Web Mining and Exploration: Algorithms and Experiments.

Debora Donato
Web Mining and Exploration: Algorithms and Experiments.
AUTHOR’S ADDRESS:

Debora Donato
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
E-MAIL: donato@dis.uniroma1.it
WWW: http://www.dis.uniroma1.it/~donato
## Contents

1 Introduction .............................................. 5

I Static and Dynamical Properties of the Webgraphs .......... 9

2 Large Scale Properties of the Web .......................... 11
   2.1 Preliminaries ........................................ 11
   2.2 The statistical and topological properties of the Web graph .... 14
      2.2.1 Indegree and outdegree .................. 14
      2.2.2 Pagerank .................................... 15
      2.2.3 Bipartite cliques ......................... 16
      2.2.4 Strongly connected components ............. 17
   2.3 Conclusions ....................................... 18

3 Secondary Memory Algorithms for Measuring Webgraphs .... 21
   3.1 Data representation and multifiles .................. 22
   3.2 Traversal with two bits for each node ............... 23
   3.3 Semi-external Breadth First Search ................ 23
   3.4 Semi-external depth first search ................... 24
   3.5 Computation of the SCCs .......................... 24
   3.6 Computation of the Bow-Tie regions ................. 25
   3.7 Disjoint bipartite cliques ........................ 26
   3.8 Pagerank ....................................... 27
   3.9 Conclusions ..................................... 28

4 Data Stream Computation for Measuring Webgraphs ....... 29
   4.1 Data Stream Algorithms ............................ 29
   4.2 Estimating rarity over data stream windows .......... 30
   4.3 Computing the rarity distribution of indegree and bipartite cliques .... 31
   4.4 Experimental Results .................................. 32
       4.4.1 Optimization of Min-wise hashing ......... 33
       4.4.2 Datasets ................................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.3</td>
<td>Rarity Indegree Distribution</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusion</td>
<td>37</td>
</tr>
<tr>
<td>5</td>
<td>Mining the Inner Structure of the Webgraph</td>
<td>41</td>
</tr>
<tr>
<td>5.1</td>
<td>The bow-tie picture</td>
<td>41</td>
</tr>
<tr>
<td>5.2</td>
<td>Experiments and results</td>
<td>43</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Macroscopic measurements</td>
<td>44</td>
</tr>
<tr>
<td>5.2.2</td>
<td>The inner structure of the bow-tie</td>
<td>44</td>
</tr>
<tr>
<td>5.3</td>
<td>Conclusion</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>Temporal Analysis of Wikigraphs</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>The role of the time in the study of the Web</td>
<td>53</td>
</tr>
<tr>
<td>6.2</td>
<td>Related Work</td>
<td>55</td>
</tr>
<tr>
<td>6.3</td>
<td>The Data Sets</td>
<td>56</td>
</tr>
<tr>
<td>6.4</td>
<td>Link Analysis of Wikigraphs</td>
<td>58</td>
</tr>
<tr>
<td>6.5</td>
<td>Temporal Information in the Wikigraphs</td>
<td>62</td>
</tr>
<tr>
<td>6.6</td>
<td>Temporal Analysis of Wikigraphs</td>
<td>64</td>
</tr>
<tr>
<td>6.7</td>
<td>Conclusions</td>
<td>70</td>
</tr>
<tr>
<td>II</td>
<td>Link Analysis Ranking Algorithms</td>
<td>77</td>
</tr>
<tr>
<td>7</td>
<td>Stability and Similarity of Link Analysis Ranking Algorithms</td>
<td>79</td>
</tr>
<tr>
<td>7.1</td>
<td>The Stability and Similarity Properties</td>
<td>79</td>
</tr>
<tr>
<td>7.2</td>
<td>Preliminaries</td>
<td>81</td>
</tr>
<tr>
<td>7.3</td>
<td>The theoretical framework</td>
<td>82</td>
</tr>
<tr>
<td>7.4</td>
<td>Stability and similarity on the product graphs</td>
<td>85</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Mathematical Tools</td>
<td>86</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Conditions for the stability of HITS</td>
<td>88</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Similarity of HITS and LATENT</td>
<td>89</td>
</tr>
<tr>
<td>7.4.4</td>
<td>Similarity of INDEGREE and LATENT</td>
<td>90</td>
</tr>
<tr>
<td>7.4.5</td>
<td>Power law graphs</td>
<td>92</td>
</tr>
<tr>
<td>7.5</td>
<td>Experimental analysis</td>
<td>92</td>
</tr>
<tr>
<td>7.6</td>
<td>Similarity of HITS and INDEGREE</td>
<td>93</td>
</tr>
<tr>
<td>7.7</td>
<td>Conclusions</td>
<td>95</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>97</td>
</tr>
</tbody>
</table>
Abstract

In this thesis we present an extensive study of the World Wide Web. After presenting some of its main previously known properties, experimental work is documented. Static and dynamic properties of different samples, collected during the last four years, are examined by software implementations of a number of algorithms devised to work in secondary memory or in a datastream fashion.

In addition, the knowledge acquired by this large-scale study of the different features of the Web, allows us to analyze some of the the problems concerning two of the most famous link analysis algorithms for ranking Web documents, Pagerank [16] and HITS [45]. We present some results concerning the stability and similarity of these algorithms.
Publications related with this thesis

The results collected in this thesis have been the subject of a number of papers, published in various journals or accepted to recent conferences and workshops.

The large scale properties of the Web described in Chapter 2 were the subject of a paper appeared on the European Journal of Physics B (published by EDP Sciences, Societ Italiana di Fisica and Springer-Verlag) and was also accepted for publication in the Journal of Graph Algorithms and Applications (published by World Scientific Publishing) and in ACM Transaction on Internet Technology (TOIT).

The paper also provided a description of the external memory algorithms, detailed in Chapter 3, that we have used to measure huge samples of the Web (and, in general, massive graphs). The dis_library developed during this thesis is available under GNU license and can be freely downloaded at: http://www.dis.uniroma1.it/~cosin/html_pages/COSIN-Tools.htm. A description of the data structures and the detailed instructions to run all the implemented functions are in [33].

The application of data stream techniques to measure the Web properties, illustrated in Chapter 4, was presented in the Workshop on Massive Geometric Data Sets (MASSIVE05, in connection with SoCG 05).

The study of the inner structure of the bow-tie on Chapter 5 was introduced in the eight International Workshop on the Web and Databases (WebDB05, in conjunction with ACM SIGMOD Conference).

The temporal Analysis of Wikipedia on Chapter 6 has been submitted to the SIGKDD Explorations special issue on Link Mining.

Finally the theoretical study of the Link Analysis Ranking (LAR) algorithms, on Chapter 7 was presented in the 32nd International Colloquium on Automata,
Languages and Programming [54] (ICALP’05).
Chapter 1

Introduction

Since its first inception at the end of ’60s, the World Wide Web, commonly known as the Internet, has clearly made evident its evolving nature. In the past decade the world has witnessed the explosion of the Web from an information repository of a few millions of hyperlinked documents into a massive world-wide “organism” that serves informational, transactional, and communication needs of people all over the globe. The numbers of this dramatic grow are impressive: at beginning of 1998 one of the estimates of the Web size was more than 320 million pages. At the end of 1999 the number of vertices was estimated around one billion. At the time of the writing of this thesis the estimate is 11 billion pages. Naturally, the study of the Web has attracted the attention of different scientific communities, from mathematical to life and social sciences, in the attempt to understand the laws that rule its structure and evolution. Nowadays the Web can be seen as the largest data repository ever devised. Its skeleton consists of HTML pages among which users can surf following the hyperlinks. Obviously the Web is more than this, with pages that can be dynamically created on demand and that disappear after they are consulted. Nevertheless most of the research has focused, during the last four years, on the analysis of the static Web.

Motivation. A natural activity associated with a data repository is information retrieval. In an highly unstructured and distributed environment as the Web, this operation, complex by itself when dealing with massive data, becomes even more difficult. As a consequence, the development of more and more efficient Web search engines has been always considered as one of the more challenging tasks. Such an interest for search engine techniques has incentivized efforts to gain a deeper insight in three of the different, but often overlapping, research fields related to the Web:

- the study of the topological and statistical properties of the Web;
- the design of ranking algorithms as Pagerank and HITS;
the refinement of algorithms and techniques more strictly related to the search engine activities as crawling strategies, indexation, clusterization and classification.

We want to remark that this three research lines, as mentioned, have several overlapping aspects. Indeed, understanding the structure and the evolution of the Webgraph is a fascinating problem by itself and, at the same time, all the acquired knowledge finds practical applications in devising better crawling strategies [50], performing clustering and classification [50] and improving browsing [20]. Furthermore the most famous ranking algorithms, developed to make more effective the answers to users queries, rely on the link structure of the Web. The knowledge of the macroscopic structure of the Web has been used in devising efficient algorithms for the computation of PageRank [44, 35].

The analytical and experimental work that converges in this thesis, has been focused on the first two research lines above outlined. Nevertheless the connections with the third field are more than evident.

State of art. The statistical properties of the Web are investigated by means of the Webgraph. A Webgraph is simply a directed graph whose nodes are the (static) HTML pages and whose edges are the hyperlinks among them, directed from the page that contains the link to the target of the link. The study of the Web was originated by the seminal work of Kleinberg et al. [46]. In this work, the term Webgraph was coined and the main features of this object were listed in a homogeneous framework.

The study of the topological structure of this graph has revealed that power laws, typical of scale-free networks, emerge in most of the significant measures (e.g. distribution of indegree, PageRank). Moreover the studies performed by Broder et al. [18] allowed to see its macroscopic structure: the Webgraph is organized in a bow-tie shape, with a CORE, a single strongly connected component, comprised by more then a quarter of the nodes. As many Web features were discovered and analyzed, the availability of stochastic models, able to capture all these characteristics, became extremely important. The devising of these models has originated another important research branch that however is outside the scope of this thesis. So, as mentioned, from the 1992 to the 2002, a huge quantity of studies has focused on the Web. As one might have expected, the great attention paid to Web matters and the deep investigations have ended up to highlight new issues and problems needing further analysis.

Open issues when I started my thesis. The list of the “at that moment” open issues is attempted below:

- All the measures had been accomplished on old and quite small samples of the Web. So, in order to monitor the evolution of such a dynamical artifact as the Web had proved to be, it became necessary to refresh all the previous measurements on more recent and wider crawls.
The studies of Broder et al. [18] had allowed to get knowledge of the macroscopic structure of the Web, pictured by the bow-tie shape, but had not provided any detail of the inner structure of the bow-tie subsets.

Very little work had been devoted to analyze the temporal evolution of the properties of the Web. This requires the design of dynamic algorithms and data structures that allow us to efficiently store, retrieve and update this huge amount of information.

**Contribution of the thesis.** The first part of this thesis aims to address part of the above outlined open issues. My contribution can be summarized in the following main points:

- We extend the measures presented in previous works on a number of more recent and wider crawls. This work requires the use and the development of sophisticated algorithmic techniques to handle large data sets, namely external memory algorithms, streaming algorithms, random sampling and property testing. In particular we implemented some external algorithms, i.e. algorithms that do not require a quantity of main memory that depends on the input size; the result of our efforts is a C++ library [33]. Moreover, we use datastream techniques for the approximate computation of some interesting measures as the indegree distribution or the number of bipartite cliques.

- We analyse the inner structure of the bow-tie sets. We show that a Webgraph do not exhibit self-similarity within the IN and OUT components and we propose a possible alternative picture of the Web as it emerges from our experiments.

