assign scores to web pages are described. However, first I will introduce the notation used from this point.

### 2.3.1 Notation

Let me use the notation used in [JW02a] for all the algorithms I will describe.

Let  $G = (V, E)$  denote the *web graph*, where  $V$  is the set of all web pages and  $E$  contains a directed edge  $\langle p, q \rangle$  iff page  $p$  links to page  $q$ . For a

page  $p$   $I(p)$  denotes the set of in-neighbours (pages pointing to  $p$ ) and  $O(p)$  the set of out-neighbours (pages pointed by  $p$ ). Individual in-neighbours are denoted as  $I_i(p)$  ( $1 \leq i \leq |I(p)|$ ), and individual out-neighbours are denoted analogously. For convenience, pages are numbered from 1 to  $n$ , and is possible to refer to a page  $p$  and its associated number  $i$  interchangeably. For a vector  $\mathbf{v}$ ,  $v(p)$  denotes *entry*  $p$ , the  $p$ -th component of  $\mathbf{v}$ . Vectors will be always typeset in boldface and scalars (e.g.,  $v(p)$ ) in normal font. All vectors are  $n$ -dimensional and have nonnegative entries. They should be thought of as distributions rather than arrows.

Let  $A$  be the matrix corresponding to the web graph  $G$ , where:

- $A_{ij} = \frac{1}{|O(j)|}$  for PageRank and HubRank
- $A_{ij} = 1$  for HITS, SALSA, Randomized HITS and SimRank

if page  $j$  links to page  $i$ , and  $A_{ij} = 0$  otherwise. The difference between the algorithms is that for PageRank we should normalize the matrix in order to make the algorithm converge. In the other algorithms, another normalization is done if necessary.

For almost every algorithm I will show some results as an example. The graph used to calculate the scores is shown in the figure 2.6

As we can see this graph contains two universities where each one point to each other and to one project. Each of these projects point to one researcher who belongs to the project. In addition we have a page with a list of the universities and a page with a list of the projects. Moreover, an external researcher points to the projects A and B and a company works (so links to) in the Project C.

### 2.3.2 PageRank

PageRank [PBMW98, BP98] is a method for computing a ranking for every web page based on the graph of the web. The idea behind PageRank is that pages with many backlinks are more important than pages with only a few backlinks. The concept is really simple, let us think of each link from page  $q$  to page  $p$  as a vote or recommendation. It means that highly linked pages are more “important” (voted or recommended).

However, this approach has some weaknesses. Imagine I create a new page and I want it to be really important. The only thing I have to do is to create many other pages pointing to that one and my notion of “importance” will get increased. Therefore, a small modification is necessary. If a page has only two backlinks but they are Netscape and Yahoo!, my page should have a high rank (even if only two pages link to that page). To solve this problem

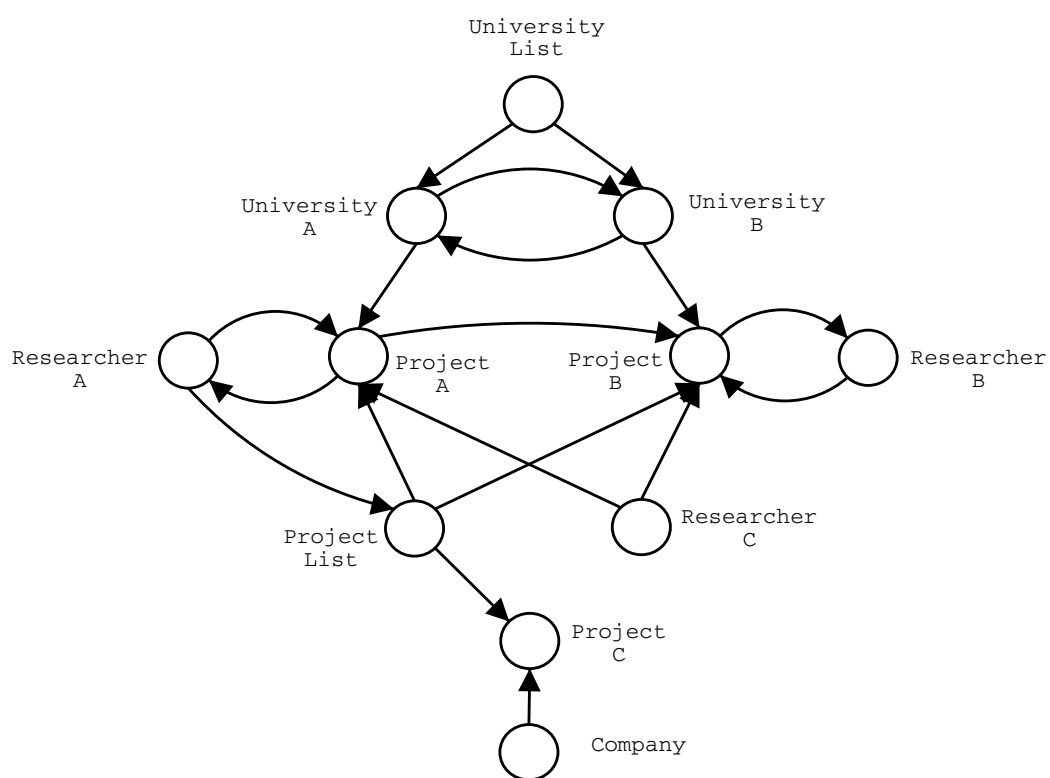


Figure 2.6: Small Subgraph

PageRank is refined with the following intuitive description: “a page has high rank if the sum of the ranks of its backlinks is high”.

Let  $p$  be a web page, the formula is the following:

$$PR(p) = (1 - c) \sum_{q \in O_p} \frac{PR(q)}{O_q} + cE(p) \quad (2.1)$$

In the calculation of PageRank a factor  $c$  is used for normalization. Note that  $c < 1$  because there are pages without outgoing links and their weight is lost.

Stated another way, the vector  $\mathbf{r}$  of pageranks is an eigenvector of the web structure matrix  $A$  with the dominant eigenvalue of  $A$ .

$$\mathbf{r}^{n+1} \leftarrow (c - 1) \cdot A \cdot \mathbf{r}^n + c \cdot \mathbf{e} \quad (2.2)$$

### Random Surfer Model

Let me explain the PageRank algorithm with what the authors called Random Surfer Model [PBMW98]. Imagine a user navigating randomly on the graph of the web. At each step, this surfer will choose randomly among the outgoing links of the current page with probability  $(c - 1)$  or he will jump (teleportation factor representing that the surfer gets bored) randomly to any other page of the graph (based on the vector  $E$  with probability  $c$ ). In fact, as it will be shown in next sections, this vector  $E$  can be used to generate customized page ranks.

This Random Surfer Model represents the standing probability distribution of a random walk on the graph of the web.

### GOOGLE

It is valuable to say that Google [Goo] not only uses PageRank to order its results. Other text-based heuristics are also used (as it is described in [Mar]). The hits (matches between the query and the index) are classified into *fancy hits* and *plain hits*.

Google associates the anchor text of a page with the page containing the link but associate this anchor text with the page it points to as well. This approach has two benefits:

- Many pages does not have the descriptive text in their anchor text. Imagine for example Google webpage which does not have the text “search engine” anywhere in its page or Toyota webpage where you cannot find the text “automobile manufacturer”

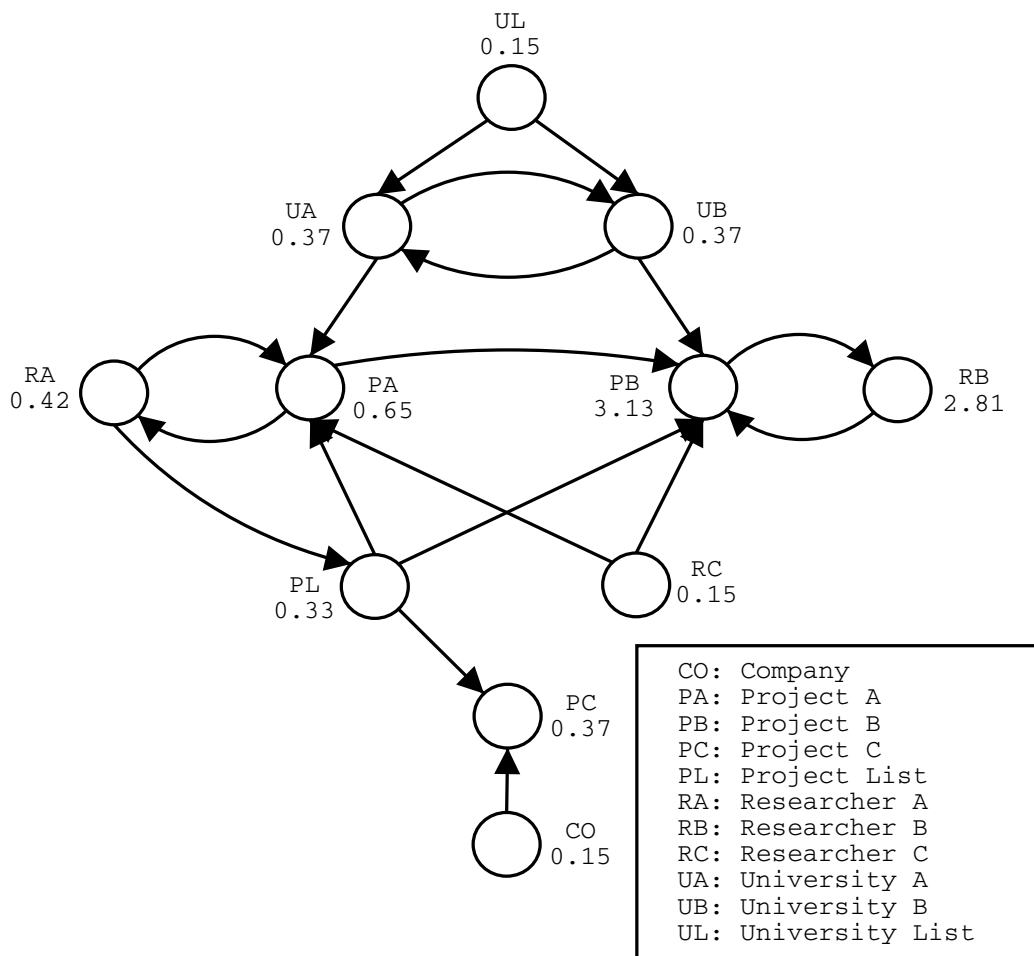


Figure 2.7: PageRank Example

- In the case of images, other non-text documents or even pages that have not been crawled yet it is not possible to associate any description extracted from them. That is why the pages pointing to them also describe them.

### Example

In the figure 2.7 are shown the scores for the nodes of the example and in table 2.1 is shown the ranking.

As we can see in the example the nodes *Project B* and *Researcher B* are the highest ranked. It is clear that they are the most important nodes in our graph (according to our definition of importance where the links of other nodes give it). We can also see that the nodes *University List*, *Company* and

Node	PageRank
University List	9
University A	5
University B	5
Researcher A	4
Project A	3
Project B	1
Researcher B	2
Project List	8
Researcher C	9
Project C	7
Company	9

Table 2.1: PageRank Ranking

*Researcher C* are the lowest ranked because they do not have in-going links.

### 2.3.3 HITS

Basically, in [Kle99] Jon M. Kleinberg define a relationship between two new entities in the web graph: *authorities* (pages that have “authoritative” information about topics) and *hubs* (pages that have links to many important pages of the same topic). He uses an algorithm called HITS (Hypertext Induced Topic Selection) to discover “authoritative” sources about topics.

Let me start with an example I tried in one of the most important search engines (you have the result in the figure 2.8). Imagine I am an Internet user and I want to buy a flight ticket. Of course I know that I can do it easily in Internet so I go to any search engine and I search with the next keywords: “airline company”. Not surprisingly I receive 2,350.000 results related to my query (probably this is the limit for the search engine in a specific time and there are even many more). These results are supposed to be the most important pages (defined as authorities) related to my query. The problem now is that if I want to get benefit of it, I have to go through all of them one by one and many of them will have nothing to do with what I wanted (e.g. news about airline companies or other kind of webs what in my case was the following ones: “Yahoo! Airline Industry News” or “AV Jobs.com - Your Aviation Job Search Homepage” and they were among the forth most important pages). The idea of this algorithm is to find those pages that join all the important pages related to one topic. Then, if we reach one of these pages we will have only pages related to what we were searching for.

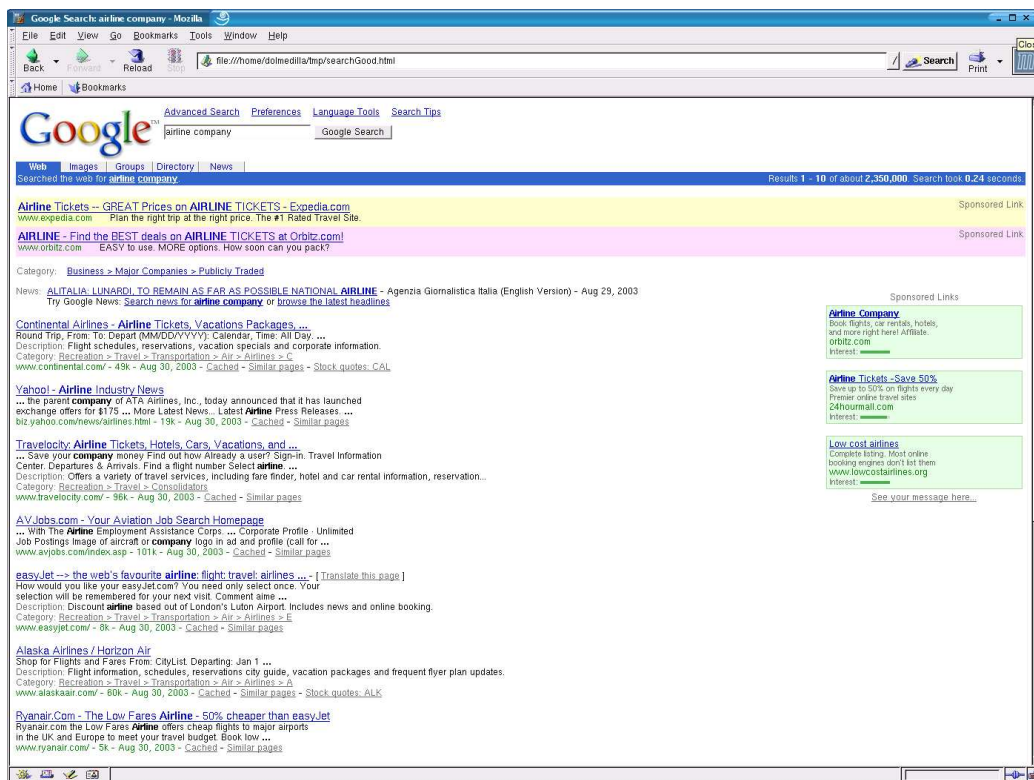


Figure 2.8: Search with keywords “airline company” in Google

In another classification of the type of queries presented in [Kle99] we have the following:

- **Specific queries:**  
e.g. is Suse a distribution of Linux?
- **Broad-topic queries:**  
e.g. find information about airline companies.
- **Page-page queries:**  
e.g. find pages similar to “www.elpais.es”.