- The temporal evolution of the properties of the Web is one of the most difficult question to address due to lack of temporal data. It requires to measure different snapshots of the same set of pages. Crawling large portions of the Web is a demanding task both in term of bandwidth, storage space and computational resources. For this reason we restrict our investigation to Wikipedia, an online encyclopedia, downloadable from the Web. Since 2001, when it was born, all the page updates have been recorded. So it is possible to generate temporal snapshots of Wikipedia. We use these snapshots, called Wikigraphs, to monitor the evolution over time of the topological and statistical properties. We devise a power law distribution on the number of vertices receiving a given number of updates and found out that the number of updates of a page is not correlated with the PageRank, indegree, outdegree and the number of visits of that page.

We stress again that the study of the dynamics of the Web is a crucial issue. In the second part of this thesis, we study how perturbations of the topological structure
of the Web impacts on ranking algorithm performance. We accomplish a theoretical analysis of the Link Analysis Ranking (LAR) algorithms. LAR algorithms assign a ranking to pages only on the basis of the topological structure of the Webgraph. It’s worth noting that a good ranking is the main ingredient of Web Information Retrieval. A number of different LAR algorithms have been developed. Many of these are simply variations of each other, and it is unclear how they differ, and which algorithm is most appropriate in each case.

We study the behavior of the HITS algorithm on the class of product graphs, i.e. a generative model introduced by Azar et al. [6] using and extending the definitions of stability and similarity introduced by Borodin et al. [13, 14] to classes of random graphs. Stability considers the effect of small changes in the graph to the output of an LAR algorithm. Similarity studies how close the outputs of two algorithms are on the same graph. We then prove that, with high probability, under some restrictive assumptions, the HITS algorithm is stable on the class of product graphs and similar to the indegree heuristic that ranks pages according to their indegree. We show that our assumptions are general enough to capture graphs with expected degrees that follow a power law distribution as those observed in the real Web. We further analyze the correlation between indegree and HITS on a large sample of the Webgraph of about 136 M vertices. The experimental analysis reveals that our theoretical results obtained on specific class of graphs also holds in the real Web. The authority values computed by indegree and HITS are highly correlated. We conclude with a study on the conditions that guarantee similarity of HITS and indegree for the class of all possible graphs.

Structure of the thesis. This thesis is structured as follows. In the Chapter 2 we present the main topological properties of the Webgraph, and present the results of the experiments conducted on the WebBase crawl. In Chapter 3 we describe the external algorithms and give technical details on their implementations. In Chapter 4 we present the application of datastream algorithms to the measurements of the Webgraph. The measurements accomplished in order to mine the inner structure of the bow-tie are illustrated in Chapter 5. Finally in Chapter 7 the HITS and the PageRank algorithms are presented and a theoretical evaluation of the stability of HITS on the broad random class of the product graphs and moreover some conditions for similarity between HITS and indegree are introduced.
Part I

Static and Dynamical Properties of the Webgraphs
Chapter 2

Large Scale Properties of the Web

In this chapter, after a brief introduction on the basic graph terminology, we present the topological properties of the Web. In particular, for all the basic concepts, we compare the measurements we performed on the Webbase crawl [71], a Web sample with more then 1.5 billion edges, with previously collected samples.

2.1 Preliminaries

Basic graph terminology. A graph (also addressed as undirected graph) consists of a finite nonempty set of nodes (also called vertices) $V$ together with a collection of pairs of distinct nodes, called edges or arcs.

A directed graph or digraph consists of a finite nonempty set of nodes (vertices) $V$ together with a collection of pairs of ordered distinct nodes, called edges or arcs.

The degree of a vertex is the number of edges incident to it.

In a directed graph the indegree (outdegree) of a node is the number of incoming (outgoing) edges. For example, if we refer to the simple digraph shown in Figure 2.1, the indegree of vertex $D$ is 2 (it is linked from $B$ and $C$) while its outdegree is 0 (it has no outgoing edges).

A bipartite core is made of two sets of nodes; all the nodes in the first set (the fan set) point to each node of the second one (the center set). An example is shown in Figure 2.2: we have on the left side the set of the fan nodes (labelled $F_1$ and $F_2$), all of them pointing to all center nodes on the right side (labelled $C_1$, $C_2$ and $C_3$).

A walk is an alternating sequence of vertices and edges $v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n$ such that each edge is incident with the two nodes immediately preceding and following it. A walk is a path if all the nodes are distinct. A walk is closed if $v_0 = v_n$ and it is open otherwise. A closed walk is a cycle if all the vertices are distinct, except $v_0$ and $v_n$, and $n \geq 3$ ($n \geq 2$ in a digraph). An acyclic graph is one that contains no cycles. In a digraph the analogous definitions of directed walk and directed path hold if we replace edge with directed edge and we consider the
A *connected component* of an undirected graph $G$ is a subset of nodes $S$ such that for every pair of vertices $u, v \in S$, $u$ is reachable from $v$ (i.e. it exists the path starting from $u$ and ending in $v$). A graph is *connected* if, for every pair of vertices $u, v \in V$, $u$ is reachable from $v$. An acyclic connected graph is a *tree*, and an acyclic graph is a *forest* (i.e. is made of many trees).

A set of nodes $S$ is a *strongly connected component (SCC)* of a digraph if and only if, for every couple of nodes $A, B \in S$, there exists a directed path from $A$ to $B$ and from $B$ to $A$ and the set is maximal. The number of nodes of $S$ is the *size* of the SCC. For example, in the graph shown in Figure 2.3 there are 3 distinct strongly connected components: vertices $A_1$, $A_2$ and $A_3$ all can reach each other: they form a strongly connected component (SCC) of size 3. The same holds for vertices $B_1, B_2, B_3$ and $B_4$, that are a size 4 SCC, and for the vertices $C_1$ and $C_2$ (size 2 SCC).

A set of nodes $S$ is a *weakly connected component (WCC)* in a directed graph $G$, if and only if the set $S$ is a connected component of the undirected graph $\overline{G}$ that is obtained by removing the orientation of the edges in $G$.

For a subset of nodes $S \subseteq V$ we define the *subgraph induced* by $S$ to be the graph $G_S = (S, E_S)$, where $E_S$ is the set of edges between the nodes in $S$.

A *traversal* of a graph explores the edges of the graph until all vertices are visited. A traversal starts from a vertex, say $u$, explores the whole portion of the graph that is reachable from $u$, and then continues with a vertex not yet visited. A forest is naturally associated to a visit by considering all edges that lead to the discovery of a new vertex. A *Breadth First Search (BFS)* is a traversal which, when visiting a new vertex, stores the adjacent vertices not yet visited in a queue. That is, it explores the local neighborhood before going any deeper. A *Depth First Search (DFS)* is a traversal which, when visiting a new vertex, stores the adjacent vertices not yet
visited in a stack. That is, it always tries to go as deeply as possible.

Good introduction to graph theory are the classical work of [42] and the more recent book of [30].

**Power law distribution.** A discrete random variable $X$ follows a power law distribution if the probability of taking value $i$ is $P[X = i] \propto 1/i^{\gamma}$, for a constant $\gamma \geq 0$. The value $\gamma$ is the exponent of the power law. An interesting review on power law distributions can be found in [58].

**Scale-free networks.** If the degree distribution of the nodes in a network follows a power law, the ratio of very connected nodes to the number of nodes in the rest of the network remains constant as the network changes in size; these networks are also called *scale-free*.

**Crawlers, spiders and robots.** A *crawl* of the Web is a set of webpages together with their links. The programs that collect Web pages are usually referred to as *crawlers, spiders or robots*. Roughly, a crawler starts from an initial set of URLs $S$ and, at each iteration, it extracts one URL $u$ from $S$, visits it and adds to $S$ the (not yet visited) URLs that are linked by $u$. A crawler is one essential component of modern search engines. As the size of the Web grows the engineering of a crawler is an increasingly difficult task; a detailed discussion can be found in [23] and [11].

**External and semi-external memory algorithms.** Modern computer systems have a hierarchical memory architecture that comprises cpu registers, cache (several levels), main memory, buffers and secondary storage devices. Traditional analysis of algorithms assumes one level of memory, i.e. computations are performed in main memory. In many problems the amount of data to be processed is far too massive to fit main memory, and the analysis of algorithms under the assumption of a single level of memory can be meaningless, because the I/O performance is the main bottleneck. Usually the analysis considers only two memory levels, i.e. one fast (main memory) and one slow (secondary memory). With *external memory algorithms* we denote the ones that are explicitely designed to perform “well” in a hierarchical memory system. When we deal with graph algorithms, we call *semi-external* algorithms the ones that are allowed to store in main memory only a limited amount of bytes for each node. An overview of this topic can be found in [69, 70].
2.2 The statistical and topological properties of the Web graph

In this section we present the results of our experiments, conducted on a 200M nodes crawl collected from the WebBase project at Stanford [71] in 2001. The repository makes several crawls available to researchers. The sample we study in our work contains only link information, i.e. no information about URLs is available. We compare these results with the ones, presented in the literature, concerning other samples of the WebGraph. According to [41] in 2005 the size of the Web is around 11.5 billions webpages. An older study made by [28] showed that in July 2000 the size of the Web was 2.1 billion webpages, and this means that the WebBase sample, when it was collected, contained about one tenth of the Web. In the same study they observed a growth rate of 7 millions webpages per day, and this rate is coherent with the result of [41].

2.2.1 Indegree and outdegree

Since the very first analysis, the Webgraph has shown the ubiquitous presence of power law distributions, a typical signature of scale-free properties. [9] and [50] suggested that the indegree of the Webgraph follows a power law distribution. Later experiments by [18] on a crawl of 200M pages from 1999 by Altavista confirmed it as a basic property: the probability that the indegree of a vertex is $i$ is distributed as $P_{\text{indegree}(u)=i} \propto 1/i^{\gamma}$, for $\gamma \approx 2.1$. In [18] the outdegree of a vertex was also shown to be distributed according to a power law with exponent roughly equal to 2.7 with the exception of the initial segment of the distribution. The average number of incoming links observed in the several samples of the Webgraph is about equal to 7 times the number of vertices.

The indegree distribution, shown in Figure 2.4, follows a power law with $\gamma = 2.1$. This confirms the observations done on the crawl of 1997 from Alexa [50], the crawl of 1999 from Altavista [18] and the notredame.edu domain [9].

We note a bump between the values 1.000 and 10,000, that has also been observed by [18] and it is probably due to a huge clique created by a single spammer. Since our sample contains only structural information and not URLs, we can’t propose or deny possible explanations for this phenomenon.

In Figure 2.5 it is shown the outdegree distribution of the WebBase crawl. While the indegree distribution is fitted with a power law, the outdegree is not, even for the final segment of the distribution. A deviation from a power law for the initial segment of the distribution was already observed in the Altavista crawl [18]. A possible
2.2. THE STATISTICAL AND TOPOLOGICAL PROPERTIES OF THE WEB GRAPH

15

Figure 2.4: Indegree distribution of the WebBase crawl.

explanation of this phenomenon is that writing a scale-free series of hyperlinks is seriously limited by the patience of webmasters.

2.2.2 Pagerank

The Pagerank algorithm is at the basis of the ranking operated by the Google Web search engine. The idea behind link analysis ranking is to give higher rank to documents pointed by many Web pages. [15] extend this idea further by observing that links from pages of high quality should confer more authority. It is not only important which pages point to a page, but also what is the quality of the pages. They propose a weight propagation algorithm in which a page of high quality is a page pointed by many pages of high quality. We discuss the Pagerank algorithm in Chapter 3 where we detail our implementation.

The correlation between the distribution of Pagerank and indegree has recently been studied in a work of Pandurangan, Raghavan and Upfal [63]. They show by analyzing a sample of 100,000 pages of the brown.edu domain that Pagerank is distributed with a power law of exponent 2.1. This exactly matches the indegree distribution, but very surprisingly it is observed very little correlation between these quantities, i.e., pages with high indegree may have low Pagerank [63].

We computed the Pagerank distribution on the WebBase crawl. Here, we confirm the observation of Pandurangan et al. by showing this quantity distributed according to a power law with exponent $\gamma = 2.109$. We also computed the statistical correlation between Pagerank and indegree. We obtained a value of 0.3097, on a range
10 of variation in $[-1, 1]$ from negative to positive correlation. This result confirms the weak correlation between Pagerank and indegree values.

2.2.3 Bipartite cliques

A surprising number of specific topological structures such as bipartite cliques of relatively small size has been observed in [50] with the aim of tracing the emergence of hidden cyber-communities. A bipartite clique is interpreted as a core of such a community, formed by a set of fans, each one pointing to a set of centers/authorities, and a set of centers, each pointed by all the fans. Over 100,000 such communities have been recognized [50] on a snapshot of 200M taken by Alexa in 1997.

In Figure 2.7 is shown the distribution of the number of bipartite cliques $(i, j)$, with $i, j = 1, \ldots, 10$. The shape of the plot follows the one presented by Kumar et al. for the 200M crawl by Alexa. However, we detect a much larger number of bipartite cliques. For instance the number of cliques of size $(4, j)$ differs from the crawl from Alexa for more than one order of magnitude. A possible (and quite natural) explanation is that the number of cyber-communities has consistently increased from 1997 to 2001. We also recall that the longevity of cyber-communities’ websites is bigger as compared to other websites [50]. A second possible explanation is that our algorithm for finding disjoint bipartite cliques, which is detailed in Chapter 3, is more efficient than the one implemented in [50].

Figure 2.5: Outdegree distribution of the WebBase crawl.
2.2. THE STATISTICAL AND TOPOLOGICAL PROPERTIES OF THE WEB
GRAPH

100000000
10000000
1000000
100000
10000
1000
100
10
1

Figure 2.6: Pagerank distribution of the WebBase crawl.

<table>
<thead>
<tr>
<th></th>
<th>SCC</th>
<th>IN</th>
<th>OUT</th>
<th>TENDR.</th>
<th>DISC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altavista</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1999) [18]</td>
<td>28%</td>
<td>21%</td>
<td>21%</td>
<td>22%</td>
<td>9%</td>
</tr>
<tr>
<td>WebBase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2001)</td>
<td>33%</td>
<td>11%</td>
<td>39%</td>
<td>13%</td>
<td>4%</td>
</tr>
</tbody>
</table>

Table 2.1: Size of regions in both Altavista and WebBase crawl

2.2.4 Strongly connected components

Broder et al. [18] identified a very large strongly connected component of about 28% of the entire crawl, and showed a picture of the whole Web as divided into five distinct regions: SCC, IN, OUT, TENDRILS and DISCONNECTED. The SCC set is the set of all the nodes in the single largest strongly connected component; in the IN (OUT) region we find all the nodes that can reach the SCC set (are reached from the SCC). TENDRILS are either nodes that leave the IN without entering the SCC or enter the OUT without leaving the SCC. In Table 2.1 we report the relative size of the 5 regions. We can still observe in the WebBase crawl a large SCC, however the biggest component is the OUT region, and both IN and TENDRILS have a reduced relative size if compared to the Altavista crawl. We also observe a huge difference between the size of the largest SCC, which consists of about 48 millions nodes, and the size of the second largest SCC that is less than 10 thousands nodes.
In Figure 2.7 it is shown the global SCC distribution of the Webbase sample and of its different regions (except the SCC region, that is a single SCC). All distributions follow a power law whose exponent is 2.07, very close to the value observed for both the indegree and the Pagerank distributions.

2.3 Conclusions

In this chapter we detailed the known statistical properties of the Web graph. We examined the algorithmic aspects of this research, and we detailed several algorithms to measure massive Webgraphs. Probably the most important aspect not considered is that, despite its dynamic nature, the Webgraph has been studied so far from a static point of view: snapshots of it have been analyzed but it is still missing the projection of its properties against a temporal axis. A temporal analysis of the properties of a particular sample of the Web, derived by the hyperlinked pages of Wikipedia, an online encyclopedia, is in Chapter 6.
Figure 2.8: SCC distribution of the Web Base crawl.
Chapter 3

Secondary Memory Algorithms for Measuring Webgraphs

In this chapter we detail a complete methodology for handling massive Webgraphs. As a first step we need to identify the distinctive components of the Webgraph. For this we need to be able to perform traversals of the Webgraph. The traditional graph algorithms are designed to work in main memory, so they present a drastic slump in performance as soon as the amount of data exceeds the available main memory space. During a graph traversal, the nodes accesses happen in a hardly expectable way and, therefore, data memory references stretch on the entire address space. The OS isn’t able to identify a working set smaller than the whole data set, as it can not exploit any locality properties induced by the traversals. The link structure of the Web graph takes several gigabytes of disk space, making it prohibitive to use traditional graph algorithms. Therefore, we examine alternative approaches that use external memory. We implement semi-external algorithms, that use only a small constant amount of memory for each node of the graph, as opposed to fully-external ones, that use an amount of main memory that is independent of the graph size.

We implemented the following algorithms.

- A semi-external Pagerank.
- A semi-external algorithm for computing bipartite cores.
- A semi-external graph traversal for determining vertex reachability using only 2 bits per node.
- A semi-external Breadth First Search that computes blocks of reachable nodes and splits them up in layers. In a second step, these layers are sorted to produce the standard traversal result.
- A semi-external Depth First Search (DFS) that needs 12 bytes plus one bit
CHAPTER 3. SECONDARY MEMORY ALGORITHMS FOR MEASURING WEBGRAPHS

for each node in the graph. This traversal has been developed following the approach suggested by [65].

- A semi-external algorithm for computing all SCCs of the graph, based on the semi-external DFS.

- An algorithm for computing the largest SCC of the Webgraph. The algorithm adopts a heuristic approach that exploits structural properties of the Webgraph to compute the biggest SCC, using a simple reachability algorithm. As a result of the algorithm we obtain the bow-tie regions of the Webgraph, and we are able to compute all the remaining SCCs of the graph efficiently using the semi-external DFS algorithm.

Remarkable performance improvements are achieved using the semi-external algorithms - vertex reachability and DFS - and exploiting the Webgraph structure. All the algorithm listed above are publicly available, together with other routines, in a software library [33]. A detailed description of some of these algorithms and of their performance analysis appears in [51].

3.1 Data representation and multifiles

The first problem when dealing with massive data in secondary memory is the size limit of a single file\(^1\). All our routines operate on a multifile, that is, we store a single graph into more than one file. More precisely, we use one .info file, that contains information about the nodes, one or more .prec files that contain information about the predecessors of each node, and one or more .succ files that contain information

\(^1\)This limit can be changed but we preferred, for portability reasons, to use a multifile format.
about the successors of each node. We refer to all these files related to a single graph as the .ips multifile. Figure 3.1 shows a simple graph together with its representation, details can be found in [33].

3.2 Traversal with two bits for each node

We now describe an algorithm for computing all the vertices reachable by a single node, or by a set of nodes. The algorithm does not use a standard graph traversal algorithm such as BFS or DFS. Instead it operates on the principle that the order in which the vertices are visited is not important. For each vertex \( u \) in the graph, it maintains only two bits of information:

1. The first bit \( \text{reached}[u] \) is true if \( u \) has already been reached, and false otherwise.
2. The second bit \( \text{completed}[u] \) is true if the adjacency list of \( u \) has been visited, that is, all adjacent vertices are marked as reached.

At the beginning, no vertex is completed, and only the nodes in the start set are reached. The files of the .ips multifile representation of the graph are sequentially scanned, and the nodes, together with their adjacency lists, are brought to main memory one by one. When we consider a node \( u \) that is reached but not completed, then all its successors are marked as reached. At this point the node \( u \) is marked as completed. If the node \( u \) is not reached, no processing is performed and we just move on to the next node. After a number of scans over the graph, all nodes that are \( \text{reached} \) are also \( \text{completed} \). At this point the graph traversal is completed.

To reduce the number of scans over the graph, we scan the files by loading in main memory a whole block of nodes (with their adjacency lists). The algorithm makes multiple passes over this block of nodes so as to extend as much as possible the traversal of the graph, that is, until all the nodes in this block that are reached and have their adjacency list stored in main memory are also completed.

The reachability traversal is a powerful and efficient tool that enables us to perform many different measurements on the Webgraph.

3.3 Semi-external Breadth First Search

The semi-external memory BFS is performed by executing a layered graph traversal. The BFS algorithm discovers vertices of the graph at increasing distance from the root. When at layer \( i \), the algorithm performs a complete scan of the graph so as to find all successors of the vertices at layer \( i \) that have not been reached so far (This information is stored into a bit vector available in main memory). These vertices will form the \((i+1)\)-th layer of the graph. We also label the vertices according to the layer
they belong to, in order to produce a BFS numbering of the graph. The efficiency of this procedure for the Web graph relies on the fact that most of the vertices of the graph can be found within few hops from the CORE.

### 3.4 Semi-external depth first search

Unfortunately, so far there are no efficient external-memory algorithms to compute DFS trees for general directed graphs. We therefore apply a recently proposed heuristic for semi-external DFS [65]. It maintains a tentative forest which is updated by bringing in from external memory a set of non-tree edges (edges that are not part of the current DFS tentative forest) so as to reduce the number of cross edges (edges between two vertices that are not in ancestor-descendant relation in the tentative forest).

The basic idea must be complemented with several implementation hacks in order to lead to a good algorithm. We refer to [65] for further details on the algorithm.

In our implementation, the algorithm maintains at most three integer arrays and three boolean arrays of size $N$, where $N$ is the number of nodes in the graph. With four bytes per integer and one bit for each boolean, this means that the program has an internal memory requirement of $(12 + \frac{3}{8})N$ bytes. The standard DFS needs to store $16dN$ bytes, where $d$ is the average degree. This can be reduced if one does not store both endpoints for every edge. Still, under memory limitations, standard DFS starts paging at a point when the semi-external approach still performs efficiently.

### 3.5 Computation of the SCCs

It is well known [27] that the computation of all SCCs of a graph can be reduced to the execution of two DFS visits, one on the graph, and one on its transpose [68].

Due to memory limitations, even a semi-external DFS is prohibitive for Web graphs of the size we consider. We tackle this problem by removing the CORE of the Web graph before proceeding with the computation of all SCCs. We can then apply the semi-external DFS.

The question is how to identify the CORE efficiently. Using the graph traversal algorithm described in section 3.2 there is a simple way for determining the SCC that contains a given node $u$. Compute the set of vertices reachable from a forward and a backward visit starting from $u$, and then return the intersection of the two. This simple method suggests a heuristic strategy for determining the largest SCC of a graph with a bow tie structure: i) select uniformly at random a starting set of nodes $S$; ii) for each node $u$ in $S$ compute the SCC that contains $u$ and return the largest one.

For a graph that includes a CORE of about a quarter of all pages, using a starting set of just 20 nodes, the probability of not finding the CORE is only $(3/4)^{20} \approx 0.3\%$. 
3.6 Computation of the Bow-Tie regions

The computation of the largest SCC returns the CORE of the bow-tie graph. Starting from the CORE we can now compute the remaining components of the bow-tie structure.

**The IN Component:** The nodes of the IN component can be found by performing a backward traversal, using the CORE as the starting set. The nodes returned are the union of the CORE and IN. The IN component can be obtained by deleting the CORE nodes from this set. This operation is performed with a simple XOR logic operation between boolean vectors.

**The OUT Component:** The nodes of the OUT component can be found by performing a forward traversal, using the CORE as the start set. The nodes returned are the union of the CORE and OUT. The OUT component can be obtained by deleting the CORE nodes from this set, using an XOR operation as before.