We can see that the first type of queries should find a small set of pages related to our question (if there is any or if they have been crawled by the search engine crawler). On the other hand, with the second type of queries we expect to receive a big amount of pages related to our request (more than the number we can process) and not all of them will be related to our query. Imagine we use the term “Madrid” in our query. We will receive pages related to the capital of Spain, to the soccer team Real Madrid, etc. The same applies for example to the term “windows”. As we can see we have two different challenges: one is to retrieve only the real most important pages (what we will call authorities) and to group them into topics.

In addition, web search engines use text matching in order to decide which pages are relevant to a user query. However, many authorities in some topics do not have the term we search for (e.g. Yahoo!, Excite or Altavista do not have the term “search engine” in their pages) so it makes it even more difficult.

The algorithm can be split in the following steps:

- *Select a starting set of pages:* the algorithm will focus on a subgraph of the whole web. Then we need a small set of pages with pages related to the topic we are searching for. This basic set can be obtained selecting the most important  $k$  pages from the results of our query in a search engine.
- *Extend the starting set of pages:* as I already said, some of the authoritative sources might not be in the results returned by the search engine so we have to extend this set in order to include them.
- *Calculate the scores:* then we calculate two scores for each page: one for the authority measure and the other for the hub measure.

These steps are described in the following sections.

Subgraph( $\sigma, \xi, t, d$ )

$\sigma$ : a query string.

$\xi$ : a text-based search engine

$t, d$ : natural numbers.

Let  $R_\sigma$  denote the top  $t$  results of  $\xi$  on  $\sigma$ .

Set  $S_\sigma := R_\sigma$

For each page  $p \in R_\sigma$

    Let  $\Gamma^+(p)$  denote the set of all pages  $p$  points to.

    Let  $\Gamma^-(p)$  denote the set of all pages pointing to  $p$ .

    Add all pages in  $\Gamma^+(p)$  to  $S_\sigma$

    if  $|\Gamma^-(p)| \leq d$ , then

        Add all pages in  $\Gamma^-(p)$  to  $S_\sigma$

    Else

        Add an arbitrary set of  $d$  pages from  $\Gamma^-(p)$  to  $S_\sigma$

End

Return  $S_\sigma$

Figure 2.9: Algorithm to construct a focused subgraph of the WWW

### Constructing a Focused Subgraph of the WWW

In order to get a good set of pages we have to take into account the following: the set has to be small, rich in relevant pages and it should contain many of the strongest authorities.

The algorithm (extracted from [Kle99]) is shown in the figure 2.9. In this algorithm, we submit our query to a search engine. We select the  $t$  highest ranked pages (suppose to be probably most of the highest authorities). Then we extend this set with the pages pointed by these pages and with many of the pages that point into this set (to any of the pages of this set). The reason to do that is that if we have many of the most important authorities, it is highly probable that some of these pages will point to or will be pointed by other high authorities.

This algorithm is also known as the *Kleinberg Extension*.

## Computing Hubs and Authorities

Now we have a set with many relevant pages and most of the strong authorities on one domain we have to rank the pages of this set. The basic idea is called “*Mutually Reinforcement*” by Jon. M. Kleinberg: a good *hub* is a page that points to many good authorities and a good *authority* is a page that is pointed to by many good hubs.

For any page we associate two different values: a nonnegative *authority weight*  $a^p$  and a nonnegative *hub weight*  $h^p$ . As higher is the authority or hub weight is better authority or hub is the page. It is needed that in each step this values over the whole set of pages be normalized so their squares sum to 1. That is:  $\sum_{p \in S_\sigma} (a^p)^2 = 1$  and  $\sum_{p \in S_\sigma} (h^p)^2 = 1$ .

In order to update the scores in each iteration of the algorithm two operations are defined.

Operation  $\varrho$ :

$$a^p \leftarrow \sum_{q:(q,p) \in E} h^q \quad (2.3)$$

Operation  $\gamma$ :

$$h^p \leftarrow \sum_{q:(p,q) \in E} a^q \quad (2.4)$$

These two operations are applied till a convergence is reached. The algorithm is described in the figure 2.10.

Then is possible to apply the following procedure to filter the top  $c$  authorities and top  $c$  hubs as described in figure 2.11.

Stated another way, given  $A$  the matrix with the link structure of the subgraph, the vector of authorities is the *principal eigenvector* of  $A^T A$ , and the vector of hubs is the *principal eigenvector* of  $AA^T$ .

### Example

In the figure 2.12 are shown the authority and hub scores for each node of the example and in table 2.2 is shown the ranking.

In this case we have two different kinds of scores. We see how the best authorities are *Project A* and *Project B* and the best hubs are *Project List* and *Researcher C*. It seems to be clear that this approach shows results according to the definitions given above and to our intuition.

### 2.3.4 SALSA

The SALSA algorithm (Stochastic Approach for Link-Structure Analysis) [LM00] is an equivalent weighted in-degree analysis of the link-structure subgraphs,

Iterate( $G, k$ )

$G$ : a collection of  $n$  linked pages

$k$ : a natural number

Let  $\mathbf{z}$  denote the vector  $(1, \dots, 1) \in \mathbf{R}^n$

Set  $a_0 := \mathbf{z}$ .

Set  $h_0 := \mathbf{z}$ .

For  $i = 1, 2, \dots, k$

    Apply the  $\varrho$  operation to  $(\mathbf{a}_{i-1}, \mathbf{h}_{i-1})$ , obtaining new a-weights  $\mathbf{a}'_i$ .

    Apply the  $\gamma$  operation to  $(\mathbf{a}'_i, \mathbf{h}_{i-1})$ , obtaining new h-weights  $\mathbf{h}'_i$ .

    Normalize  $\mathbf{a}'_i$ , obtaining  $\mathbf{a}_i$ .

    Normalize  $\mathbf{h}'_i$ , obtaining  $\mathbf{h}_i$ .

End

Return  $(\mathbf{a}_k, \mathbf{h}_k)$ .

Figure 2.10: HITS algorithm

Filter( $G, k, c$ )

$G$ : a collection of  $n$  linked pages

$k, c$ : natural numbers

$(\mathbf{a}_k, \mathbf{h}_k) := \text{Iterate}(G, k)$ .

Report the pages with the  $c$  largest coordinates in  $\mathbf{a}_k$  as authorities.

Report the pages with the  $c$  largest coordinates in  $\mathbf{h}_k$  as hubs.

Figure 2.11: Filter algorithm

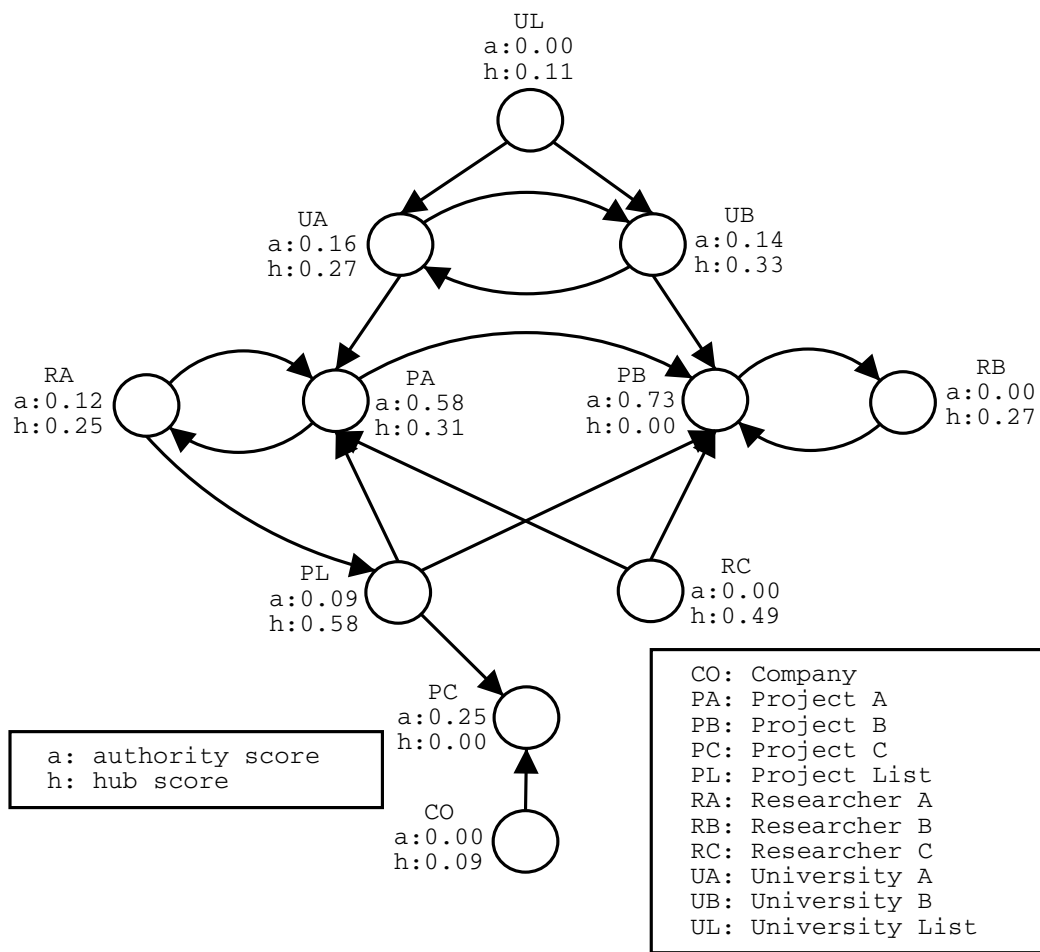


Figure 2.12: HITS example

Node	Authority	Hub
University List	9	8
University A	4	6
University B	5	3
Researcher A	6	7
Project A	2	4
Project B	1	10
Researcher B	8	5
Project List	7	1
Researcher C	9	2
Project C	3	11
Company	9	9

Table 2.2: HITS Ranking

making it more efficient than the Mutual Reinforcement approach of HITS.

In SALSA the subject that web sites pertain to a given topic is split into hubs and authorities is preserved. However, they replace the Kleinberg's Mutual Reinforcement approach by a new stochastic approach. In this new approach the coupling between hubs and authorities is less tight.

In SALSA they define an already known concept: *informative link*. An *informative link* is a link  $p \rightarrow q$  where a page  $p$  suggests (can also be seen as a recommendation) that the surfers visiting  $p$  follow the link and visit  $q$ . Normally this way it is supposed that  $p$  and  $q$  has some common topics of interest. The innovation of this definition is that we are supposing not only that this link is a recommendation but also that it shows that both pages share some common topics of interest.

I will start defining the meta-algorithm used to finally describe the SALSA algorithm and give an example.

### A Meta-Algorithm for Link Structure Analysis

A high-level algorithm to find hubs and authorities (also used in [Kle99]) is:

- Given a topic  $t$  construct a set of many hubs and authorities related to that topic ( $t$ -hubs and  $t$ -authorities) but it should not contain hubs or authorities related to other topics. Let  $n = |C|$ .
- Derive from  $C$  and the link structure two matrices of dimension  $n \times n$ . These matrices have the next property:

Let  $M$  be the matrix,  $M$  will have a unique real eigenvalue  $\lambda(M)$  of multiplicity 1, such that for any other eigenvalue  $\lambda'$  of  $M$ ,  $\lambda(M) > |\lambda'(M)|$ . Denote by  $v_{\lambda(M)}$  the (unique) unit eigenvector which corresponds to  $\lambda(M)$ . It will be positive vector and will be referred as the *principal eigenvector* of  $M$ .

- For some  $k < n$ , the  $k$  sites that correspond to the largest coordinates of  $v_{\lambda(A)}$  will form the *principal algebraic community of authorities* in  $C$ . The *principal algebraic community of hubs* in  $C$  is defined similarly.

## SALSA

The idea behind this algorithm is that a random walk will surf more likely (with high probability) the t-authorities of our web subgraph. In previous experiences with random walks (e.g. [PBMW98]), they analyse one Markov chain but in this case they analyse two different Markov chain: a chain of hubs and a chain of authorities. The random walk will surf in both chains traversing two links in a row, one link forward and one link backward (or vice versa). It will be explained in more detail below.

In this case we have a bipartite undirected graph  $\tilde{G} = (V_h, V_a, E)$  from our site collection  $C$  and the links structure associated to it:

- $V_h = \{s_h | s \in C \text{ and } O(s) > 0\}$  (the *hub-side* of  $\tilde{G}$ )
- $V_a = \{s_a | s \in C \text{ and } I(s) > 0\}$  (the *authority-side* of  $\tilde{G}$ )
- $E = \{(s_h, r_a) | s \rightarrow r \text{ in } C\}$

As the t-hubs and t-authorities should be highly visible in the subgraph  $\tilde{G}$ , it is expected that the t-authorities be amongst the most visited nodes by the random walk on  $V_a$  and the t-hubs be amongst the most visited nodes by the random walk on  $V_h$ .

Then the transition matrices associated to the Markov chains are as follows:

The *authority matrix*  $\tilde{A}$ :

$$\tilde{a}_{pq} = \sum_{\{k | (k_h, p_a), (k_h, q_a) \in \tilde{G}\}} \frac{1}{I(p_a)} \cdot \frac{1}{O(k_h)} \quad (2.5)$$

and the *hub matrix*  $\tilde{H}$ :

$$\tilde{h}_{pq} = \sum_{\{k | (p_h, k_a), (q_h, k_a) \in \tilde{G}\}} \frac{1}{O(p_h)} \cdot \frac{1}{I(k_a)} \quad (2.6)$$

Node	Authority	Hub
University List	9	2
University A	3	2
University B	3	2
Researcher A	6	2
Project A	2	2
Project B	1	8
Researcher B	6	8
Project List	6	1
Researcher C	9	2
Project C	3	11
Company	9	8

Table 2.3: SALSA Ranking

Another way to define the matrices  $\tilde{A}$  and  $\tilde{H}$  is described also in [LM00]. Let  $W$  be the adjacency matrix of the directed graph defined by  $C$  and its link structure. Denote by  $W_r$  the matrix which results by dividing each nonzero entry of  $W$  by the sum of the entries in its row (*normalization by rows*, so the sum of each row will be 1), and by  $W_c$  the matrix which results by dividing each nonzero element of  $W$  by the sum of the entries in its columns (*normalization by columns*, so the sum of each column will be 1). Then  $\tilde{H}$  consists of the non-zero rows and columns of  $W_r W_c^T$  and  $\tilde{A}$  consists of the non-zero rows and columns of  $W_c^T W_r$ .

Then we have to calculate the *principal eigenvector* of  $\tilde{A}$  and extract the  $k$ -sites having the highest entries to get the principal community of authorities. In the same way, we have to calculate the *principal eigenvector* of  $\tilde{H}$  and extract the  $k$ -sites having the highest entries to get the principal community of hubs.

### Example

In the figure 2.13 are shown the authority and hub scores for each node of the example and in table 2.3 is shown the ranking.

SALSA gives us similar results as HITS but the authors claim that it is more stable. As we can see the best authorities are the same than in HITS (*Project A* and *Project B*) and the best hub is still *Project List*. In this case the difference among hubs is not so big because we are taken into account only the out-going links what is a simpler and more stable approach but with less variation than HITS.

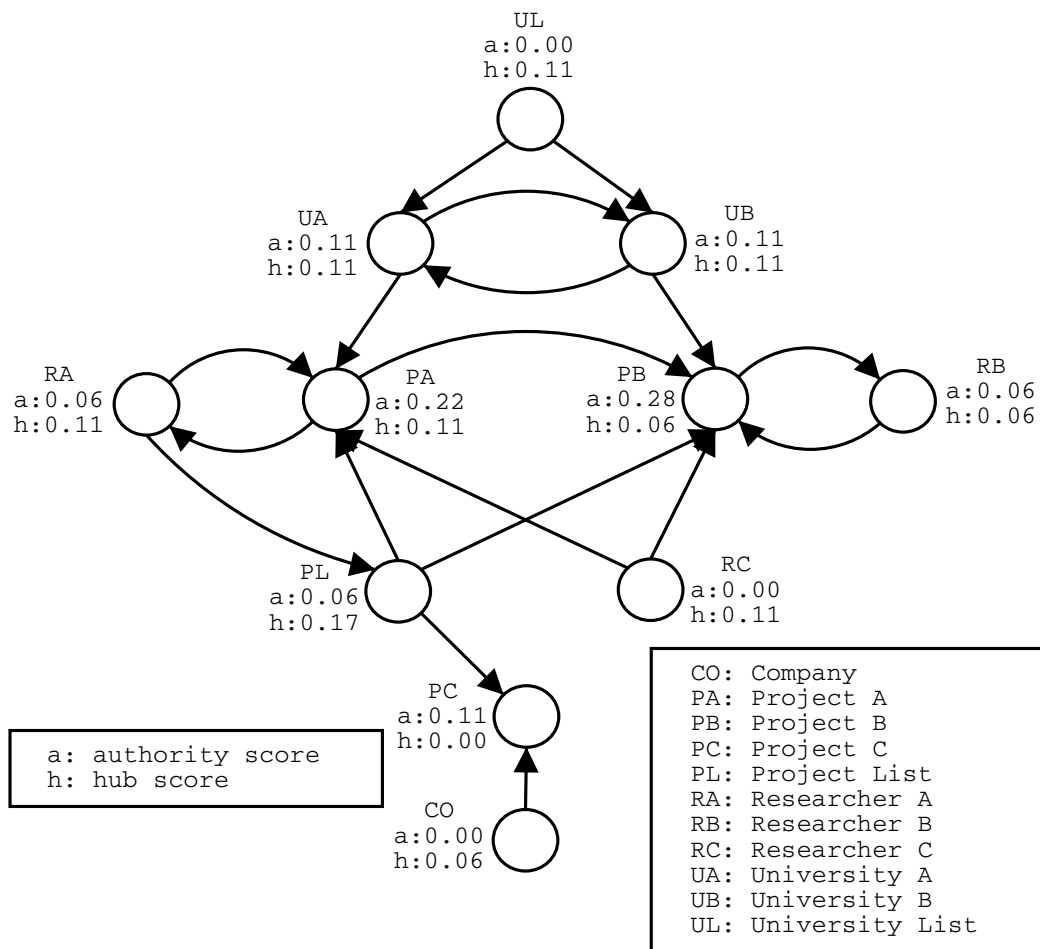


Figure 2.13: SALSA example

### 2.3.5 Randomized HITS

The World Wide Web is changing continuously. If we have a rank algorithm that orders the results we expect that this rank algorithm will return similar results with small perturbations into the source set. For example, if we send a query to a search engine and we receive that the second result returned is the webpage <http://www.abcd.com>, I would expect that in the next day, if I send the same query, this webpage would still be between the best results (and not in the 200th position). This is called *stability* in [NZJ01b, NZJ01a]

HITS [Kle99] and PageRank [PBMW98] we calculate the principal eigenvectors of the associated matrices based on the link structure of the web graph. These eigenvectors based methods are sensible to perturbations. In [NZJ01b, NZJ01a] they use some ideas from matrix perturbation theory and Markov chain theory to test the stability of these algorithms.

The results of their study show how HITS is highly unstable with specific changes of the web graph. Imagine we have two big communities of pages without links between them. The highest authorities are found in the first community by HITS so the principal eigenvector of the matrix associated point to that community. Imagine that now, we add a new link joining both communities. The new eigenvector change drastically because the higher authorities will come from both communities. PageRank shows to be more stable because of the dumping factor (“teleportation” factor) what they called “reset-to-uniform” in [NZJ01b, NZJ01a]. This dumping factor makes PageRank almost immune to the perturbations.

Based on these results they proposed a new algorithm called *Randomized HITS* where they make use the apparently immunity of PageRank adding a dumping factor to the HITS algorithm.

The idea is to have a random surfer that will walk forward and backward in our web graph. In HITS we had a surfer jumping forward and afterwards backward choosing between the outgoing and ingoing links respectively for each jump. In this case a teleportation factor is introduced so in each step we will add the possibly to jump to a page chosen uniformly at random with probability  $c$ . That way we get the following new formulas:

$$\mathbf{a}^{t+1} = (1 - c)A_{row}^T \mathbf{h}^t + c \vec{\mathbf{1}} \quad (2.7)$$

$$\mathbf{h}^{t+1} = (1 - c)A_{col} \mathbf{a}^{t+1} + c \vec{\mathbf{1}} \quad (2.8)$$

where  $A_{row}$  is  $A$  normalized by rows,  $A_{col}$  is  $A$  normalized by columns and  $\vec{\mathbf{1}}$  is the vector with all the values equal to 1.

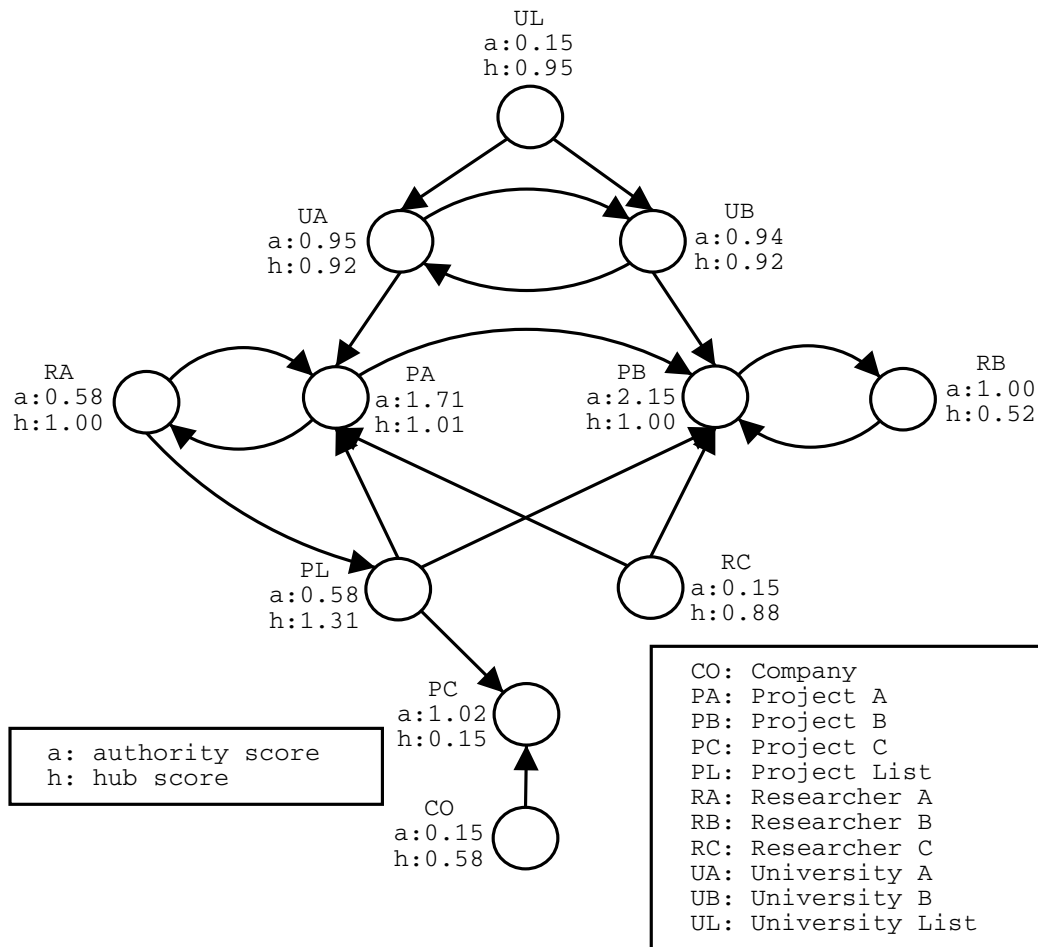


Figure 2.14: Randomized HITS example

### Example

In the figure 2.14 are shown the authority and hub scores for each node of the example and in table 2.4 is shown the ranking.

The results of Randomized HITS seem to be really good. The best authorities are still *Project A* and *Project B* and the best hub is the *Project List*. However, in this case the variation among the different node scores is much better and much more accurate.

### 2.3.6 Subspace HITS

Based on the previous study, another algorithm is proposed in [NZJ01b] (and it was already pointed in [Kle99]). This algorithm is called *Subspace HITS*

Node	Authority	Hub
University List	9	5
University A	5	7
University B	6	6
Researcher A	7	3
Project A	2	2
Project B	1	4
Researcher B	4	10
Project List	8	1
Researcher C	9	8
Project C	3	11
Company	9	9

Table 2.4: Randomized HITS Ranking

and it tries to combine several eigenvectors in order to increase the stability.

HITS is based in finding the principal eigenvector of the associated matrix of the web graph (this is find the eigenvector of the highest eigenvalue). In this new approach the idea is to find  $k$  eigenvectors and combine them (giving an appropriated weight to each one regarding on the eigenvalues so higher eigenvalues more importance). This approach is much more stable than the original in HITS.

The algorithm extracted from [NZJ01b] would be the following:

1. Find the first  $k$  eigenvectors  $x_1, \dots, x_k$  of  $S = A^T A$  (or  $AA^T$  for hub weights), and their corresponding eigenvalues  $\lambda_1, \dots, \lambda_k$ .
2. Let  $e_j$  be the  $j$ -th basis vector (whose  $j$ -th element is 1, and all other elements 0). Calculate the authority scores  $a_j = \sum_{i=1}^k f(\lambda_i)(e_j^T x_i)^2$ . (This is the square of the length of the projection of  $e_j$  onto the subspace spanned by  $x_1, \dots, x_k$ , where the projection in the  $x_i$  direction is *weighted* by  $f(\lambda_i)$  ).

This algorithm is a generalization where we can combine several eigenvector in order to have a single measure of *authoritativeness*. For example, if we choose  $f(\lambda_i) = 1$  when  $\lambda_i = \lambda_{max}$  and  $f(\lambda_i) = 0$  otherwise we get the HITS algorithm.

### 2.3.7 SimRank

If we look at the development of the search engines we can realize that at the beginning only text techniques were applied in order to satisfy user queries.

Now several techniques are used including some related to graph theory. The same applies to the problem of finding related documents. Many text techniques has been investigated and are being investigated but in [JW02b] a new technique base in a graph-theoretic model is presented. It is considered that “two objects are similar if they are related to similar objects”.

Let me put it into our context. In the web two pages are related if they have hyperlinks between them. Therefore, if we have two pages and they are pointed by similar pages then they are similar. The base case is that pages are similar to themselves.

Let define our graph  $G = (V, E)$ . As well as in our previous algorithms  $V$  represents the vertices (pages) and  $E$  represents the edges (links between pages). As I said before, as this is a recursive problem, we have to set the base case. In our case a page is maximally similar to itself so it will receive a score of 1.