**TENDRILS and TUBES:** The TENDRILS are sets of nodes, not belonging to the CORE, that are either reachable by nodes in IN, or that can reach nodes in OUT. The TUBES are subsets of the TENDRILS that are reachable by nodes in IN, and that can reach nodes in OUT. They form paths that lead from IN to OUT without passing through the CORE. The computation of these two sets is accomplished in three steps:

1. In the first step, we identify the set TENDRILS\_IN which consists of all the nodes that are reachable by IN and belong neither to the CORE, nor to the OUT set. In order to determine TENDRILS\_IN, we perform a forward visit from IN, where all the nodes in the CORE are marked as completed, and the nodes in IN are marked as reached. From the set computed in this way, we discard the nodes that belong to IN or OUT. The OUT nodes are reachable through the TUBES, but they should not be considered in the TENDRILS\_IN set.

2. In the second step, symmetric to the first one, we identify the set TENDRILS\_OUT which consists of all the nodes that point to the OUT and don’t belong neither to the CORE nor to IN.

3. In the last step, we compute the TENDRILS and TUBES:

\[
\text{TENDRILS} = \text{TENDRILS} \_\text{IN} \cup \text{TENDRILS} \_\text{OUT} \\
\text{TUBES} = \text{TENDRILS} \_\text{IN} \cap \text{TENDRILS} \_\text{OUT}
\]

**DISC:** The DISC consists of all the remaining nodes. These are sets of nodes that are not attached in any way to the central bow-tie structure.
3.7 Disjoint bipartite cliques

[50] propose a greedy algorithm that detects the largest possible number of disjoint bipartite cliques \((i, j)\), with \(i\) being the fan vertices on the left side and \(j\) being the center vertices on the right side, and \(i, j \leq 10\). Note that the problem of enumerating disjoint bipartite cliques is NP-complete.

This algorithm is composed of a pruning phase that consistently reduces the size of the graph in order to store it into main memory. A second phase enumerates all bipartite cliques of the graph. A final phase selects a set of bipartite cliques that form the solution. Every time a new clique is selected, all intersecting cliques are discarded. Two cliques are intersecting if they have a common fan or a common center. A vertex can then appear as a fan in a first clique and as a center in a second clique.

In the following, we describe our semi-external heuristic algorithm for computing disjoint bipartite cliques. The algorithm searches bipartite cliques of a specific size \((i, j)\).

Two \(n\)-bit arrays \(Fan\) and \(Center\), stored into main memory, indicate with \(Fan(v) = 1\) and \(Center(v) = 1\) whether fan \(v\) or center \(v\) has been removed from the graph. We denote by \(I(v)\) and \(O(v)\) the list of predecessors and successors of vertex \(v\). Furthermore, let \(\bar{I}(v)\) be the set of predecessors of vertex \(v\) with \(Fan(\cdot) = 0\), and let \(\bar{O}(v)\) the set of successors of vertex \(v\) with \(Center(\cdot) = 0\).

The core algorithm is preceded by two pruning phases. The first phase removes vertices of high degree as prescribed in [50] since the objective is to detect cores of hidden communities. In a second phase, we remove vertices that cannot be selected as fans or centers of an \((i, j)\) clique.

We outline in the following the idea underlying the algorithm, see [51] for a detailed description. Consider a fan vertex \(v\) with at least \(j\) successors with \(Center(\cdot) = 0\), and enumerate all size \(j\) subsets of \(\bar{O}(v)\). Let \(S\) be one such subset of \(j\) vertices. If \(\bigcap_{u \in S} I(u)\) \(\geq i\) then we have detected an \((i, j)\) clique. We remove the fan and the center vertices of this clique from the graph. If the graph is not entirely stored into main memory, the algorithm has to access the disk for every retrieval of the list of predecessors of a vertex of \(O(v)\). Once the exploration of a vertex has been completed, the algorithm moves to consider another fan vertex.

In our semi-external implementation, the graph is stored on secondary memory in a number of blocks. Every block \(b, b = 1, \ldots, \lceil N/B \rceil\), contains the list of successors and the list of predecessors of \(B\) vertices of the graph. Denote by \(b(v)\) the block containing vertex \(v\), and by \(B(b)\) the vertices of block \(b\). We start by analyzing the fan vertices from the first block and proceed until the last block. The block currently under examination is moved to main memory. Once the last block has been examined, the exploration continues from the first block.

We start the analysis of a vertex \(v\) when block \(b(v)\) is moved to main memory for the first time. We start considering all subsets \(S\) of \(\bar{O}(v)\) formed by vertices of block
b(v). However, we also have to consider those subsets of \( O(v) \) containing vertices of other blocks, for which the list of predecessors is not available in main memory. For this purpose, consider the next block \( b' \) that will be examined that contains a vertex of \( O(v) \). We buffer into an auxiliary file \( A(b') \), associated with block \( b' \), both \( O(v) \) and the lists of predecessors of the vertices of \( O(v) \cap B(b) \). When a block \( b \) is moved to main memory, we first seek to continue the exploration from the vertices of \( A(b) \).

If the exploration of a vertex \( v \) in \( A(b) \) cannot be completed within block \( b \), the list of predecessors of the vertices of \( O(v) \) in blocks from \( b(v) \) to block \( b \) are stored into the auxiliary file of the next block \( b' \) containing a vertex of \( O(v) \). We then move to analyze the vertices \( B(b) \) of the block. We keep on doing the process till all fan and center vertices have been removed from the graph. It is rather simple to see that every block is moved to main memory at most twice. The auxiliary files are also buffered in our implementation.

### 3.8 Pagerank

The computation of Pagerank can be expressed in matrix notation as follows. Let \( N \) be the number of vertices of the graph and let \( n(j) \) be the outdegree of vertex \( j \). Denote by \( M \) the square matrix whose entry \( M_{ij} \) has value \( 1/n(j) \) if there is a link from vertex \( j \) to vertex \( i \). Denote by \( \left[ \frac{1}{N} \right]_{N \times N} \) the square matrix of size \( N \times N \) with entries \( \frac{1}{N} \). Vector \( \text{Rank} \) stores the value of Pagerank computed for the \( N \) vertices. A matrix \( M' \) is then derived by adding transition edges of probability \( (1 - c)/N \) between every pair of nodes to include the possibility of jumping to a random vertex of the graph:

\[
M' = cM + (1 - c) \times \left[ \frac{1}{N} \right]_{N \times N}
\]

A single iteration of the Pagerank algorithm is

\[
M' \times \text{Rank} = cM \times \text{Rank} + (1 - c) \times \left[ \frac{1}{N} \right]_{N \times 1}
\]

We implement the external memory algorithm proposed by [43]. The algorithm uses a list of successors \( \text{Links} \), and two arrays \( \text{Source} \) and \( \text{Dest} \) that store the vector \( \text{Rank} \) at iteration \( i \) and \( i + 1 \). The computation proceeds until either the error \( r = ||\text{Source} - \text{Dest}|| \) drops below a fixed value \( \tau \) or the number of iterations exceeds a prescribed value.

Arrays \( \text{Source} \) and \( \text{Dest} \) are partitioned and stored into \( \beta = \lceil N/B \rceil \) blocks, each holding the information on \( B \) vertices. \( \text{Links} \) is also partitioned into \( \beta \) blocks, where \( \text{Links}_\ell, \ell = 0, ..., \beta - 1 \), contains for every vertex of the graph only those
successors directed to vertices in block \( \ell \), i.e. in the range \([\ell B, (\ell + 1)B - 1]\). We bring to main memory one block of \( Dest \) per time. Say we have the \( ith \) block of \( Dest \) in main memory. To compute the new Pagerank values for all the nodes of the \( ith \) block we read, in a streaming fashion, both arrays \( Source \) and \( Links_i \). From array \( Source \) we read previous Pagerank values, while from \( Links_i \) we have the list of successors (and the outdegree) for each node of the graph to vertices of block \( i \), and these are, from the above Pagerank formula, exactly all the information required.

The main memory occupation is limited to one float for each node in the block, and, in our experiments, 256MB allowed us to keep the whole \( Dest \) in memory for a 50M vertices graph. Only a small buffer area is required to store \( Source \) and \( Links \), since they are read in a streaming fashion.

### 3.9 Conclusions

In this Chapter we presented the algorithms we developed and implemented to measure the properties of large scale graphs. We want to extend all the data structures in order to explicitly consider the time factor. A first step towards this direction has been presented in [47], where each edge is labelled with the dates of its first and last appearance in the Web. This new data, of course, pose several challenges; among them we cite (i) the problem of efficiently representing dynamic graphs in secondary memory, (ii) whether it is possible to adapt the Webgraph compression techniques to it and (iii) if it is possible to design algorithms able to deal explicitly with the time labels without the need of generating multiple snapshots from it.
Chapter 4

Data Stream Computation for Measuring Webgraphs

4.1 Data Stream Algorithms

Data stream algorithms aim to maintain the underlying information of a stream of data, using small memory space. The data is processed on the fly, as it is generated, or it can also be read from second memory devices. Typical applications of data stream algorithms are originated from massive datasets such as network traffic measurements, telephone call records, biological datasets and atmospheric observations. In these applications is unnecessary or impractical to read data multiple times. In many cases, the data is not even stored. This Chapter focuses on a "new" natural application for data streams. We are interested in using data stream algorithms for monitoring statistical and topological properties of large graphs such as the webgraph. By webgraph we mean the directed graph generated from the link structure of webpages: each webpage is a node and each hyperlink is an arc in this graph. Likewise, sub-graphs can be generated from specific webpage collections such as blogs, online encyclopedias, online bookstores, the collection of webpages within a domain, and many others. The graph read in a streaming fashion considers each edge as an item and the stream is not required to be structured.

The main advantage of using data streams instead of exact algorithms is that the space used for managing and mining the stream is small, without resorting to external memory algorithms. Furthermore, results can be output anytime during the stream processing, not requiring that the whole data input be processed in advance. On the other hand, data stream algorithms do not provide exact values, but an approximation that depends on the precision required and the amount of resources we are willing to invest.

Several theoretical results have been proposed in this new research field, some of them have not yet been implemented and experimented, some of them are not
practical. In this Chapter we observe how a data stream algorithm behaves in practice for computing the indegree rarity distribution of a graph over the arc arrivals. More specifically, we maintain the distribution of the number of nodes that has a given indegree over the total number of different nodes seen in the stream so far. We use the algorithm proposed by Datar and Muthukrishnan [29] and show experimentally that the results are very close to the optima even when a low precision is requested. The original algorithm proposes the use of min-wise hash functions, whereas we use universal hashing [29]. This decision is due to the fact that computing min-wise hashing consumes about two orders of magnitude more time than universal hashing without providing better results in practice for the graphs we have tested.

When considering a specific structure in the data stream, other properties can be computed. For example, reading the stream in an adjacency list fashion, the same rarity algorithm can be used for estimating the density of minors such as small bipartite cliques.

The indegree of webpages is an important measure of their popularity. The experimental observation of the indegree distribution has been the subject of seminal works aimed to characterize the structure of the webgraph [9, 18]. This study has also revealed a surprising number of dense subgraphs, specifically bipartite cliques, of moderately small size [50], considered as cores of hidden web communities.

In the next section we present the $\alpha$-rarity algorithm of Datar and Muthukrishnan [29]. Section 3 describes the adaptations of the $\alpha$-rarity algorithm for computing indegree rarity distribution, as well as for computing minors of small size. In section 4 we present experimental results for rarity of indegree bipartite cliques of size three $(k3,3)$ and for the minors $k(1,3)$ and $k(2,3)$. We generalize the set of all minors mentioned above using the term $k(i,3)$, where $i$ denotes the number of nodes in the graph that points to each node of a triple (set of three nodes). Comparison with the results of an optimal computation shows excellent practical results of our implementations. Section 4 also described the optimization of a an implementation of min-wise hash functions [73].