The similarity between two pages  $p$  and  $q$  is defined as  $s(p, q) \in [0, 1]$ . Then we have  $s(p, q) = 1$  if  $p = q$  and the following formula otherwise:

$$s(p, q) = \frac{C}{|I(p)||I(q)|} \sum_{i=1}^{|I(p)|} \sum_{j=1}^{|I(q)|} s(I_i(p), I_j(q)) \quad (2.9)$$

where  $0 \leq C \leq 1$  (which represents the *decay factor*. The summation  $s(p, q) = 0$  if  $I(p) = 0$  or  $I(q) = 0$  ( $p$  or  $q$  do not have in-neighbours).

This score is calculated for each pair of pages. The similarity between two pages is the average similarity between in-neighbours of both pages. A convergence proof is detailed in [JW01]. It is ease to see from equation 2.9 that SimRank scores are symmetric so  $s(p, q) = s(q, p)$ .

### Bipartite SimRank

The algorithm can also be used in recommendation systems if we define two different kinds of objects in the graph (users and items for example). If two persons purchase similar items we can conclude that they two persons are similar. Moreover, if some items are purchased by similar people we can also conclude that these items are similar. We have a *mutually-reinforcement* relationship:

- People are *similar* if they purchase *similar* items. Therefore, for two users  $A$  and  $B$  where  $A \neq B$  we have:

$$s(A, B) = \frac{C_1}{|O(A)||O(B)|} \sum_{i=1}^{|O(A)|} \sum_{j=1}^{|O(B)|} s(O_i(A), O_j(B)) \quad (2.10)$$

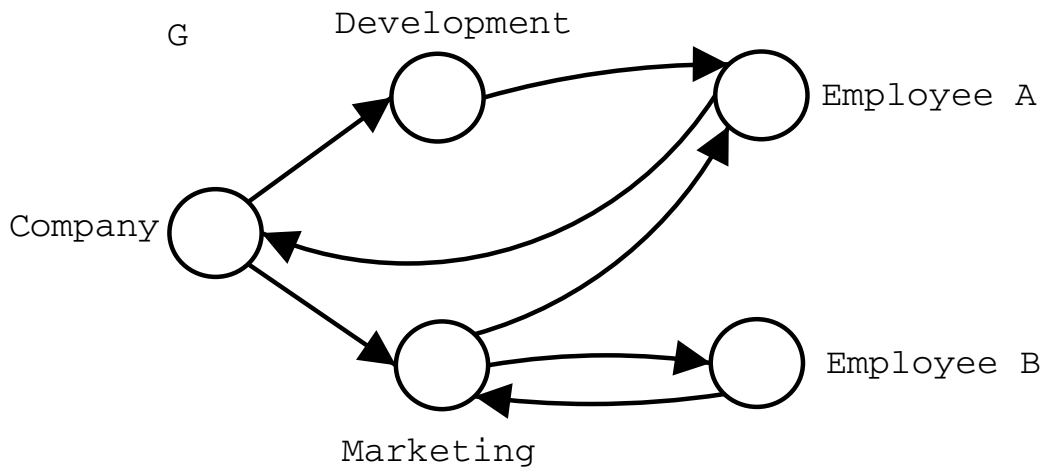


Figure 2.15: Small Web graph G

- Items are *similar* if they are purchased by *similar* people. Therefore, for two items  $c$  and  $d$  we have:

$$s(c, d) = \frac{C_2}{|I(c)||I(d)|} \sum_{i=1}^{|I(c)|} \sum_{j=1}^{|I(d)|} s(I_i(c), I_j(d)) \quad (2.11)$$

and  $s(A, B) = 1$  if  $A = B$  and  $s(c, d) = 1$  if  $c = d$ .

### Example

In the figure 2.15 there is drawn a small web graph  $G$  as an example. In the figure 2.16 the couples of nodes as well as their SimRank scores are depicted.

The table 2.5 shows the results obtained using  $C = 0.8$ .

As it was expected the rank of similarity between the departments is really high as well as the one between the two employees. In addition, the similarity between any employee and the company or any employee and any department is low.

### 2.3.8 Other algorithms

Here some other algorithms are just named and briefly described.

#### LSI

Latent Semantic Indexing ([DDL<sup>+</sup>90, PTRV98] exploits dependencies or "semantic similarity" between terms. This is done by simultaneously modelling

Page 1	Page 2	Score
Company	Company	1
Company	Development	0.0990227
Company	Marketing	0.281552
Company	Employee A	0.124039
Company	Employee B	0.126603
Development	Company	0.0990227
Development	Development	1
Development	Marketing	0.450542
Development	Employee A	0.152058
Development	Employee B	0.225105
Marketing	Company	0.281552
Marketing	Development	0.450542
Marketing	Marketing	1
Marketing	Employee A	0.158469
Marketing	Employee B	0.187458
Employee A	Company	0.124039
Employee A	Development	0.152058
Employee A	Marketing	0.158469
Employee A	Employee A	1
Employee A	Employee B	0.580167
Employee B	Company	0.126603
Employee B	Development	0.225105
Employee B	Marketing	0.187458
Employee B	Employee A	0.580167
Employee B	Employee B	1

Table 2.5: SimRank scores



all the interrelationships among terms and documents. It is assumed that there is some underlying or “latent” structure in the pattern of word usage across documents, and use statistical techniques to estimate this latent structure. A description of terms, documents and user queries based on the underlying, “latent semantic”, structure (rather than surface level word choice) is used for representing and retrieving information. One advantage of the LSI representation is that a query can be very similar to a document even when they share no words.

### **CLEVER Algorithm**

Run by IBM Almaden Research Center the CLEVER [CDG<sup>+</sup>98] algorithm is a variation of HITS [Kle99]. Two of the steps of HITS are modified: the set expansion and the authorities and hubs computation.

- *Set expansion*: all the pages with distance of two or less from the pages contained in the root set are added.
- *Calculation of authorities and hubs*: the summation of the iteration is now weighted one. The weighting is based on the number of query term matches ( $n(t)$ ) within a given distance. The new weighted score is  $w(p, q) = 1 + n(t)$ .

### **Hub-Averaging-Kleinberg Algorithm**

This is another variation of HITS [Kle99] proposed by Borodin where the idea is that a hub can be considered very important because it points to a very important authorities even if it points also to a very bad authorities. In this algorithm, the calculation of the hubs is modified in order to give a hub an average measure of all the scores from the authorities it is pointing to. That way a hub becomes a good hub only if it points only to good authorities.

### **Threshold-Kleinberg Algorithm**

This is another variation of HITS [Kle99] also proposed by Borodin. In HITS if we have a hub pointing to many authorities, it will become a good hub (even if all of the authorities it is pointing to are really bad). The same applies to an authority pointed by many bad hubs. This new algorithm adds a threshold so the score of a hub or an authority will only be taken into account if it is higher than a lower bound or threshold. Here three different possibilities are applied:

- **Hub-Threshold algorithm:** calculating the authority score of a page it only includes hubs whose hub weight score is at least equal to the average hub weight score of all pages pointing to the current page.
- **Authority-Threshold algorithm:** calculating the hub score of a page it only includes authorities whose authority weight score is at least equal to the average authority weight score of all pages pointing to the current page.
- **Full-Threshold algorithm:** this is the implementation of the both previous algorithms simultaneously.

### PHITS Algorithm

This is a statistical algorithm for the computation. A probabilistic model in which a latent “factor” or “topic”  $z$  causes a citation  $c$  of a document  $d$  is presented. They ([CC00]) further postulate that conditional distributions exist for a citation  $c$  given a factor  $z$ ,  $P(c|z)$ , and for factor  $z$  given a document  $d$ ,  $P(z|d)$ . They then produce a *likelihood function* in terms of these conditional distributions.

### WebQuery

WebQuery is a link-based analysis implemented in a similar manner to HITS but lacking the notion of hubs and authorities. The WebQuery ([CK97]) algorithm has two main components, a pre-processor phase and a run-time phase.

The results obtained by this algorithm show some similarity to the HITS results in that some of the most highly ranked pages did not occur in the initial hit set. It is also acknowledged that the approach taken can cause topic drift if the hit set contains a non-relevant node with high connectivity, such as the Yahoo! homepage.

## 2.4 Personalization

Once we reach this point of the document, the reader should have a basic knowledge of the most used ranking algorithms at the moment. However, these algorithms are far away to be the best possible ones. One of the biggest weaknesses of these algorithms is that they provide a good rank score, but they do not distinguish among the different users that are using them. It is needed a method to rank the results depending on the user interests.

### 2.4.1 Personalized PageRank

In [PBMW98], the authors already gave a notion of this personalized ranking. They called it *Personalized PageRank*. In the equation 2.1 there are two different parts. In the second part of the equation we can see that there is a vector called  $\mathbf{E}$ . This vector is the source vector for the ranks. It is used to represent the random surfer jumping to a random page of the web graph with higher or lower probability. However, it is also a good method to adjust the ranks scores. Normally, PageRank is calculated with this vector initialised a uniform vector so all the pages of the web graph will have the same importance and will contribute in the same way to the ranks.

The authors proposed an example where they started to calculate PageRank with a vector  $\mathbf{E}$  with all the values 0 except the one of the user's home page (which received the value 1). With this approach, the user's home page will have more contribution than others pages, the pages pointed by the user's home page will contribute more than the rest and so on. It is supposed that in your home page you give enough information about your contextual information based on the links you have.

Another possibility they pointed was to use the bookmarks of a user as a vector  $\mathbf{E}$ . It is supposed that the bookmarks measure more or less the interests of a user so they can be used as a source of ranks.

In conclusion, the vector  $E$  represents a good way to personalize the final ranks giving more importance to pages which are known to be interesting for the user. This method allows to have personalized rankings but has two big problems: it needs to have a different  $\mathbf{E}$  and vector of rankings for each user (this is not possible due to potential amount of users and the limited storage resources) and you have to calculate this rankings for each one of them (this is also not possible as the calculation is high time consuming and the time is also a limited resource).

### 2.4.2 Topic-sensitive PageRank

PageRank computes the link-based ranks for each page offline and it does not depend of the query submit by the user. Then the scores are merged with other scores from the matching between the pages and the query (query specific scores).

In [Hav02] a new approach is proposed. They choose several topics and compute personalized ranks for each topic and each page (so each page will have one rank for each topic associated). Then, at query time the search engine can choose the rank whose topic fit the best to the query. This approach solves the case where PageRank ranks high some pages that have

<b>Topics</b>
Arts
Business
Computers
Games
Health
Home
Kids & Teens
News
Recreation
Reference
Regional
Science
Shopping
Society
Sports
World

Table 2.6: Topics used in Topic-Sensitive PageRank

no authority for some queries.

First of all, they choose 16 topics from the Open Directory Project [Ope] (the categories are shown in table 2.6). Then they compute 16 biased PageRank vector using the urls present below each of the 16 top-level categories of the ODP as a personalization vector. It means that given  $T_i$  that represents the set of links below the top-level of the category  $c_i$ , the vector  $\mathbf{E}$  is constructed as follows:  $E_{ij} = \frac{1}{|T_i|}$  if  $j \in T_i$  or  $E_{ij} = 0$  if  $j \notin T_i$ . The PageRank vector associated to each  $c_i$  is denoted  $\vec{PR}(\alpha, \vec{e}_i)$ . The value of  $\alpha$  represents how much are we taking into account the biased vector. For example, with  $\alpha = 1$  the urls in  $T_i$  receive the score  $\frac{1}{|T_i|}$  and the rest 0. As far as we decrease the value of  $\alpha$  the urls in  $T_i$  become less relevant to the final scores. Heuristically they decided to use  $\alpha = 0.25$ .

They also compute 16 class *term-vectors*  $\vec{\mathbf{D}}_i$  consisting on the terms in the documents below each of the 16 top-level categories. Then  $D_{it}$  is how many times the term  $t$  occurs in the documents listed below the category  $c_i$ .

These first two steps are computed off-line while the next will be computed at query time.

Given a query  $q$   $q_i$  represent the term  $i$ th of the query. Then, the class probabilities for each of the 16 top-level ODP classes (conditioned to  $q$ ) are

calculated. For each  $c_i$  we have:

$$P(c_i|q) = \frac{P(c_i) \cdot P(q|c_i)}{P(q)} \alpha P(c_i) \cdot \prod_j P(q_j|c_i) \quad (2.12)$$

where  $P(q_j|c_i)$  can be easily computed from the class term vector  $\vec{\mathbf{D}}_i$  and  $P(c_i)$  is initialised uniformly (it could be also personalized but the authors did not do it).

In addition they define the rank  $r$  of each document  $d$  by the rank vector  $\vec{\mathbf{PR}}(\alpha, \vec{\mathbf{e}}_i)$  as  $r_{id}$ .

In response to the query they retrieve the documents containing the query terms of  $q$ . Then, they compute the score  $s_{qd}$  of each document (called *query-sensitive importance score*) as follows:

$$s_{qd} = \sum_i P(c_i|q) \cdot r_{id} \quad (2.13)$$

All the urls returned as results are ranked according to the scores  $s_{qd}$ .

### 2.4.3 Scaling Personalized Web Search

As I said then ranking algorithms presented in the section 2.3 calculate the scores without taking into account the query or the user that submit the query. These scores could be biased according to a user-specified set of initially interesting pages (e.g. his bookmarks), redefining importance according to user preference. However, calculate all the possible personalized views and store them is not possible due to our constraint of limited resources.

In [JW02a] a new technique is presented where the use of *partial vectors* is described. Partial vectors are shared across multiple personalized views and their computation and storage costs scale well with the numbers of views.

Let me redefine the vector  $\mathbf{E}$  I was describing in previous sections as a *Personalized PageRank Vector (PPV)* because this is the term given by the author of [JW02a]. He defines this vector as *a personalized view of the importance of pages on the web*. It means that given a set of pages  $P$  interested to a user, we can construct the associated *PPV*. Then, as we saw in the previous section, we could use it to bias the scores calculation and get personalized ranks. However, ranks calculation is high time consuming because it requires multiple scans of the whole web graph so it is not possible to calculate them at query time and if we want to store the rank vector for each user it is only possible for a small set of users because of limited storage capacity.

Let define the *magnitude* of a vector  $\mathbf{v}$  as  $|v| = \sum_{i=1}^n v(i)$  and  $0 \leq |v| \leq 1$  where  $v(i)$  represents the entry  $i$ th in the vector  $\mathbf{v}$ . Our vector  $\mathbf{e}$  defines the preference set of pages  $P$  where each  $e(p)$  is the amount of preference for page  $p$ . As it was described in section 2.3.2 we have the next equation to calculate the scores:

$$\mathbf{v} = (\mathbf{1} - c)\mathbf{A}\mathbf{v} + c\mathbf{e} \quad (2.14)$$

where  $0 \leq c \leq 1$ . The vector  $\mathbf{v}$  is the *Personalized PageRank Vector* for the preference vector  $\mathbf{e}$  (the global PageRank is a specific case where the uniform distribution vector is used as a preference vector).

### Basis Vector

Let  $\mathbf{x}_1, \dots, \mathbf{x}_n$  be the unit vectors in each dimension (it means that the vector has a value 1 in the  $i$ th entry and 0 the rest). Let  $\mathbf{r}_i$  be the *PPV* corresponding to  $x_i$ . Then each  $r_i$  (called *basis vector*) represent the teleportation factor where a surfer teleport to page  $i$  with probability  $c$ . Each entry  $j$  of  $\mathbf{r}_i$  represents the  $j$ 's importance in  $i$ 's view. Then a PPV  $\mathbf{v}$  can be written as a linear combination of the basis vector  $\mathbf{r}_i$  as the following:

$$\mathbf{v} = \sum_{i=1}^n \alpha_i \mathbf{r}_i \quad (2.15)$$

The preference set of pages are restricted to subsets of a set of hub pages  $H$  (it is expected that they will have high PageRank). Depending of how big is this set bigger will be the degree of personalization. Then we can compute and store a *basis hub vector* (or hereafter *hub vector*) for each  $p \in H$  and any PPV corresponding to a preference set  $P$  with  $k$  nonzero entries can be computed adding up the  $k$  corresponding hub vectors  $r_p$  with the appropriate weights  $\alpha_p$ . However the computation of these hub vectors is computationally expensive and this cost is increased linearly with the number of hub vectors ( $|H|$ ).

### Decomposition of Basis Vector

As it was said in the last section, compute and store all hub vectors is not possible. However, they can be decomposed into *partial vectors* and the *hubs skeleton* in order to compute a large number of them. With these components the hub vectors can be calculated quickly at query time (if we have enough resources they can be constructed offline and then query time would be constructed time).

Basically a partial vector is computed for each hub page  $p$  which encodes the part of the hub vector  $\mathbf{r}_p$  unique to  $p$ . The complement is the hubs skeleton which captures the interrelationships among hub vectors.

**Inverse P-distance** The *inverse P-distance* from a pages  $p$  to a page  $q$  is defined as

$$r'_p(q) = \sum_{t:p \rightsquigarrow q} P[t]c(1-c)^{l(t)} \quad (2.16)$$

where the summation is taken over all *tours*  $t$  (which may contain cycles) from  $p$  to  $q$  (it is possible to pass through  $p$  or  $q$  multiple times). The length  $l(t)$  of a tour  $t$  is the number of edges in  $t$ . The term  $P[t]$  is the “probability of travelling  $t$ ”. It is defined as 1 if  $l(t) = 0$  or  $\prod_{i=1}^{k-1} \frac{1}{|O(w_i)|}$  otherwise. As it is demonstrated in [JW02a]  $r'_p(q) = r_p(q)$  for all  $p, q \in V$  so hereafter  $r_p(q)$  will be used to denote the inverse P-distance and the personalized PageRank score.

See the following formula:

$$r_p^H(q) = \sum_{t:p \rightsquigarrow H \rightsquigarrow q} P[t]c(1-c)^{l(t)} \quad (2.17)$$

It restricts the equation 2.16 considering only the tours which pass through some page  $h \in H$  somewhere other than the endpoints.

**Partial Vectors** If we choose well our set  $H$  (e.g. hubs with high PageRank) it is true that  $r_p(q) - r_p^H(q) = 0$  for many pages  $p$  and  $q$ . Therefore we can take advantage of this property and define

$$\mathbf{r}_p = (\mathbf{r}_p - \mathbf{r}_p^H) + \mathbf{r}_p^H \quad (2.18)$$

Then it is possible to precompute the *partial vector*  $(\mathbf{r}_p - \mathbf{r}_p^H)$  instead of the full hub vector  $\mathbf{r}_p$  and add  $\mathbf{r}_p^H$  at query time to compute the full hub vector. However, compute and store  $\mathbf{r}_p^H$  could be as expensive as  $\mathbf{r}_p$  itself.

**Hubs Skeleton** Let define the next theorem

**Theorem 2.4.1** For any  $p \in V, H \subseteq V$ ,

$$\mathbf{r}_p^H = \frac{1}{c} \sum_{h \in H} (\mathbf{r}_p(h) - c\mathbf{x}_p(h))(\mathbf{r}_h - \mathbf{r}_h^H - c\mathbf{x}_h) \quad (2.19)$$

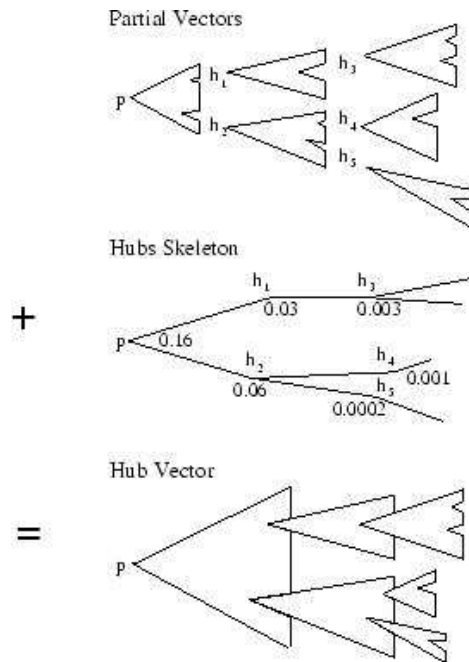


Figure 2.17: Construction of hub vectors

It means that the distance from page  $p$  to any page  $q \in V$  through  $H$  is the distance  $r_p(h)$  from  $p$  to each  $h \in H$  times the distance  $r_h(q)$  from  $h$  to  $q$ , correcting for the paths among hubs by  $r_h^H(q)$ .

Then, the quantity  $(\mathbf{r}_h - \mathbf{r}_h^H)$  of the equation 2.19 is the partial vectors described in the previous section. Therefore we have

$$\mathbf{r}_p = (\mathbf{r}_p - \mathbf{r}_p^H) + \frac{1}{c} \sum_{h \in H} (\mathbf{r}_p(h) - c\mathbf{x}_p(h))[(\mathbf{r}_h - \mathbf{r}_h^H) - c\mathbf{x}_h] \quad (2.20)$$

The set  $r_p(H)$  has size at most  $|H|$  what is much smaller than the full hub vector  $\mathbf{r}_p$  which can have up to  $n$  nonzero entries.

The set  $S = \{r_p(H) | p \in H\}$  forms the *hubs skeleton*, giving the interrelationships among partial vectors.

In the figure 2.17 an intuitive view of the construction of hub vectors from partial vectors and the hubs skeleton is given.

### Algorithms for Computing Basis Vectors

Computing these *partial quantities* naively using a fixed-point iteration for each  $p$  would scale poorly with the number of hub pages. Here there are

three scalable algorithms to compute these quantities efficiently by using dynamic programming to leverage the interrelationships among them. I will only present them so for more information I point the reader to [JW02a].

The three algorithms are iterative fixed-point computations and maintain a set of *intermediate results*  $(\mathbf{D}_k[*], \mathbf{E}_k[*])$ . For each  $p$ ,  $\mathbf{D}_k[\mathbf{p}]$  is a lower-approximation of  $\mathbf{r}_p$  on iteration  $k$ . Simultaneously the error components  $\mathbf{E}_k[\mathbf{p}]$  is calculated. The following invariant is maintained: for all  $k \geq 0$  and all  $p \in V$ ,

$$\mathbf{D}_k[\mathbf{p}] + \sum_{\mathbf{q} \in V} \mathbf{E}_k[\mathbf{p}](\mathbf{q}) \mathbf{r}_q = \mathbf{r}_p \quad (2.21)$$

so  $\mathbf{D}_k[\mathbf{p}]$  is a lower-approximation of  $\mathbf{r}_p$  with error  $|E_k[p]| = |\sum_{q \in V} E_k[p](q) \mathbf{r}_q|$ . At the beginning  $D_0[p] = 0$  and  $E_0[p] = \mathbf{x}_p$ .

**Basic Dynamic Programming Algorithm** On iteration  $k$ , we derive  $(D_{k+1}[p], E_{k+1}[p])$  from  $(D_k[p], E_k[p])$  using the equations

$$\mathbf{D}_{k+1}[\mathbf{p}] = \frac{1 - c}{|\mathbf{O}(\mathbf{a})|} \sum_{i=1}^{|\mathbf{O}(\mathbf{p})|} \mathbf{D}_k[\mathbf{O}_i(\mathbf{p})] + c \mathbf{x}_p \quad (2.22)$$

$$\mathbf{E}_{k+1}[\mathbf{p}] = \frac{1 - c}{|\mathbf{O}(\mathbf{a})|} \sum_{i=1}^{|\mathbf{O}(\mathbf{p})|} \mathbf{E}_k[\mathbf{O}_i(\mathbf{p})] \quad (2.23)$$

**Selective Expansion Algorithm**  $(\mathbf{D}_{k+1}[\mathbf{p}], \mathbf{E}_{k+1}[\mathbf{p}])$  are derived by “distributing” the error at each page  $q$  to its out-neighbours. Results on iteration- $k$  are computed using the equations

$$\mathbf{D}_{k+1}[\mathbf{p}] = \mathbf{D}_k[\mathbf{p}] + \sum_{\mathbf{q} \in \mathbf{Q}_k(\mathbf{p})} c \mathbf{E}_k[\mathbf{p}](\mathbf{q}) \mathbf{x}_q \quad (2.24)$$

$$\mathbf{E}_{k+1}[\mathbf{p}] = \mathbf{E}_k[\mathbf{p}] - \sum_{\mathbf{q} \in \mathbf{Q}_k(\mathbf{p})} \mathbf{E}_k[\mathbf{p}](\mathbf{q}) \mathbf{x}_q + \sum_{\mathbf{q} \in \mathbf{Q}_k(\mathbf{p})} \frac{1 - c}{|\mathbf{O}(\mathbf{q})|} \sum_{i=1}^{|\mathbf{O}(\mathbf{q})|} \mathbf{E}_k[\mathbf{p}](\mathbf{q}) \mathbf{x}_{\mathbf{O}_i(\mathbf{q})} \quad (2.25)$$

for a subset  $Q_k(p) \subseteq V$ .

**Repeated Squaring Algorithm** Similar to the selective expansion algorithm but the computations are essentially iteration- $2k$  results using the equations

$$\mathbf{D}_{2k}[\mathbf{p}] = \mathbf{D}_k[\mathbf{p}] + \sum_{\mathbf{q} \in \mathbf{Q}_k(\mathbf{p})} \mathbf{E}_k[\mathbf{p}](\mathbf{q}) \mathbf{D}_k[\mathbf{q}] \quad (2.26)$$

$$\mathbf{E}_{2k}[\mathbf{p}] = \mathbf{E}_k[\mathbf{p}] - \sum_{\mathbf{q} \in \mathbf{Q}_k(\mathbf{p})} \mathbf{E}_k[\mathbf{p}](\mathbf{q}) \mathbf{x}_{\mathbf{q}} + \sum_{\mathbf{q} \in \mathbf{Q}_k(\mathbf{p})} \mathbf{E}_k[\mathbf{p}](\mathbf{q}) \mathbf{E}_k[\mathbf{q}] \quad (2.27)$$

### Computation

Finally, let me described the process to compute the *partial quantities*:

1. Compute partial vectors  $(\mathbf{r}_{\mathbf{p}} - \mathbf{r}_{\mathbf{p}}^{\mathbf{H}})$ ,  $p \in H$ : a simple specialization of the selective expansion algorithm is used. We take  $Q_0(p) = V$   $Q_k(p) = V - H$  for  $k > 0$ , for all  $p \in V$  (we never “expand” hub pages after the first step, so tours passing through a hub page  $H$  are never considered).
2. Compute the hubs Skeleton  $S = \{r_p(H) | p \in H\}$ : a specialization of the repeated squaring algorithm is used. We apply the algorithm to the results of the previous step  $(\mathbf{D}_k[\mathbf{p}], \mathbf{E}_k[\mathbf{p}])$  using  $Q_k(p) = H$  for all successive iterations.
3. Construction of PPV's: Let  $\mathbf{u} = \alpha_1 \mathbf{p}_1 + \dots + \alpha_z \mathbf{p}_z$  be a preference vector where  $p_i \in H$  for  $1 \leq i \leq z$ . Let  $Q \subseteq H$  and let

$$r_{\mathbf{u}}(h) = \sum_{i=1}^z \alpha_i (r_{p_i}(h) - cx_{p_i}(h)) \quad (2.28)$$

which can be computed from the hubs skeleton. Then the PPV  $\mathbf{v}$  for  $\mathbf{u}$  can be constructed as

$$\mathbf{v} = \sum_{i=1}^z \alpha_i (\mathbf{r}_{\mathbf{p}_i} - \mathbf{r}_{\mathbf{p}_i}^{\mathbf{H}}) + \frac{1}{\mathbf{c}} \sum_{\substack{\mathbf{h} \in \mathbf{Q} \\ \mathbf{r}_{\mathbf{u}}(\mathbf{h}) > 0}} \mathbf{r}_{\mathbf{u}}(\mathbf{h}) [(\mathbf{r}_{\mathbf{h}} - \mathbf{r}_{\mathbf{h}}^{\mathbf{H}}) - \mathbf{c} \mathbf{x}_{\mathbf{h}}] \quad (2.29)$$

Both the terms  $(\mathbf{r}_{\mathbf{p}_i} - \mathbf{r}_{\mathbf{p}_i}^{\mathbf{H}})$  and  $(\mathbf{r}_{\mathbf{h}} - \mathbf{r}_{\mathbf{h}}^{\mathbf{H}})$  are partial vectors, which we assume have been precomputed. If  $Q = H$  then equation 2.29 represents a full construction of  $\mathbf{v}$ .