4.2 Estimating rarity over data stream windows

We use the $\alpha$–rare algorithm of Datar and Muthukrishnan [29] for driving our experiments. Consider a stream of items $a_i$ generated in a universe $U=[1,..,n]$. A stream is a set of $m$ elements $a_1,a_2,...,a_m$ such that $a_i \in U$. An item $i$ is called $\alpha$–rare if it appears exactly $\alpha$ times in the stream. Let’s call $\#\alpha$–rare the number of elements that appear exactly $\alpha$ times in the stream. Likewise, $\#\text{distinct}$ denotes de number of distinct items in the stream. The $\alpha$-rarity $\rho_\alpha$ is defined as the ratio $\rho_\alpha = \frac{\#\alpha\text{-rare}}{\#\text{distinct}}$. In other words, the $\alpha$-rarity of a stream is the measure of number of items that repeat exactly $\alpha$ times in the stream.

The algorithm proposed by Datar and Muthukrishnan [29] for computing the $\alpha$-
4.3. COMPUTING THE RARITY DISTRIBUTION OF INDEGREE AND
BIPARTITE CLIQUES

rarity of a stream uses min-wise hash functions. Min-wise independent permutation families are defined in [17]. Let \( S_n \) be the set of all permutations \( \pi \) of \([1,...,n]\).

A permutation family \( F \) (subset of all permutations over \([1..n]\)) is exactly \textit{min-wise independent} if for any subset \( X \) of \([1..n]\), and any \( x \in X \), when \( \pi \) is chosen at random from \( F \) we have \( \Pr\{ \min\{\pi(X)\} = \pi(x) \} = \frac{1}{|X|} \). In other words, it is required that all elements of a given set \( X \) have an equal chance to become the minimum element of the image of \( X \) under \( \pi \).

The referred algorithm uses only \( O(\log N + \log u) \) space, and \( O(\log \log N) \) per item processing time. It estimates \( \rho \) by \( \hat{\rho} \in [1 \pm \epsilon]\rho + \epsilon\rho \) for a given fraction \( \epsilon \), with high probability. The algorithm uses \( h = 2e^{-3}p^{-1}\log \tau^{-1} \) hash functions and two \(|h|\)-vectors, \( \min \) and \( \text{count} \), in main memory. Each position \( i \) of the vector \( \min \) contains the minimum value found so far by the min-wise hash \( h_i \), whereas \( \text{count} \) maintains, for each position \( i \), the number of times that the current minimum min-wise value was found. For each value of \( \alpha \), \( \hat{\rho} \) is computed as the ration between the number of counters that have exactly value \( \alpha \) and \( h \).

A slightly different algorithm is proposed for computing the \( \alpha \)-rarity of a windowed stream. E.g., Considering a fix window size equal to \( \tilde{W} \), the algorithm maintains the \( \alpha \)-rarity of the last \( \tilde{W} \) items seen in the stream. In this case, to maintain the current minimum for each hash function is a bit more complicated. Instead of the \(|h|\)-vector \( \min \), a linked list \( \text{LM} \) is used. For each min-wise hash function, all non dominated minima are maintained, also with indication of the number and the time of the occurrences of that minimum. For non dominated minima we mean the min-wise hash values that are larger than the current minimum, but were generated more recently and belong to the current window. The number of times that each minimum was found is stored in another linked list \( \text{LT} \), instead of using the \(|h|\)-vector \( \text{count} \). Each element in this list contain information about the time the correspondent minimum was found. For each new item processed in the stream, the lists are updated twice. First the items no longer in the lists are removed (checking \( \text{LT} \) info) and next, the lists are update with the new element.

### 4.3 Computing the rarity distribution of indegree and bipartite cliques

In this section we describe how the not-windowed algorithm described in the previous section is adapted to compute the \( \alpha \)-rarity algorithm for computing the indegree and \( k(i, 3) \) rarity distributions of a graph.

Considering an arbitrary scan of a digraph \( G=(V,E) \), where \( V \) is the set of nodes and \( E \) is the set of edges of this graph. The items of the stream, in this case, are the list of edges. The \( \alpha \)-rarity of the stream can be understood as the percentage of nodes that has indegree \( \alpha \). With the underlying data stored for estimating \( \alpha \), we can compute the \( \alpha \)-rarity for any \( \alpha_i < \alpha \). So, computing the rarity distribution for an \( \alpha \)
large enough, we obtain the rarity indegree distribution of the graph considering any value $\alpha$. The rarity distribution can be computed for a complete stream, or for the window of the last $W$ items seen in the stream.

When considering some structure in the stream, other properties can be computed. For example, reading the stream in an adjacency list fashion, the same rarity algorithm can be used for approximating the density of minors, such as small bipartite cliques. Such kind of structured data stream can be found naturally on some applications. For example, during a crawling process, each current fetched page is parsed and all outgoing links of this page identified. It is exactly this kind of order that we are considering here.

Now we describe the adaptation of the $\alpha$-rarity algorithm for computing the $k_{i,3}$ rarity distribution on a graph. The digraph $G$ is read in a streaming fashion, e.g., all outgoing links of a node $i \in V$ are read in sequence. The lists of outgoing edges are not required to be in any specific order, as well as the edges internal to each list. So, for each node $u$, for each outgoing edge $(\bar{u}, \bar{a}) \in OUT(u)$, triples are calculated considering node $a$ and all combinations two by two of the head-nodes of the edges seen so far in $OUT(u)$. E.g, triples $(a, b, c)$ are calculated for nodes $b, c \in OUT(u)$ considering edges $(\bar{u}, \bar{b})$ and $(\bar{u}, \bar{c})$ previously located in $OUT(u)$ than $(\bar{u}, \bar{a})$. So, the overall number of triples ($T$) of the graph is the sum of the combination three by three of head-nodes of the outgoing list of each node $u \in V$, e.g.,

$$T = \sum_{i=1}^{d_i} \frac{d_i \cdot (d_i - 1) \cdot (d_i - 2)}{6}$$

where $d_i = |OUT(i)|$ is the outdegree of the node $i$.

We require to store in main memory the whole outgoing adjacency list of the current node.

### 4.4 Experimental Results

In this section we describe the experimental results we performed using the $\alpha$-rarity algorithm. The algorithms were coded in g++ version 3.3.2. The experiments were conducted in a Intel Pentium IV, with 1GB RAM, running Mandrake 9.0.

Due to the excessive computational time spent by min-wise hash functions, we use universal hash functions instead. We used the hash function ($hash31$) and the random number generator ($prng\_int$) from the online available codes from the MassDAL group of Rutgers (http://www.cs.rutgers.edu/muthu/massdal-code-index.html).

The implementation of all algorithms presented in this section, as well as the optimized version of min-wise hash functions, are available by e-mail request.

We start describing the optimization applied to the online available implementation of min-wise functions. Next, we describe the datasets we used and afterwards present results for the indegree distribution for the entire graph view and for the windowed case. We conclude our experiments with some results for the $\alpha$-rarity algorithm applied for the $k_{i,3}$ case.
4.4. EXPERIMENTAL RESULTS

4.4.1 Optimization of Min-wise hashing

We use an optimized version of Jerry Zhao's implementation [73] of an approximate restricted min-wise independent permutation family proposed by Alon et al. [4]. In practice, one can allow certain relaxations. One of the relaxed definition is approximate restricted min-wise permutation family. The implementation uses a linear feedback shift register with a irreducible polynomial as feedback rule as described in [4] to generate hash values. A certain number of those hashing functions is then used to compute the permutation value [17]. The time to compute a permutation using this implementation is $O(n(\log \log n + \log k + \log \frac{1}{\epsilon}))$ using a $\frac{1}{\epsilon}$ away approximate min-wise permutation for $[0..n]$ restricted by $2^k$-wise independence. The space cost of the original implementation is $O(\log \log n + \log k + \log \frac{1}{\epsilon})$. Our modification is to store intermediate results during the calculation of the register values instead of starting over from the beginning every time the hash function is called. For this purpose we memorize blocks of consequent values long enough to be able to resume calculation from this point i.e. of the length of the feedback rule. Thus the space cost is increased but the software remains usable on a normally equipped system. The calculation time however – together with slight changes to avoid expensive functions – was reduced by a factor of orders of magnitude.

4.4.2 Datasets

We conducted our experiments on streams of Wikipedia graphs. A graph of this type is generated from the link structure of the online and free-content encyclopedia Wikipedia (www.wikipedia.org). It started in January 15, 2001 with a few English articles. Four years later, Wikipedia has more than 1 million articles, available in more than 100 languages: The English version is the largest one, with about half million articles. Following the definition of a webgraph, each article is a node, and each hyperlink is a link in the graph. One graph is extracted for each language.

There are a few reasons why we are using wikipedia graphs for tests:

- **Independency of external links**: wikipedia articles link mainly to articles on the same dataset.

- **Variety of graph sizes**: it can be collected one graph by language, and the graph dimensions vary from a few hundred pages up to half million pages.

- **Generation on time**: wikipedia provides time information associated with nodes. Moreover, it provides old information: time information of data of creation and dates of modification for each page on the dataset.

- **Available on dumps**: it can be dumped as mysql tables, instead of been crawled. New dumps are provided almost weekly.
We generate streams of edges of the wikipedia graphs following their generation on time. In our experiments we use the graphs wikiEN, wikiDE, wikiFR, wikiIT, wikiPT from the datasets extracted from the English, German, French, Italian and Portuguese languages, respectively. The graphs were obtained from an old dump of July 2004. Due to space restrictions, we limited the presentation of experimental results in this extended abstract to the wikiEN and wikiPT graphs. Some comments are added about the experimental results on the other three graphs. Graph wikiPT contains 8,131 nodes and 48,168 edges, while graph wikiEN is two orders of magnitude larger containing 286,754 nodes and 4,065,530 edges.

4.4.3 Rarity Indegree Distribution

This subsection describes the results obtained using the $\alpha$-rarity algorithm for the entire stream (unbounded) and the windowed cases. Figure 4.1 presents results for the rarity for the unbounded case, using 1000 hash functions. The lines are plot for a logarithmic number of indegree values. The plot omits results for indegree higher than 63 for the sake of clarity of the figure, but a complete plot would present additional lines on the bottom of the figure, appearing on increasing order of the number of edges processed.

For a good approximation, a larger number of hash functions are required. For example, if we set $\epsilon = p = \tau = 0.1$, 10,000 hash function are required. For $\epsilon = p = \tau = 0.2$, just 437 hash functions are needed. But we observed, that even with a small number of hash functions, the results are close to the optimas. Figure 4.2 presents results when using only 100 hash functions.

For the windowed case, similar quality of results were found. Figure 4.3 presents results for windows of 100,000 items, estimated using 100 hash functions.

We also found good approximation when using the $\alpha$-rarity algorithm for computing the rarity distribution of $k(i,3)$ on the graph. Results for $i=1,2,3$ are plot in Figure 4.4. The plot is in log scale to be able to visualize all three distributions clearly on the same plot. Usually the number of $k(1,3) \gg k(2,3) \gg k(3,3)$. The difference between this values decrease with the increase of $i$. Observe, for example, the precision on results between the estimated and exact computation of $k(1,3)$ and $k(2,3)$. Since $k(1,3)$ is found many more times than $k(2,3)$, the results are more accurate. For values of $i > 4$ we did not plot for the sake of clarity of the plot, but the precision on the results decrease with the increase of $i$. As expected, we have less precision for computing $\hat{\rho}$ of $\alpha$-rare elements that occur less frequently.