## Chapter 3

### Our work

In this section I will explain which is our contribution to this field. First of all, let me summarize some of the biggest problems existing with the current approaches and which do we think would be some possible solutions to that problems:

- Rank quality: current rankings have shown a really high quality and the proof is that most of them have been used (or they are still being used) successfully. However, some improvements can be done on them.
- Authority vs. hub quality: which are the most important measure? PageRank focus on the authority measure and it is one of the reasons of the success of Google [Goo]. However, we think that hubs are also really important so we try to combine both approaches.
- Personalization: the work presented in previous section about personalized ranking has several problems like limited resources (computation time or storage capacity), level of personalization (only several topics are covered) or limited user personalization (the user has to choose among a set of pages already given). Here we give more power to the user letting him to provide his preference set of pages ( $P$ ).

Our approach to solve all these problems at once is to improve all of the steps involved in the ranking itself. Starting from the crawl till the point where we give the scores for each web page. Roughly the idea is to get from the users a set of starting pages which they consider important or interesting enough and use it (after several modifications taking into account the whole link structure as well specific heuristics) as a personalization set of pages with the [JW02a] algorithm.

In order to implement our ideas we have modified an existing crawler (the modifications to the Webbase crawler are described in section 3.1). In addition, the new algorithms we introduce are based of the existing PageRank and Kleinberg Extension algorithms. Our modifications over them seem to improve these algorithms considerably. However, our main contribution is the whole process. As I said we have tried to improve and refine each of the steps involved in the process so not only each algorithm is important but also the information they exchange (input and output of each step).

### 3.1 Webbase

The first step needed in the process is to extract the link structure of the web (the web graph). This is not a simple task. Instead of develop a new whole crawler (what was out of the scope of our work) we decided to work with an already implemented crawler. We chose to use the WebBase crawler [ACGM<sup>+</sup>00, HRGMP00]. These were some of the main issues that made us decide for it:

- It allows several instances to be executed in parallel.
- It is a focused crawler so we provide the sites we want to crawl and it will crawl only pages from this domain.
- It crawls only text pages (images and other kind of documents are not crawled)
- The crawler check the *robots.txt* policy for crawlers. This policy is a method for sites to restrict access to crawlers.
- It is developed in C++ what assure maximum performance.
- The quality of the previous release of this crawler (it was used as a basis of the Google crawler)
- And the most important one: its condition of open source so we could look into the code and change it.

After choosing it, we had to go into the code and learn how it works. The only thing we need from the crawler was the link structure of the web graph. It seems to be pretty easy to do it but it was not. In addition, we wanted to improve even more the performance of the crawler removing some issues we did not need for our purposes. Here there are the improvements we developed:

- Link structure generation: of course we included the necessary code in one of the modules of the crawler in order to get the link structure of the web pages crawled. Basically, we constructed two new structures to manage this information: one for the list of all the urls seen so far (in this set would be not only the urls into the same domain but also all the urls pointing to other domains) and another one for the list of all the urls crawled so far (so we have not only the url but also the links provided by the url). For this purpose we use hash maps in order to increase the performance. Due to the low storage needs for our approach we maintain these structures in memory in order to have faster access to all of them. At the end (and periodically as a backup) these structures are backed up into two files: one with the urls seen and the other with the urls crawled.
- Start over an already crawled structure: as the crawl is a long process and we also wanted to try our algorithms with different sizes of the web, we constructed our set of crawled pages incrementally. For that reason, we implemented a new feature into the crawler: the possibility to start the crawler with a link structure as a start source. In this approach, both files with urls seen and urls crawled are provided to the crawler and stored in memory before the crawler starts. In addition, we developed a new application to merge different link structures. Basically two link structures are provided and both are merged into a new one that is the union of the two original ones.
- Remove the process and storage tasks: of course a crawler not only surf over the web retrieving the urls and links. It also retrieves the pages, it process the page and store it into the file system. For our purposes these tasks were not valuable (we did not need the pages crawled, only the link structure of them) so we decided to remove them. Removing them we got a high improvement in the performance (basically due to the high cost of any I/O operation to the file system).

With all these developments we got the input necessary for the next steps: the link structure of the web.

## 3.2 Ranking algorithms

Before to create or improve something first you need to know what already exists. Following this philosophy we developed the most important ranking algorithms. Our goal was to know perfectly how they work as well as to have

measures to compare new algorithms (it means that we can directly compare any new algorithm against the already known ones).

In this case the challenge was not only to implement some advanced algorithms which make use of complex spectral techniques from web graph theory to calculate scores (like eigenvector calculations). In all of them we had to deal with a huge amount of data (the whole link structure of the crawled web).

These are the algorithms we implemented:

- PageRank
- HITS
- SALSA
- Randomized HITS
- SimRank

All of them were described in previous sections so I will not go further at this point.

### 3.3 HubRank

PageRank has demonstrated to be one of the most important rank algorithms developed so far in order to find good authorities. However it can be modified to provide better results focusing on different aspects. Our idea here is to bias the PageRank algorithm in order to rank higher pages that are likely hubs.

Let me remember you the equation of PageRank (also equation 2.2)

$$\mathbf{r}^{n+1} \leftarrow (c - 1) \cdot A \cdot \mathbf{r}^n + c \cdot \mathbf{e} \quad (3.1)$$

where  $\mathbf{e}$  represents the personalization vector used for the teleportation factor.

In the original algorithm this vector was the uniform distribution with  $\frac{1}{n}$  in each entry (where  $n$  is the total number of pages). Our approach (what we called HubRank) is to give a personalization vector that shows some user preference for hubs. Therefore, for each entry  $i$  of the personalization vector  $e$  the value  $e_i = O_i \frac{n}{|O|}$  where  $|O|$  is the summation of the outgoing links over the whole web graph. As the normal PageRank focus on the authorities (first part of the formula 3.1) our intend is to focus on the hubs in the second part.

The idea behind this formula is that we can use the personalization vector to bias the PageRank algorithms with any user preference. In our case, the user prefers to rank higher pages that are likely hubs so they will have a high number of outgoing links. A similar approach can be inferred to rank higher pages that are likely authorities (so they will have a high number of outgoing links). For this approach we can just substitute the previous formula and use  $e_i = I_i \frac{n}{|I|}$  instead of.

### 3.4 Proxy

In order to determine which are the interests of an user (his bookmarks are another way to determine them and we will use both together) we decided to develop a proxy that tracks down the user behaviour when he is surfing in internet. Basically, a user only has to configure his explorer to use this proxy. After this step, the pages surfed by the user will be written in a file as well as some basic information about them (like which pages visited the user or how much time was he looking at that page). From this log currently we extract the most visited pages (which are supposed to be preferred by the user) and we use this set of pages as a preference set for this user.

Some other applications to extract information of these logs have been also implemented.

### 3.5 HubFinder

Our intend here is to find the most important pages (hubs and authorities) related to the initial set of pages given to us by the user. In addition, we want to maintain the set of pages given by the user because they are supposed to be the most important ones from the user point of view. In order to do that we have developed a new algorithm (which we called HubFinder) that relies in the Kleinberg Extension presented previously. The algorithm is presented in the figure 3.1.

Basically we extend the initial set of pages applying the Kleinberg Extension algorithm and filter the results to keep important pages (importance in our context can mean global importance or importance regarding to the user interests) and discard the rest.

In each iteration we extend the subgraph with pages and keep only those pages that are important. These important pages will be extended and filtered again in the next iteration and so on. That way we try to construct a graph containing important pages regarding to the user preferences (as we

Subgraph( $\Gamma, k$ )

$\Gamma$ : initial set of pages

$k$ : number of iterations

Let  $KE$  be a function that applies the Kleinberg Extension once

Let  $Sel1$  and  $Sel2$  be functions that select important pages

$\Gamma = KE(\Gamma)$

$\tilde{\Gamma} = \Gamma$

For  $i = 1$  to  $k$

$\Gamma^* = KE(\tilde{\Gamma})$

$\Gamma^* = Sel1(\Gamma^*)$

$\Gamma = \Gamma + \Gamma^*$

$\tilde{\Gamma} = \Gamma^*$

$\Gamma = Sel2(\Gamma)$

return  $\Gamma$

Figure 3.1: Algorithm to find important pages to a initial subset

used the user input as an input). The last selection at the end of the algorithm tries to reduce the total amount of pages returned (in case it is too big). As at this point we would have the whole subgraph we could try with different approaches or heuristics.

### 3.6 Personalized Ranks

At this point there are two main problems with the description given in section 2.4.3:

1. The user has to choose between a given set of important hubs to receive personalized ranks. The problem of this approach is that the user may not have or find the right hubs among the given ones or even if there are some related ones the accuracy is not good enough.
2. The level of personalization depends of the size of the hubs set. In order to have a good personalization we have to have a really big hub set covering almost every field with enough detail.

Our approach intends to solver these two problems at the same time. As we provide the user bookmarks and user surfed pages as an input to find the

most important hubs, the accuracy of this process will be much higher as well that we do not force the user to choose or find his interests into a given set of hubs. In addition, even if the user has a small set of preference pages, it will be enlarge to reach a good amount of hubs and to assure a good level of personalization.

### 3.7 Summary of the process

As many new ideas have been presented I will try to give a brief summary of the whole process. Some of the steps could be skipped but in order to receive the best result all of them should be followed.

1. Crawl: as a basis for any of the ranking algorithms described we need to have a link structure crawled from the web. In addition, in order to have representative results this crawl should be big enough.
2. Calculate ranks: at this point the ranks chosen to be used during the process would be calculated.
3. Get bookmarks from users: an important step getting an input from the user.
4. Get user's surfing information: this is the second step getting the input from the user. In this case is not directly given by the user but it is tracked from the user behaviour when he is surfing the web.
5. Find the important hubs: here is where we apply the knowledge about the user we have received in order to find important hubs related to the user interests. We have followed two approaches:
  - Use HubFinder over the set of bookmarks.
  - Use HubFinder over the most visited pages by the user (using the information we tracked from him).

The first case seems to be more representative of the user interests so we decided to mix both approaches giving more importance to the first case. The way to give more importance to the first approach is to having a bigger amount of pages from the hubs found from the bookmarks than from the hubs found from the surfed pages.

6. Construction of the preference set: this set will be the union of the bookmarks set and the most visited pages by the user.

7. Construction of the basis hub set: at this point we need to create the hub set we will use from the information harvested by now. Our hub set will be formed as the union of the preference set and the important hubs returned by the HubFinder.
8. Compute similar pages: in order to improve the quality of the results we apply the SimRank algorithm to the basis hub set in order to get some other related pages (supposed to be also important hubs).
9. Construction of the hub set: the union of the basis hub set and the related hub pages found in the previous step forms the hub set.
10. Apply the personalized rank: apply the personalized rank described in section 2.4.3 using the preference set and the hub set as an input.

# Chapter 4

## Results

In this section I will describe and explain some results we got from the work we have been doing. Basically I will try to justify experimentally what I already explained theoretically in previous sections.

### 4.1 Webbase

In this section I would only like to present some numbers about the test bed we have built up. As I said we used the WebBase crawler with some improvements in order to retrieve the link structure of the web. As our web crawler is a focused web crawler we had to choose some basic set of domain sites to crawl.

We have executed three different times our modified crawler. The first one used the whole list of universities from Spain and Germany. This was a previous decision where we were going to focus on an educational domain. Afterwards we decided to increment our link structure with different domains extracted from the Open Directory Project [Ope]. The final link structure was extracted from a total of 4,106 different domain sites.

In our link structure we crawled 729,384 different pages and we found 3,587,842 different links (many of them pointing to a domains which were not crawled and from where we extracted the following sites to crawl).

Currently we are crawling another 4,887 new sites from the urls our previous crawlers discovered and did not crawl in order to increase our link structure.

In conclusion, our test bed is formed by the link structure of 729,384 different pages from 4,106 different domains.

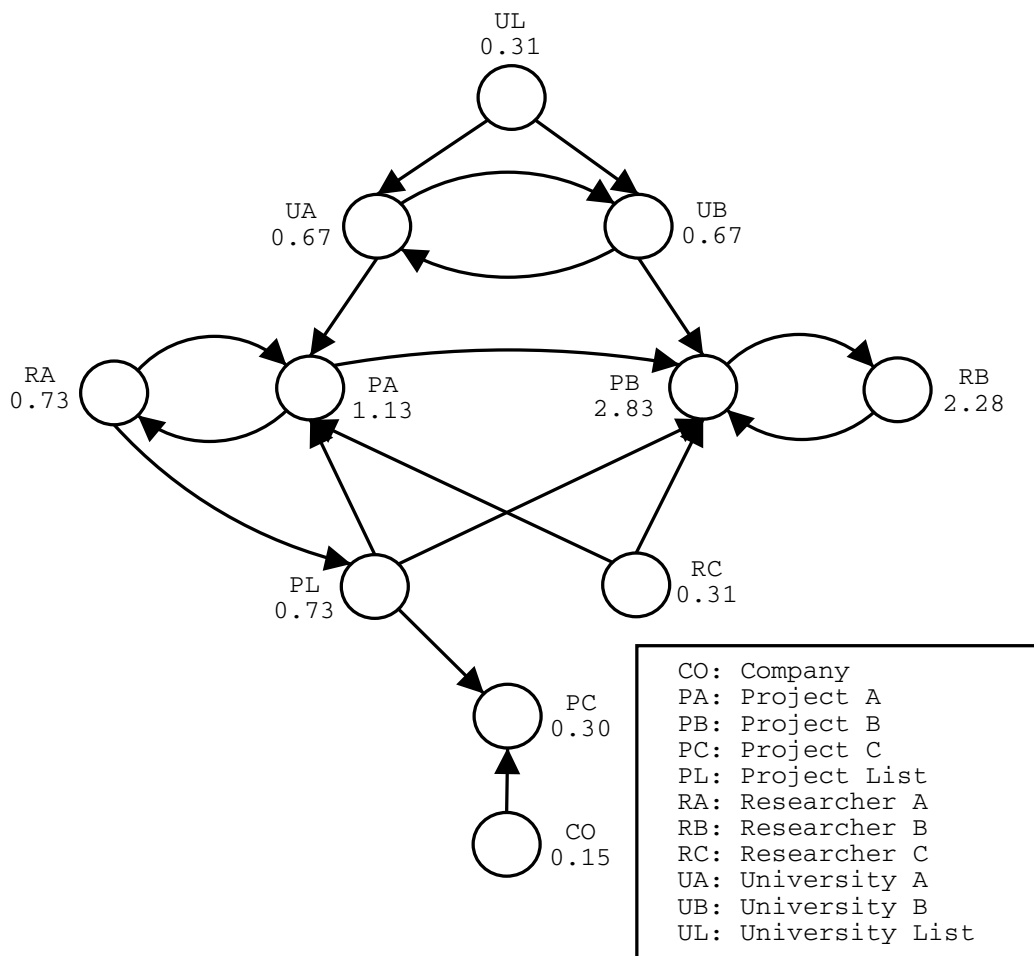


Figure 4.1: HubRank Example

## 4.2 HubRank

In this section I will present some results of the HubRank algorithm. For these examples we have executed the HubRank algorithm giving more importance to hubs using a dumping factor of 0.25 in order to increase a little bit the difference against the PageRank algorithm.

### 4.2.1 Small Graph

Let us start with the small graph used in the examples with other ranking algorithms. In figure 4.1 there are the result of applying the HubRank algorithm to this graph.

The table 4.1 is a comparison between all the results returned for each

Node	PageRank	HITS	SALSA	R.HITS	HubRank
Company	0.15	0.09	0.06	0.58	0.15
Project A	0.65	0.31	0.11	1.01	1.13
Project B	3.13	0.00	0.06	1.00	2.83
Project C	0.37	0.00	0.00	0.15	0.30
Project List	0.33	0.58	0.17	1.31	0.73
Researcher A	0.42	0.25	0.11	1.00	0.73
Researcher B	2.81	0.27	0.06	0.52	2.28
Researcher C	0.15	0.49	0.11	0.88	0.31
University A	0.37	0.27	0.11	0.92	0.67
University B	0.37	0.33	0.11	0.92	0.67
University List	0.15	0.11	0.11	0.95	0.31

Table 4.1: Score comparison among algorithms for the graph of figure 2.6

rank algorithm and the HubRank with the same graph (for the HITS, SALSA and Randomized HITS algorithms I have taken the hub scores because the results of the HubRank is biased on hubs).

In table 4.2 I show the order of nodes given by each rank algorithm and the HubRank (for the HITS, SALSA and Randomized HITS algorithms I have taken the hub scores).

Of course the graph was not chosen by chance. This graph shows many of the different properties a web graph can have with authorities and hubs of different level of importance.

First of all, let me describe what is expected in this small graph. In this graph we can easily see that the best authorities are *Project B* and *Researcher B* so they are supposed to be the most important pages. Therefore we would expect that these nodes had high scores. However, there are some hubs like *Project List* and the *Researcher C* for the projects and the *University List* for the universities which would be also quite interesting for the user (but only if they are important enough what means if they have an authority value high enough).

Let me go through the different rankings. As we can see HITS, SALSA and Randomized HITS classify the *Project List* node in the first position what is not bad because it is the best hub and it has a good authority value. However, HITS classify as the second to the node *Researcher C* that has not authority at all. SALSA does not distinguish between the next six nodes when it is clear that not all of them should have the same scores. Randomized HITS seems to behave much better but it classifies *Researcher A* and *Project A* higher than *Project B* and *Researcher B* which are higher

Node	PageRank	HITS	SALSA	R.HITS	HubRank
University List	9	8	2	5	8
University A	5	6	2	7	6
University B	5	3	2	6	6
Researcher A	4	7	2	3	5
Project A	3	4	2	2	3
Project B	1	10	8	4	1
Researcher B	2	5	8	10	2
Project List	8	1	1	1	4
Researcher C	9	2	2	8	8
Project C	7	11	11	11	10
Company	9	9	8	9	11

Table 4.2: Rank comparison between algorithms for the graph of figure 2.6

authorities. Moreover, all the three classify quite bad the highest authorities of the graph. HITS classifies *Project B* as the 10th, SALSA as the 8th and Randomized HITS does it well with *Project B* but it classifies *Researcher B* as the 10th. It is clear these are not the expected results.

Let me now go through the PageRank results. PageRank give the highest authorities to *Project B* and *Researcher B* what agree with our expectation. However, it ranks *Project List* as the 8th. Even it classifies better *University A* and *University B* what is not at all what we would like.

Now, let us take a look into the HubRank example. As we can see it is right giving the highest scores to the highest authorities (*Project B* and *Researcher B*). Moreover, it ranks well the *Project List* being the 4th. Even if its authority rank is not so high, its good quality as a hub makes it be ranked higher than other nodes with higher authority value. Let us go to the rest of the nodes. The other hubs *University List* and *Researcher C* are ranked as the 8th because they do not have authority value at all. *Project C* is ranked 10th because even although it does not have bad authority value, it has not hub quality at all.

As it can be seen, HubRank choose between the best authorities of the graph and biases the results giving more importance to such pages that are also good pages. It proves to be more accurate than any of the other algorithms presented so far.

### 4.2.2 Big crawl

I will present now the results of the different ranks over the whole link structure crawled with our version of the WebBase crawler.

Some results from the PageRank point of view are in the table 4.3. Let me remind the reader that these results have been computed with a big link structure with 4,106 domains what is good enough to test our algorithms but what is of course not enough to be compared with current search engines. I said that because the reader will probably miss pages like for example “www.yahoo.com” or “www.netscape.com” among the highest ranked urls but it happens because these domains were not crawled. The reader can also think that probably the site “www.macromedia.com” should be ranked higher but in our set of domains crawled there were not so many sites pointing to it so that is why it is not in the top authorities.

Pages marked with an asterisk (\*) were ranked outside of the 50000 pages.

In this example we can see how HubRank seems to behave really well. If we take a look to the best authorities found by PageRank, we can also see that HubRank rank almost all of them good (what is not done by the other algorithms: HITS, SALSA or Randomized HITS). Moreover, we can see that the hubs found by HITS among the pages shown in the table are also well ranked by HubRank. Here it is clear that HubRank filter the results of PageRank giving more importance to hubs.

Some results, but this time from the point of view of the HubRank algorithm are presented in the table 4.4.

If we check now this new list from the point of view of the HubRank algorithm we can observe that all the urls classify high by the HubRank are also important pages from the point of view of the PageRank algorithm (are good authorities). In addition, these pages have also good quality as a hubs as we can see with the results from the other algorithms where almost all the entries are classified by HITS, SALSA and Randomized HITS. Of course in our examples we are giving more importance to the authority value than to the hub value (the dumping factor used is 0.25) and that is why the ranks from the other algorithms are not the best to our pages (they measure good hubs, but HubRank measures a mix between good authorities and good hubs). A good example of this theory is the site “www.java.sun.com” where it is more or less equal by PageRank and HubRank because is a good authority and also a good hub. Another example is the cases of “www.planning.org” which turns out to be a really good hub for the architecture community and it was ranked quite badly by PageRank.

Url	PageR	HITS	SALSA	RHITS	HubR
www.trendmicro.com/en/home/...	1	*	*	*	26
www.trendmicro.com/en/about/...	2	*	*	*	27
edis.ifas.ufl.edu	3	*	14895	974	618
www.w3.org	4	33716	8982	1665	36
www.planum.net/lato.htm	5	*	*	8144	2106
www.amoebasoft.com	6	11204	*	*	1
...					
www.oribtz.com	37	*	23406	269	2522
www.acm.org	38	11081	34352	6128	87
www.morrisnathansondesign.com	39	*	*	*	2661
www.planning.org	40	*	3472	16064	3
www.aardvarktravel.net	41	*	*	*	174
www.umass.edu	42	*	*	10916	273
www.voxengo.com/phorum/...	43	*	27077	19252	91
www.macromedia.com	44	*	*	9319	49
...					
www.gardenvisit.com	99	*	13213	1613	2113
www.steinberg.net	100	*	42036	6938	5344
www.kaspersky.com	101	*	35894	26285	86
...					
www.ecoiq.com/syndicated...	1000	*	*	*	2080
www.bangaloreit.com/...	1001	*	33467	2175	7179
www.wired.com/animation/...	1002	44055	*	*	3735

Table 4.3: Scores from the ranking algorithms over the whole graph

Url	HubR	HITS	SALSA	RHITS	PageR
www.amoebasoft.com	1	11204	*	*	6
www.amoebasoft.com/index.asp	2	11205	*	*	7
www.planning.org	3	11081	16064	16064	40
...					
www.trendmicro.com/en/about...	27	*	*	*	2
www.visualbuilder.com	28	2292	4938	8804	618
www.java.sun.com	29	32365	*	*	32
...					
www.voxengo.com/phorum.index...	35	*	*	*	31
www.w3.org	36	33716	8982	1665	4
www.maps.com	37	27708	24402	5285	105
...					
stats.bls.gov/iif/home.htm	10000	*	26794	5835	9572
www.ecoiq.com/landuse/magazine...	10001	39124	8642	3974	12069
www.scottlondon.com/bio/index...	10002	*	*	*	4497
...					
www.jspin.com/home/apps/filemani	49999	49509	*	7239	43276
www.stuffit.com/compression/file...	50000	*	*	*	39121

Table 4.4: Scores from the ranking algorithms over the whole graph

### 4.2.3 Conclusion

The conclusion would be that our HubRank improve the existing algorithms giving a new measure where authority value and hub quality are taken into account. It improves the HITS, SALSA and Randomize HITS algorithms taken into account not only the hub quality but also the authority value, so HubRank assures that only important pages will be returned. It shows to be better than PageRank filtering its results and giving more importance to pages with high hub quality. In fact, as we can see in the examples, not all the highest ranked pages from PageRank are good ranked by HubRank (because PageRank only measures authority) but all the pages ranked high by HubRank are also good ranked by PageRank (because we do take into account the authority value of the pages).

It is worth to remark that we can personalize the degree or accuracy of this algorithm varying the dumping factor  $c$  used in order to give more importance to authorities over hubs or vice versa.

Regarding to the computation time, as we use the same basis of the PageRank algorithm, the computation time is almost the same.

## 4.3 HubFinder

Let me remember which is the purpose of this algorithm. The goal is to find as many hubs as possible related to a starting set of pages. In addition, these hubs should be important enough to be representative.

In order to choose the pages we will keep after each iteration we will use different algorithms over the whole graph and keep the best ranked ones. That is why we have executed the algorithm once with each of them.

We used the pages from the bookmark set (shown in the table 4.5) as a starting pages. Then we applied the HubFinder algorithm to this starting pages with a different algorithms in our *Select* function so we used the ranks provided by these algorithms to decide which pages are kept and which ones are discarded. In table 4.6 are shown the final results.

The *Total* shows the results to apply HubFinder with each algorithm. Every page in each round has to fulfil some requirements or it will be discarded. In this example we check this pages to have a good rank (high enough) and a minimum out-degree.

As we can see Randomized HITS and HubRank are the best ones. While HITS and SALSA will give us really good hubs but maybe not so important (low authority) PageRank will give us only good authorities so any of them is not desirable for our case. Let us know take a look into the Randomized HITS

<a href="http://www.domus3d.com/default.asp">www.domus3d.com/default.asp</a>
<a href="http://www.gardenvisit.com">www.gardenvisit.com</a>
<a href="http://www.urban-advantage.com/index.html">www.urban-advantage.com/index.html</a>
<a href="http://www.metropolismag.com/html/designmart/index.html">www.metropolismag.com/html/designmart/index.html</a>
<a href="http://phillipsplanning.com/pastprojects1.html">phillipsplanning.com/pastprojects1.html</a>
<a href="http://www.museumofarchitecture.com/main.html">www.museumofarchitecture.com/main.html</a>
<a href="http://deck-porch-gazebo-plan.vadeck.com">deck-porch-gazebo-plan.vadeck.com</a>
<a href="http://www.wrdanielsdesign.com/gallery.html">www.wrdanielsdesign.com/gallery.html</a>
<a href="http://www.architekturmuseum.ch/htmls/indexe.htm">www.architekturmuseum.ch/htmls/indexe.htm</a>
<a href="http://www.bauhaus.de/english/museum/index.htm">www.bauhaus.de/english/museum/index.htm</a>
<a href="http://www.bauhaus.de/english/bauhaus1919/architektur/index.htm">www.bauhaus.de/english/bauhaus1919/architektur/index.htm</a>
<a href="http://chi-athenaeum.org/gdesign/gdesin0.htm">chi-athenaeum.org/gdesign/gdesin0.htm</a>
<a href="http://www.archiradar.com/eng/index.htm">www.archiradar.com/eng/index.htm</a>
<a href="http://gdlalliance.com">gdlalliance.com</a>
<a href="http://www.aztechsoft.com/gpscdrv.htm">www.aztechsoft.com/gpscdrv.htm</a>
<a href="http://www.fatcad.com/home.asp">www.fatcad.com/home.asp</a>
<a href="http://www.cadfx.com/2004.phtml">www.cadfx.com/2004.phtml</a>
<a href="http://www.command-digital.com/panorama1.htm">www.command-digital.com/panorama1.htm</a>
<a href="http://www.contractcaddgroup.com/3d.htm">www.contractcaddgroup.com/3d.htm</a>
<a href="http://www.e-magine.ws/products.htm">www.e-magine.ws/products.htm</a>
<a href="http://www.atlanticarchitects.com/about.htm">www.atlanticarchitects.com/about.htm</a>
<a href="http://www.architecture.com/go/Architecture/Reference/Library_898.html">www.architecture.com/go/Architecture/Reference/Library_898.html</a>

Table 4.5: Bookmarks set

	<b>PageRank</b>	<b>HITS</b>	<b>SALSA</b>	<b>Randomized HITS</b>	<b>HubRank</b>
Total	632	334	334	676	650

Table 4.6: HubFinder comparison with different algorithms

and HubRank results. Randomized HITS give us more results. Depending on which are our preferences we could choose between any of both algorithms. Randomized HITS seems to give more hubs while HubRank give a good amount of hubs (less than Randomized HITS) but it assures that all of them will be good hubs and also good authorities.