We finalize the experimental results section with a time analysis as a function of the number of hash functions used and the number of elements hashed. Table 4.1 presents the average time spent for computing windows of 1,000, 10,000, 100,000 and 1,000,000 items ($\bar{W}$). For each value of $\bar{W}$, the use of 100 and 1000 universal hash functions are considered. The first column ($\bar{W}$) indicates the number of elements.
4.4. EXPERIMENTAL RESULTS

Figure 4.1: Estimated and exact indegree rarity distributions computed for edges arrivals of graph wikiEN. The estimation makes use of 1000 universal hashing functions. Values are presented to $\alpha$ up to 63, presented as $\log_2$ plot. This plot presents the percentage of nodes with a given indegree (y-basis) considering the amount of edges processed so far (x-basis). Results are plot every 100,000 items processed.
CHAPTER 4. DATA STREAM COMPUTATION FOR MEASURING WEBGRAPHS

Figure 4.2: Estimated and exact indegree rarity distribution computed for edges arrivals of graph \texttt{wikiEN}. The estimation makes use of 100 universal hashing functions. Values are presented to $\alpha$ up to 63, presented as $\log_2$ plot. This plot presents the percentage of nodes with a given indegree (y-basis) considering the amount of edges processed so far (x-basis). Results are plot every 100,000 items processed.

Times presented for the indegree rarity distribution are very small, even when considering the larger windows. For example, using a thousand hash functions just half second is spent on average for processing 1 billion items. Another observation is that the time grows linearly with the increase of the number of hash functions used and with the number of items considered in each window considered. For the windowed case, much higher times were found. That happens because each update on the dynamic lists take $O(L)$, where $L$ represents the size of the list. Again the times increase linearly with increase of the window size and the number of hash functions used. For the $k(i,3)$ computation, much more time was spent. The bottleneck of this application is that all triples of nodes within and adjacency list have to be computed. This enumeration takes long time, since often nodes of webgraphs have very large outdegree. For example, for the \texttt{wikiPT} graph used for the experiments and average of 624 triples are composed for each edge (his head node is considered for composing triples). Again the times increase linearly with the increase of $W$ and $\#h$. The graph \texttt{wikiPT} was used in this experiments since it is not possible to compute
4.5 Conclusion

In this Chapter we use in practice data stream algorithms for computing statistical and topological properties of large graphs. We presented experimental results for the $\alpha$-rarity algorithm applied on webgraphs for computing the rarity distribution of indegree and $k(i, 3)$ and obtained very good approximations. For the windowed case, applied for the indegree distribution, we observed again good approximation in a reasonable time. For the $k(i, 3)$ estimation we obtained good approximations, but spending long time. That happens because, in this case, all triples obtained are hashed by the $#h$ hash functions. For the wikiPT graph, we observe a total of 624 triples generated for each edge processed.

We conclude that using universal hashing by this algorithm speed up a lot the codes, maintaining good approximations.
CHAPTER 4. DATA STREAM COMPUTATION FOR MEASURING WEBGRAPHS

Figure 4.4: Plot in log scale of the estimated and exact $k_i, 3$ rarity distribution, for $i=1, 2$ and 3, computed for edges arrivals of graph wikiPT. The estimation makes use of 1000 universal hash functions. This plot presents the percentage of triples pointed by exactly $i$ nodes (y-basis) considering the amount of triples seen so far (x-basis). The triples are computed accordingly with the edges arrivals. Results are plot every 10,000 triples processed.

As further work, we would like to test other algorithms that estimates interesting statistical and topological properties of webgraphs. Moreover, dynamic aspects of webgraphs also could be explored, as edges being inserted and removed over time. The $\alpha$-rarity algorithm does not have solution for deletions. But a recent publication of Cormode, Muthukrishnan and Rozenbaum [39] presents an algorithm that maintain results considering also deletions. Likewise, a sampling algorithm was presented by Frahling, Indyk and Sohler [40], also for maintaining distributions under insertions and deletions. Another important issue is on computing minors. The only reference on data streams for computing minors is by Yossef, Kumar and Sivakumar [72], but the bounds are not encouraging. Algorithms for computing minors as triangles and small cliques, with implementable bounds, would be a great contribution of that stream algorithms for our purposes.
Table 4.1: Average computation times in seconds for streams of fix size from the wikiEN (results for the indegree rarity distribution) and wikiPT (results for the \( k(i,3) \) rarity distribution) graphs. For each one of the three applications (indegree, windowed indegree and \( k(i,3) \) rarity distribution), times are printed for each \( W \) elements hashed, considering the use of 100 and 1000 hash functions. For the \( k(i,3) \) column, values for \( W \) of triples processed were add in parenthesis.

<table>
<thead>
<tr>
<th>( W )</th>
<th>#h</th>
<th>I</th>
<th>WI</th>
<th>( k(i,3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1,000</td>
<td>0.003</td>
<td>0.03</td>
<td>9.36 (0.015)</td>
</tr>
<tr>
<td>1000</td>
<td>1,000</td>
<td>0.03</td>
<td>0.67</td>
<td>118.53 (0.19)</td>
</tr>
<tr>
<td>100</td>
<td>10,000</td>
<td>0.03</td>
<td>0.37</td>
<td>131.01 (0.21)</td>
</tr>
<tr>
<td>1000</td>
<td>10,000</td>
<td>0.30</td>
<td>10.22</td>
<td>1166.61 (1.87)</td>
</tr>
<tr>
<td>100</td>
<td>100,000</td>
<td>0.32</td>
<td>3.90</td>
<td>1272.67 (2.04)</td>
</tr>
<tr>
<td>1000</td>
<td>100,000</td>
<td>2.93</td>
<td>141.35</td>
<td>11765.96 (18.86)</td>
</tr>
<tr>
<td>100</td>
<td>1,000,000</td>
<td>3.24</td>
<td>40.54</td>
<td>12776.61 (20.48)</td>
</tr>
<tr>
<td>1000</td>
<td>1,000,000</td>
<td>29.32</td>
<td>1708.30</td>
<td>117859.27 (188.92)</td>
</tr>
</tbody>
</table>
Chapter 5

Mining the Inner Structure of the Webgraph

5.1 The bow-tie picture

The first large-scale study of the Web graph was performed by Broder et al. [18] and it revealed that the Web graph contains a giant component that consists of three distinct components of almost equal size: the CORE, made up of a single strongly connected component; the IN set, comprised by nodes that can reach the CORE but cannot be reached by it; the OUT set, consisting of nodes that can be reached by the CORE but cannot reach it. These three components form the well known bow-tie structure of the Web graph, shown in Figure 5.1.

The bow-tie picture describes the macroscopic structure of the Web. However, very little is known about the inner structure of the components that comprise it. Broder et al. [18] pose it as an open problem to study further the structure of those components. Understanding the finer details of the Web graph is an interesting problem on its own, but it is also important in practice for improving the performance of algorithms that rely on the link structure of the Web. Furthermore, it could be useful for refining the existing stochastic models for the Web [9, 64, 46].

The study of the Web graph poses additional challenges. Typically, the Web graph consists of millions of nodes and billions of edges. Performing standard graph algorithms (such as BFS and DFS) on a graph of this size is a non-trivial task since data cannot be stored in main memory. It is therefore necessary to devise external-memory algorithms [21] that can work on massive graphs. The challenge is to customize the algorithms to the Web graph, taking advantage of the specific structure of the Web.

In this chapter we study the finer structure of the Web graph, addressing the open question raised by Broder et al. [18]. We refine the bow-tie picture by providing

\footnote{The figure is reproduced from [18]}
details for its individual components. In the process we develop a suite of algorithms for handling massive graphs. Our contributions can be summarized as follows.

- We implement a number of external and semi-external memory graph theoretic algorithms for handling massive graphs, which can run on computers with limited resources. Our algorithms have the distinct feature that they exploit the structure of the Web in order to improve their performance. A detailed description of these algorithms is in the Chapter 2.

- We experiment with four different crawls and we observe the same macroscopic properties previously reported in the literature: the degree distributions follow a power-law, and the graph has a bow-tie structure, although (depending on the crawler) a little different in shape.

- We study in detail the inner structure of the bow-tie graph. We perform a series of measurements on the CORE, IN and OUT components. Our measurements reveal the following surprising fact: although the individual components share the same macroscopic statistics with the whole Web graph, they have substantially different structure. We suggest a refinement of the bow-tie picture, the daisy structure of the Web graph, that takes our findings into account.

The rest of the Chapter is structured as follows. In Section 5.2 we present our experimental findings. We conclude in Section 5.3 with a discussion on the implications of our findings, and possible future experiments.
5.2. EXPERIMENTS AND RESULTS

We experiment with four different crawls. The first three crawls are samples from the Italian Web (the .it domain), the Indochina Web (the .vn, .kh, .la, .mm, and .th domains), and the UK Web (the .uk domain) collected by the "Language Observatory Project"\(^2\) and the "Istituto di Informatica e Telematica"\(^3\) using Ubicrawler \(^1\). The fourth crawl is a sample of the whole Web, collected by the WebBase project at Stanford\(^4\) in 2001. This sample contains 360 millions of nodes and 1.5 billion of edges. In order to eliminate non-significant data, we pruned the frontier nodes (i.e. the nodes with in-degree 1 and out-degree 0, on which the crawler has been arrested). The sizes of the crawls are shown in Table 5.1 and 5.2.

\(^2\)www.language-observatory.org
\(^3\)www.itt.cnr.it
\(^4\)http://www-diglib.stanford.edu/testbed/doc2/WebBase/

<table>
<thead>
<tr>
<th></th>
<th>Italy</th>
<th>Indochina</th>
<th>UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>41.3M</td>
<td>7.4M</td>
<td>18.5M</td>
</tr>
<tr>
<td>edges</td>
<td>1.15G</td>
<td>194.1M</td>
<td>298.1M</td>
</tr>
<tr>
<td>CORE</td>
<td>29.8M (72.3%)</td>
<td>3.8M (51.4%)</td>
<td>1.2M (65.3%)</td>
</tr>
<tr>
<td>IN</td>
<td>13.8K (0.03%)</td>
<td>48.5K (0.66%)</td>
<td>312.6K (1.7%)</td>
</tr>
<tr>
<td>OUT</td>
<td>11.4M (27.6%)</td>
<td>3.4M (45.9%)</td>
<td>5.9M (31.8%)</td>
</tr>
<tr>
<td>TENDRILS</td>
<td>6.4K (0.01%)</td>
<td>50.4K (0.66%)</td>
<td>139.4K (0.8%)</td>
</tr>
<tr>
<td>DISC</td>
<td>1.25K (0%)</td>
<td>101.1K (1.4%)</td>
<td>80.2K (0.4%)</td>
</tr>
</tbody>
</table>

Table 5.1: Sizes and bow-tie components for the crawls of Italy, Indochina and UK

<table>
<thead>
<tr>
<th></th>
<th>WebBase</th>
<th>AltaVista</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>135.7M</td>
<td>203.5M</td>
</tr>
<tr>
<td>edges</td>
<td>1.18G</td>
<td>1.46G</td>
</tr>
<tr>
<td>CORE</td>
<td>44.7M (32.9%)</td>
<td>56.4 (27.7%)</td>
</tr>
<tr>
<td>IN</td>
<td>14.4M (10.6%)</td>
<td>43.3 (21.3%)</td>
</tr>
<tr>
<td>OUT</td>
<td>53.3M (39.3%)</td>
<td>43.1 (21.2%)</td>
</tr>
<tr>
<td>TENDRILS</td>
<td>17.1M (12.6%)</td>
<td>43.8 (21.5%)</td>
</tr>
<tr>
<td>DISC</td>
<td>6.2M (4.6%)</td>
<td>16.7 (8.2%)</td>
</tr>
</tbody>
</table>

Table 5.2: Sizes and bow-tie components for WebBase and the Alta Vista graph

5.2 Experiments and results
CHAPTER 5. MINING THE INNER STRUCTURE OF THE WEBGRAPH

5.2.1 Macroscopic measurements

As a first step in our analysis of the Web graph, we repeat the experiments of Broder et al. [18] on the macroscopic analysis of the graph. We computed the in-degree, out-degree and SCC size distributions. As expected, the in-degrees, and the sizes of SCCs follow a power-law distribution, while the out-degree distribution follows an imperfect power-law. All our measurements are in agreement with the respective measurements of Broder et al. [18] for the Alta-Vista crawl. More detailed results on the various distributions for the WebBase crawl are reported in [51].

We also computed the macroscopic structure of the Web graph. We observe a bow-tie structure. The relative sizes of the components of the bow-tie are shown in Table 5.1 and Table 5.2, where we also present the numbers for the AltaVista crawl [18], for the purpose of comparison. The first observation is that for the Italian, Indochina, and UK crawls, the IN and TENDRILS components are almost non-existent. As a result either the CORE is overgrown (for the Italian and UK crawls), or the nodes are equally distributed between the CORE and the OUT component. For the WebBase crawl we observe that the relative size of IN (11%) is significantly smaller than that observed in the AltaVista crawl, while the OUT component (39%) is now the largest component of the bow-tie. These discrepancies with the AltaVista crawl can most likely be attributed to different crawling strategies and capabilities, rather than to the evolution of the Web. The first three crawls are relatively recent, and all crawls are generated using a small number of starting points. Unfortunately, large-scale crawls are not publicly available.

5.2.2 The inner structure of the bow-tie

We now study the fine-grained structure of the Web graph. We are interested in understanding not only the characteristics of each component individually, but also how the components relate to each other. For this purpose we label each node with the name of the component to which it belongs. This gives us five sets of nodes (CORE, IN, OUT, TENDRILS, DISC). For each such subset we obtain the induced subgraph, resulting in five different subgraphs. For example, when referring to the IN graph, we mean the graph that consists of the nodes in IN and all the edges between these nodes.

As a first step in the understanding of the individual components we compute the same macroscopic measures as for the whole Web graph. We compute the in-degree, out-degree and SCC size distributions for each of the IN, OUT, TENDRILS and DISC graphs. Figure 5.2 shows the plots of the distributions for each component and for the whole graph, for the case of the WebBase crawl. It is obvious that the same macroscopic laws that are observed on the whole graph are also present in the individual components.
The structure of the IN and OUT components

Given the fact that the in-degree, out-degree, and SCC size distributions in the IN and OUT components are the same as for the whole Web graph, it is tempting to conjecture that the Web has a self-similar structure. That is, the bow-tie structure repeats itself inside the IN and OUT components. Dill et al. [31] demonstrated that the web exhibits self-similarity when considering “thetically unified” sets of web pages. These subsets are structurally similar to the whole Web. Similar observations are made by Pennock et al. [64]. However, the subsets considered by these previous works are composed by nodes that may belong to any of the components of the bow-tie graph. The question we are interested in is if such self-similarity appears when considering the individual components of the bow-tie graph.

The first indication that the self-similarity conjecture is not true comes from the fact that there is no large SCC in the IN and OUT components. For the OUT component, in all crawls, the largest SCC is only a few thousands of nodes. Given that the size of the OUT component is in the order of millions, the largest SCC is staggeringly
small. Furthermore, this is also the second largest SCC in the graph, which, compared
the largest one (the CORE), is minuscule. We observe a similar phenomenon for the
IN component. For the WebBase graph (which is the most interesting case, since the
IN component is a non-trivial fraction of the graph) the largest SCC in the IN com-
ponent is less than 6,000 nodes. Detailed numbers about the size of the largest SCC
in the IN and OUT components are given in Table 5.3.

Therefore, it appears that there exists no sizable SCC in the IN and OUT com-
ponents that could play the role of the CORE in a potential bow-tie. However it is
still possible that there exists a giant weakly connected component (WCC) in each
component. We therefore computed the WCCs of the two sets. Surprisingly we dis-
covered that there is no giant WCC in either of the two components. In fact, there
is a large number of WCCs per component and their sizes follow a power law dis-
tribution. Figure 5.2(d) shows the WCC size distribution for the WebBase graph.
Statistics for all graphs are reported in Table 5.3. Most of the WCCs are of size one.
The singleton WCCs comprise 10-22% of the IN component (with the exception of
Indochina), and 20-45% of the OUT component. On the other hand, the largest WCC
is never more than 30% of the component it belongs to, which is small compared to
the giant WCC in the Web graph, which contains more than 90% of the nodes. For
the WebBase graph, the largest WCC in the IN component consists of just 1% of the
nodes, while the largest WCC in the OUT component consists of 28% of the nodes.

We also investigate how the nodes in the largest WCCs in the IN and OUT com-
ponents are connected to see if they organized in a bow-tie shape. Our investigation
revealed that starting from the largest SCC in the WCC, we can create a bow-tie that
is no more than 15% of the WCC (for the Italian Web), and usually less than 5%.
The rest belongs to the DISC component. (Note that a node that points to the ten-
drils coming out of IN, or is pointed to by those going into OUT, belongs to DISC,
although it is still weakly connected to the graph). This suggests that the WCC con-
sists of multiple small atrophic bow-ties that are sparsely interconnected with each
other.

In order to better understand how the nodes in IN and OUT are arranged with
respect to the CORE, we performed the following experiment. We condensed the
CORE in a single node and we performed a forward and a backward BFS. This allows
us to split the nodes in the IN and OUT components in levels depending on their
distance from the CORE. The depths of the components are shown in Table 5.4. In all
graphs, the depths of the components are relatively small. Furthermore, most nodes
are concentrated close to the CORE. Typically, about 80-90% of the nodes in the OUT
component are found within the first 5 layers. For the WebBase graph, although the
OUT is much deeper, with 580 levels, more than 58% of its nodes are at distance 1
from the CORE, and 93% are within distance 5. Furthermore, after level 305 there
exists only a single chain of nodes that extends until level 580, making the effective
depth of the OUT 305. The node distributions, level by level, for the WebBase graph
are shown in Figure 5.3(b) and 5.3(c), for the IN and OUT sets respectively. The
5.2. EXPERIMENTS AND RESULTS

Table 5.3: Statistics for the IN, OUT and CORE components for each crawl

<table>
<thead>
<tr>
<th>Component</th>
<th>Italy</th>
<th>Indochina</th>
<th>UK</th>
<th>WebBase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The IN component</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodes in IN</td>
<td>13.8K (0.03%)</td>
<td>48.5K (0.66%)</td>
<td>312.6K (1.69%)</td>
<td>14.4M (11%)</td>
</tr>
<tr>
<td>max SCC</td>
<td>1,590</td>
<td>7,867</td>
<td>4,171</td>
<td>5,876</td>
</tr>
<tr>
<td>number of WCCs</td>
<td>1,633</td>
<td>117</td>
<td>62K</td>
<td>3.68M</td>
</tr>
<tr>
<td>max WCC</td>
<td>4,085 (29.5%)</td>
<td>13.2K (27.2%)</td>
<td>8,246 (2.7%)</td>
<td>197.5K (1.3%)</td>
</tr>
<tr>
<td>singleton WCCs</td>
<td>1,543 (11.15%)</td>
<td>63 (0.13%)</td>
<td>56K (17.89%)</td>
<td>3.2M (22.46%)</td>
</tr>
<tr>
<td><strong>The OUT component</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodes in OUT</td>
<td>11.4M (27.6%)</td>
<td>3.4M (45.9%)</td>
<td>5.9M (31.8%)</td>
<td>53.3M (39%)</td>
</tr>
<tr>
<td>max SCC</td>
<td>19,170</td>
<td>39,283</td>
<td>26,525</td>
<td>9,349</td>
</tr>
<tr>
<td>number of WCCs</td>
<td>3.73M</td>
<td>729.6K</td>
<td>1.97M</td>
<td>25.4M</td>
</tr>
<tr>
<td>max WCC</td>
<td>1.43M (12.52%)</td>
<td>335.9K (9.85%)</td>
<td>457.4K (7.75%)</td>
<td>14.94M (28.01%)</td>
</tr>
<tr>
<td>singleton WCCs</td>
<td>3.49M (30.6%)</td>
<td>672K (19.71%)</td>
<td>1.84M (31.11%)</td>
<td>24.48M (45.91%)</td>
</tr>
<tr>
<td><strong>The CORE component</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodes in CORE</td>
<td>29.8M (72.3%)</td>
<td>3.8M (51.4%)</td>
<td>1.2M (65.28%)</td>
<td>44.7M (33%)</td>
</tr>
<tr>
<td>entry points</td>
<td>10.2K (0.03%)</td>
<td>2.3K (0.06%)</td>
<td>106.3K (0.88%)</td>
<td>2.6M (5.87%)</td>
</tr>
<tr>
<td>exit points</td>
<td>15.6M (52.2%)</td>
<td>2.3M (59.6%)</td>
<td>4.8M (39.8%)</td>
<td>29.6M (72.03%)</td>
</tr>
<tr>
<td>bridges</td>
<td>6.25K (0.02%)</td>
<td>1.5K (0.04%)</td>
<td>61.8K (0.51%)</td>
<td>2M (4.58%)</td>
</tr>
<tr>
<td>connectors</td>
<td>1.7M (5.71%)</td>
<td>164.2K (4.32%)</td>
<td>537.9K (4.45%)</td>
<td>2.96M (6.63%)</td>
</tr>
<tr>
<td>petals</td>
<td>325.3K (1.09%)</td>
<td>52.5K (1.38%)</td>
<td>138K (1.14%)</td>
<td>1.4M (3.14%)</td>
</tr>
</tbody>
</table>
CHAPTER 5. MINING THE INNER STRUCTURE OF THE WEBGRAPH

![Graphs showing distribution of IN and OUT nodes level by level](image)

Figure 5.3: Characteristics of the IN and OUT components

<table>
<thead>
<tr>
<th></th>
<th>Italy</th>
<th>Indochina</th>
<th>UK</th>
<th>WebBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth IN</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>depth OUT</td>
<td>26</td>
<td>21</td>
<td>25</td>
<td>580</td>
</tr>
</tbody>
</table>

Table 5.4: IN and OUT depth

plots are in logarithmic scale.

Therefore, we conclude that the IN and OUT components are shallow and highly fragmented. They are comprised of several sparse weakly connected components of low depth. Most of their volume consists of nodes that are directly linked to the CORE.

The structure of the CORE

As a first step in the study of the CORE graph, we examine its relation with the IN and OUT components. A first attempt to refine the notation of [18] is presented in [7] where the analysis, focused on the website connectivity of the Chilean domain, shows several relation between the macro structure of the Web, site age, and quality of pages and sites. We define an entry point to the CORE to be a node that is pointed to by at least one node in the IN component, and an exit point to be a node that points to at least one node in the OUT component. A bridge is a node that is both an entry and an exit point. The number of entry and exit points is shown in Table 5.3. It is interesting to observe that a large fraction of the entry points act like bridges. Furthermore, with the exception of the UK crawl, the majority of the nodes in the CORE is connected to the “outside” world. In the WebBase crawl, this number is around 80% of the whole CORE, while the “deep CORE” consists of a little more than 20%.

We also compute the in-degree distribution of the entry points when we restrict
the source of the links to be in the IN component, and, as expected, we observe a power law. This implies that most nodes “serve” as entry points to just a few nodes in the IN component, while there exist a few nodes that serve as entry points to a large number of IN nodes. Similar distributions are obtained when we consider the out-degree distribution of the exit points, restricted to the OUT component.

We then study the connectivity of the CORE. We first look for nodes that are loosely connected to the CORE. We define a connector to be a node of the CORE that has a single in-coming and out-going link. A connector forms a petal if the source of the incoming link, and the target of the out-going link are the same node. Large number of connectors would imply weak connectivity of the CORE. The number of connectors is shown in Table 5.3 and it is on average around 5%. Of these 20 to 45% are petals. Therefore, connectors are only a small part of the CORE.