Regarding to computation time and performance HubFinder is faster than the original algorithm. The reason is simple, as we are filtering the graph at each step, we are taking a little bit more of time to select which pages are kept and which ones are discarded but we save much more time in next executions having a smaller set. We have found HubFinder up to 33% faster in most of the situations where we were running it.

## 4.4 Personalized Ranks

In order to test the final results of the whole process presented here we have execute the algorithm described in 2.4.3 with two different input sets. As a reference the first time we did it with the 30 top pages ranked by the PageRank algorithm as a preference set and the top 250 pages ranked also by the PageRank algorithm as the hubs set.

In the second execution (with our results) we used the results obtained in previous steps. The resulting set from the union of the bookmarks and the most surfed pages contains a total of 79 pages. This set was used as an input for the preference set. The hubs set was the union of the preference set and the output computed from the HubFinder algorithm over the bookmarks and over the most surfed pages. This set contained 493 different pages.

The table 4.7 contains the results of these executions:

As we they are really different (as it was expected because they used different sets as an input). All the pages from our preference set used as an input for the algorithm are among the top 300 ranked pages. This is also what we expected because they are suppose to be the most important pages for us and the pages related to them should also have higher rank.

As the user can have noticed the SimRank algorithm has not been used so far. The reason is that the performance of the algorithm is not good enough to be executed with so big graph so it was only tested with small crawls. More work has to be done on this aspect.

Url	Original	Ours
www.trendmicro.com/en/home/global/legal.htm	1	*
www.trendmicro.com/en/about/privacy/overview.htm	2	*
www.amoebasoft.com	3	*
www.amoebasoft.com/index.asp	4	*
...		
www.domus3d.com/default.asp	*	66
www.gardenvisit.com	*	9
www.urban-advantage.com/index.html	*	24
www.metropolismag.com/html/designmart/index.html	*	29
phillipsplanning.com/pastprojects1.html	*	72
www.museumofarchitecture.com/main.html	*	3
deck-porch-gazebo-plan.vadeck.com	*	30
www.wrdanielsdesign.com/gallery.html	*	21
www.architekturmuseum.ch/htmls/indexe.htm	*	70
www.bauhaus.de/english/museum/index.htm	25790	69
www.bauhaus.de/english/bauhaus1919/architektur/index.htm	26457	287
chi-athenaeum.org/gdesign/gdesin0.htm	*	50
www.archiradar.com/eng/index.htm	*	44
gdlalliance.com	*	478
www.aztechsoft.com/gpscdrv.htm	*	86
www.fatcad.com/home.asp	*	11
www.cadfx.com/2004.phtml	*	216
www.command-digital.com/panorama1.htm	*	19
www.contractcaddgroup.com/3d.htm	*	79
www.e-magine.ws/products.htm	*	100
www.atlanticarchitects.com/about.htm	*	115
www.architecture.com/go/Architecture/Reference/Library_898.html	26954	34
...		
java.sun.com/downloads	4405	171

Table 4.7: Personalized Ranks

## Chapter 5

# Conclusions and Further work

In this document we have explained the current state of the ongoing work in the data-mining field focused on search engines and rank algorithms. We have described the current processes used and which are the main and biggest problems with them. We have proposed some lines to follow in order to solve them or at least to improve them. A whole process from the beginning to the end has been presented and analysed in order to provide better results. In addition, we have experimented with real cases (not only theoretical examples) and real data proving that our solution is feasible and that it works well. Under these assumptions we have proved that our solution behaves better than the existing ones.

Some of the next steps in our research include the following:

1. **Increase the size of the crawl:** the current crawl is big enough to check if our algorithm is feasible and to receive some basic results. However, in order to have a real test we need to use a bigger crawl. Currently we are retrieving data from the web in order to enlarge the current link structure we already have. Then, we will test again our algorithm in order to check not only the results against a bigger graph but also to check performance (basically computation time and memory usage).
2. **SimRank improvement:** as we said we have not used the SimRank in our experiments because the current version (the one given by the author) is not good enough to be computed over the whole graph due to a big needs of resources. We are studying different ways to improve it in order to implement it and added to our experiments.
3. **More information from the user:** currently our implementation uses only the most visited pages from an user extracted from the proxy.

We could think in other information we could get from it like for example some feedback from the search engine about which pages does he choose from the whole list of results.

4. **More experiments and evaluation:** of course the results shown in this document are only preliminary. More experiments are needed and with bigger set of data have to be done in order to be able to prove that our algorithm is really more convenient that the existing ones. Even, user evaluation would be a good idea in order to get real feedback from people who are not involved in the research itself.

# Bibliography

- [ACGM<sup>+</sup>00] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the web. Technical report, Stanford Digital Library Technologies Project, 2000.
- [Alt] Altavista search engine. <http://www.altavista.com/>.
- [BKM<sup>+</sup>00] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, and R. Stata. Graph structure in the web. In *In Proceedings of the 9th International World Wide Web Conference*, 2000.
- [BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
- [Bro02] Andrei Broder. A taxonomy of web search. Technical report, IBM Research, 2002.
- [CC00] David Cohn and Huan Chang. Learning to probabilistically identify authoritative documents. In *Proc. 17th International Conf. on Machine Learning*, pages 167–174. Morgan Kaufmann, San Francisco, CA, 2000.
- [CDG<sup>+</sup>98] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th International World Wide Web Conference*, 1998.
- [CGMP98] Junghoo Cho, Hector García-Molina, and Lawrence Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.
- [CK97] J. Carriere and R. Kazman. Webquery: Searching and visualizing the web through connectivity. In *Proceedings of the International WWW Conference*, 1997.

- [DDL<sup>+</sup>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(6):391–407, 1990.
- [Diga] Digital library initiative.  
<http://www.dli2.nsf.gov/dlione/>.
- [Digb] Digital library initiative phase 2.  
<http://www.dli2.nsf.gov/projects.html>.
- [Exc] Excite search engine. <http://www.excite.com/>.
- [GL] Jean-Loup Guillaume and Matthieu Latapy. The web graph: an overview.
- [Goo] Google search engine. <http://www.google.com>.
- [Hav02] T. Haveliwala. Topic-sensitive pagerank. In *In Proceedings of the Eleventh International World Wide Web Conference, Honolulu, Hawaii*, May 2002.
- [HRGMP00] Jun Hirai, Sriram Raghavan, Hector García-Molina, and Andreas Paepcke. WebBase: A repository of web pages. In *Proceedings of the Ninth World-Wide Web Conference*, 2000.
- [Inf] Infoseek search engine. <http://infoseek.go.com/>.
- [JW01] Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. Technical report, Stanford University Database Group, 2001.
- [JW02a] G. Jeh and J. Widom. Scaling personalized web search. Technical report, Stanford University, 2002.
- [JW02b] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity, 2002.
- [Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
- [LM00] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):387–401, 2000.

- [LV00] Peter Lyman and Hal R. Varian. How much information, 2000.
- [LW01] Mark Levene and Richard Wheeldon. Web dynamics, 2001.
- [Lyc] Lycos search engine. <http://www.lycos.com/>.
- [Mar] Peter Marendy. A review of world wide web searching techniques, focusing on hits and related algorithms that utilise the link topology of the world wide web to provide the basis for a structure based search technology.
- [MM00] B. H. Murray and A. Moore. Sizing the internet, July 2000.
- [NZJ01a] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. Link analysis, eigenvectors and stability. In *IJCAI*, pages 903–910, 2001.
- [NZJ01b] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. Stable algorithms for link analysis. In *Proc. 24th Annual Intl. ACM SIGIR Conference*. ACM, 2001.
- [Ope] Open directory project. <http://dmoz.org/>.
- [PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
- [Pin94] Brian Pinkerton. Finding what people want: Experiences with the web crawler. In *The second International WWW Conference Chicago*, 1994.
- [PTRV98] Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. pages 159–168, 1998.
- [Sim] Simple crawler and search engine.  
<http://www.learninglab.de/~olmedilla/projects/search/search.php?lang=en&menu=pro&menu2=sea>.
- [Sta] Stanford webbase project.  
<http://www-diglib.stanford.edu/testbed/doc2/webbase/>.
- [VR79] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

- [Wha]      What's the value of data resource management?  
[http://www.dama.org/data\\_facts\\_you\\_can\\_use.htm](http://www.dama.org/data_facts_you_can_use.htm).
- [Yah]      Yahoo! search engine. <http://www.yahoo.com>.

## Extra Bibliography

- [AFK<sup>+</sup>01] Yossi Azar, Amos Fiat, Anna R. Karlin, Frank McSherry, and Jared Saia. Spectral analysis of data. In *ACM Symposium on Theory of Computing*, pages 619–626, 2001.
- [AM01] Achlioptas and McSherry. Fast computation of low rank matrix approximations. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2001.
- [And] Taher Haveliwala And. The second eigenvalue of the google matrix.
- [APDR77] N. M. Laird A. P. Dempster and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B*, 39(1):1–38, Nov. 1977.
- [BB98] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *Proc. of the 7th World-Wide Web Conference (WWW7)*, 1998.
- [BDO94] Michael W. Berry, Susan T. Dumais, and Gavin W. O’Brien. Using linear algebra for intelligent information retrieval. Technical Report UT-CS-94-270, 1994.
- [BH98a] Krishna Bharat and Monika R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, AU, 1998.
- [BH98b] Andrei Z. Broder and Monika Rauch Henzinger. Information retrieval on the web. In *IEEE Symposium on Foundations of Computer Science*, page 6, 1998.

- [BLCL<sup>+</sup>94] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, 1994.
- [BRRT01] Alan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Finding authorities and hubs from link structures on the world wide web. In *World Wide Web*, pages 415–429, 2001.
- [CDG<sup>+</sup>98] Soumen Chakrabarti, Byron E. Dom, David Gibson, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Experiments in topic distillation. In *In SIGIR workshop on Hypertext Information Retrieval*, 1998.
- [CDK<sup>+</sup>99] Soumen Chakrabarti, Byron E. Dom, S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew Tomkins, David Gibson, and Jon Kleinberg. Mining the Web’s link structure. *Computer*, 32(8):60–67, 1999.
- [CDKS] Steve Chien, Cynthia Dwork, Ravi Kumar, and D. Sivakumar. Towards exploiting link evolution.
- [CGS] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. I/o-efficient techniques for computing pagerank.
- [DFK<sup>+</sup>99] Drineas, Frieze, Kannan, Vempala, and Vinay. Clustering in large graphs and matrices. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1999.
- [DHH<sup>+</sup>02] Chris Ding, Xiaofeng He, Parry Husbands, Hongyuan Zha, and Horst Simon. PageRank, HITS and a unified framework for link analysis. Technical Report 49372, LBNL, 2002.
- [DKNS01] Cynthia Dwork, S. Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *World Wide Web*, pages 613–622, 2001.
- [FHP] Lisa K. Fleischer, Bruce Hendrickson, and Ali Pinar. A divide-and-conquer algorithm for identifying strongly connected components.

- [FLG00] Gary Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, Boston, MA, August 20–23 2000.
- [FLGC02] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 35(3):66–71, 2002.
- [GKR98] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *UK Conference on Hypertext*, pages 225–234, 1998.
- [GL89] C. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1989.
- [G.S93] G.Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1993.
- [Hav99] Taher Haveliwala. Efficient computation of pageRank. Technical Report 1999-31, 1999.
- [Hen00] Monika Rauch Henzinger. Web information retrieval. In *ICDE*, page 693, 2000.
- [HMS] Monika R. Henzinger, Rajeev Motwani, and Craig Silverstein. Challenges in web search engines.
- [KC96] R. Kazman and J. Cartiere. An adaptable software architecture for rapidly creating information visualizations. In *Proc. 5th International World Wide Web Conference*, 1996.
- [KHM03] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank, 2003.
- [KKR<sup>+</sup>99] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–??, 1999.
- [Kle00] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

- [KRR<sup>+</sup>00] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, and Eli Upfal. The Web as a graph. In *Proc. 19th ACM SIGACT-SIGMOD-AIGART Symp. Principles of Database Systems, PODS*, pages 1–10. ACM Press, 15–17 2000.
- [KRRT99] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1481–1493, 1999.
- [KSGM03a] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. Eigenrep: Reputation management in p2p networks. In *Proc. World-Wide Web Conference, 2003*.
- [KSGM03b] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks, 2003.
- [KT99] Jon M. Kleinberg and Andrew Tomkins. Applications of linear algebra in information retrieval and hypertext analysis. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 185–193. ACM Press, 1999.
- [Mor] More evil than dr. evil?.  
<http://searchenginewatch.com/sereport/99/11-google.html>.
- [SFA97] M. Swain, C. Frankel, and V. Athitsos. Webseer: An image search engine for the world wide web, 1997.
- [TEO] Teoma. <http://www.teoma.com/>.
- [Tre] Trends in the evolution of the public web.  
<http://www.dlib.org/dlib/april03/lavoie/04lavoie.html>.