In order to further understand the connectivity of the CORE, we test the resilience of the CORE to targeted attacks by performing the following experiment. For some \( k \) we delete all nodes from the CORE that have total degree (in-degree plus out-degree) at least \( k \). We then compute the size of the largest SCC in the resulting graph. Table 5.5 shows how the size of the largest SCC changes as we decrease \( k \), and we increase the number of deleted nodes for the case of the WebBase graph. Similar trends are observed in the other crawls. We observe that the threshold on the total degree must become as low as 100 in order to obtain an SCC of size less than 50% of the CORE.

We note that there is a large discrepancy between the values of the in-degrees and out-degrees in the Web graph. The highest in-degree is close to 566K, while the highest out-degree is just 536. Note that an upper-bound on the out-degree may be imposed by the crawler, if it limits the number of outgoing links of a page that it explores. Therefore, it may be the case that when deleting the nodes with high total degree, we only delete nodes with high in-degree. We experiment with a different kind of attack that removes (approximately) \( k \) nodes with the highest in-degree and \( k \) nodes with the highest out-degree. The results are shown in Table 5.6. The CORE remains resilient even against this combined attack. An interesting observation while performing this experiment was that the nodes with the highest in-degree and the nodes with the highest out-degree are quite distinct. Actually, the correlation between the in-degree and out-degree is close to zero. It appears that nodes that are strong hubs in the CORE are not also strong authorities.

There are two ways to interpret these results. The first is that there are no obvious failure points in the CORE, that is, strong hubs or authorities that pull the rest of the nodes together, and whose removal from the graph causes the immediate collapse of the network. In order to disconnect the CORE you need to remove nodes with sufficiently low degree. On the other hand, note that we managed to reduce the largest SCC to 35-40% of the original by removing about 1M nodes. However this is less than 1% of the total nodes. In that sense the CORE is vulnerable to targeted attacks.
### Table 5.5: Sensitivity of the CORE under targeted attacks: deleting nodes with high total degree

<table>
<thead>
<tr>
<th>deg</th>
<th>del</th>
<th>max SCC</th>
<th>max SCC %</th>
<th>SCC num</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>9</td>
<td>44.2M</td>
<td>98.9%</td>
<td>81K</td>
</tr>
<tr>
<td>33,600</td>
<td>19</td>
<td>44,005,976</td>
<td>98.4%</td>
<td>114,900</td>
</tr>
<tr>
<td>21,500</td>
<td>39</td>
<td>43.7M</td>
<td>97.9%</td>
<td>175K</td>
</tr>
<tr>
<td>14,500</td>
<td>101</td>
<td>43,530,874</td>
<td>97.4%</td>
<td>224,732</td>
</tr>
<tr>
<td>10,000</td>
<td>199</td>
<td>43.2M</td>
<td>96.6%</td>
<td>285K</td>
</tr>
<tr>
<td>6,000</td>
<td>542</td>
<td>42,717,973</td>
<td>95.5%</td>
<td>415,906</td>
</tr>
<tr>
<td>4,000</td>
<td>1.1K</td>
<td>42.3M</td>
<td>94.7%</td>
<td>505K</td>
</tr>
<tr>
<td>1,000</td>
<td>55K</td>
<td>35.1M</td>
<td>78.6%</td>
<td>3.7M</td>
</tr>
<tr>
<td>500</td>
<td>120K</td>
<td>31M</td>
<td>69.6%</td>
<td>5.7M</td>
</tr>
<tr>
<td>100</td>
<td>1.03M</td>
<td>14.8M</td>
<td>34.6%</td>
<td>14.7M</td>
</tr>
</tbody>
</table>

### Table 5.6: Sensitivity of the CORE under targeted attacks: deleting nodes with high in-degree and out-degree

<table>
<thead>
<tr>
<th>in-deg</th>
<th>del</th>
<th>out-deg</th>
<th>del</th>
<th>total del</th>
<th>max SCC</th>
<th>max SCC %</th>
<th>SCC num</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,000</td>
<td>1.1K</td>
<td>233</td>
<td>1,154</td>
<td>2,263</td>
<td>42.2M</td>
<td>94.4%</td>
<td>595K</td>
</tr>
<tr>
<td>2,850</td>
<td>5.239</td>
<td>200</td>
<td>5,975</td>
<td>11,212</td>
<td>40.9M</td>
<td>91.5%</td>
<td>1,204,071</td>
</tr>
<tr>
<td>2,600</td>
<td>9.9K</td>
<td>185</td>
<td>10K</td>
<td>20.6K</td>
<td>39.8M</td>
<td>89.0%</td>
<td>1.75M</td>
</tr>
<tr>
<td>1,750</td>
<td>26K</td>
<td>158</td>
<td>25K</td>
<td>51K</td>
<td>37M</td>
<td>82.9%</td>
<td>3M</td>
</tr>
<tr>
<td>1,000</td>
<td>52K</td>
<td>130</td>
<td>54K</td>
<td>105K</td>
<td>33.7M</td>
<td>75.5%</td>
<td>4.75M</td>
</tr>
<tr>
<td>500</td>
<td>112K</td>
<td>105</td>
<td>108K</td>
<td>219K</td>
<td>29.4M</td>
<td>66.1%</td>
<td>7M</td>
</tr>
<tr>
<td>125</td>
<td>259K</td>
<td>82</td>
<td>227K</td>
<td>487K</td>
<td>23.5M</td>
<td>53.3%</td>
<td>10M</td>
</tr>
<tr>
<td>120</td>
<td>518K</td>
<td>62</td>
<td>499K</td>
<td>949K</td>
<td>17.8M</td>
<td>40.8%</td>
<td>13M</td>
</tr>
</tbody>
</table>

Table 5.5: Sensitivity of the CORE under targeted attacks: deleting nodes with high total degree

Table 5.6: Sensitivity of the CORE under targeted attacks: deleting nodes with high in-degree and out-degree
5.3 Conclusion

In this Chapter we undertook a study of the Web graph at a finer level. We observed that the ubiquitous presence of power laws describing several properties at a macroscopic level does not necessarily imply self-similarity in the individual components of the Web graph. Indeed, the different components have quite distinct structure, with the IN and OUT being highly fragmented, while the CORE being well interconnected.

Our work suggests a refinement of the bow-tie pictorial view of the Web graph [18]. The bow-tie picture seems too coarse to describe the details of the Web. The picture that emerges from our work can better be described by the shape of a daisy (Figure 5.4): the IN and OUT regions are fragmented into large number of small and shallow petals (the WCCs) hanging from the central dense CORE.

It would be interesting to obtain larger, and more “realistic” crawls, and perform the same measurements to verify our hypothesis. Our current results are sensitive to the choices and limitations of the crawlers, and it is not clear if the available crawls are representative of the actual Web graph. Unfortunately, there are no publicly available crawls that have been collected with the aim of validating our hypothesis on the structure of the Web graph. We plan in the future to collect crawls with this goal in mind.

A deeper understanding of the structure of the Web graph may also have several consequences on designing more efficient crawling strategies. The fact that IN and OUT are highly fragmented may help in splitting the load between different robots without much overlapping. Moreover, the fact that most of the vertices are at few hops from the CORE may explain why breadth first search crawling is more effective than other crawling strategies [60].

Our work motivates further experiments on the Web graph. It would be inter-
Interesting to devise efficient algorithms for estimating the clustering coefficient, a commonly used measure for connectivity. Furthermore, further exploration of the structure of the CORE is necessary to gain a deeper understanding. Possible measurements could include spectral properties, or clustering and community discovery. As a concluding remark, we observe that we are still very far from devising a theoretical model that is able to capture the finer connectivity properties of the Web graph.
Chapter 6

Temporal Analysis of Wikigraphs

6.1 The role of the time in the study of the Web

The role of time is becoming more and more crucial in order to develop search-engine algorithms able to provide the final user with the most up-to-date results possible. In the past decade, the Web has experienced a very rapid growing rate and the most recent research [41] estimates that the indexable web exceeds 11.5 billion pages. Because of this huge amount of data, the search engines have to constantly afford the burdensome task of updating their index in order to keep in touch with the evolving Web.

The study of evolving web has been mainly focused on the degree and the frequency of changes in the Web pages. The statistics collected on sequential crawls of the web have been used to devise incremental crawlers able to guarantee high freshness of the pages in their indexes. The rapid rate of the pages change is not the main feature of the web. It has been observed [62] that the hyperlinked structure evolution is even more dynamic. After one year, the percentage of initial links still present in the Web is only 24% against a number of unchanged pages that reaches 50%. This characteristic is even more important since the hyperlinked structure is the basis of the algorithms that assign an authoritativeness score to the pages. To capture the relation between the popularity, authority and time, some recent studies [5, 47, 48] have presented structures that directly couple hyperlinks with temporal data.

It is worth to underline that extracting the link structure of the web at a certain point in time is not simple. An attempt can be made by collecting a series of static snapshots by sequential crawls. From the analysis of these snapshots, it can be inferred if a page has been modified or deleted during a certain time frame but it is not possible to determine exactly the instant when the update or deletion occurred. The attempt mentioned in [5] to obtain the update time from the HTTP server answers is not efficient. Only 40% of the HTTP headers present time information (i.e. creation and last update time) and a '404' answer do not necessary imply that the page was...
CHAPTER 6. TEMPORAL ANALYSIS OF WIKIGRAPHS

deleted.
In [48], the authors overcome this problem considering the evolution of the Blogspace, the space of weblogs (or blogs). A blog is commonly a page that contains a series of dated entries. Each time that a new entry is inserted, the page can be considered updated. In this way, all the information concerning the “time” can be directly extracted.

The same feature characterizes Wikipedia, an on-line and free content encyclopedia written in more than 100 languages, whose evolution we analyze in this experimental work. First of all, it is important to stress that although the temporal information is not directly comprised within each page as for the blogs, it is possible to obtain for each page both the old version and the complete list of updates (more details in Section 6.3). Since each language is formed by an independent subset of articles, we can have a continuous time vision of the evolution for each of them.

There are a number of reasons that lead us to consider the Wikipedia encyclopedia (www.Wikipedia.org) a good dataset for webgraph-type experiments:

• **sociological reasons**: the encyclopedia collects pages written by a number of independent and heterogeneous individuals. Each of them autonomously decides the content of their articles with the only constraint of a prefixed layout. The autonomy is a common feature of the content creation in the Web. The Wikipedia authors’ community is comprised by members that are pushed by the only wish to make available to the world concepts and topics that they consider meaningful. In some sense, the evolution of the Wikipedia subsets highlights the develop of significant trends within each linguistic community.

• **generation on time**: Wikipedia provides time information associated with nodes. Moreover, it provides old information: time information regarding the creation and the updates for each page on the dataset.

• **independence of external links**: Wikipedia articles link mainly to articles on the same dataset.

• **variety of graph sizes**: since we have one graph for each language, the graph dimensions vary from a few hundred up to half million pages.

We present a study of the hyperlinked graphs originated from the link structure of the pages of the online encyclopedia Wikipedia. This work aims to verify if any evolving trend is observable in the statistical properties of the Web (degree, PR, number of updates) and if these measures are correlated each other over the time. We want to stress that, up to now, very little research work has been devoted to the evolution of the statistical and topological properties of hyperlinked graph as webgraphs, blog graphs and wikigraphs.

The study of the topological properties of the Web has started in [9] [50]. A more complete analysis of the webgraph was later presented in [18] where many measures
of the Web were presented together with the bow-tie picture, a macroscopic characterization of the Web structure. Later, these results were extended by an extensive study of a large sample of the Web provided by the Webbase project [32].

The study reveals that the wikigraphs share the same properties of the samples collected crawling large portions of the Web and that they are characterized by a well-connected structure with more than 80% of the nodes belonging to a large strongly connected component. The temporal analysis shows that

- the single snapshots exhibit very similar properties with respect to indegree and outdegree distribution;
- the number of updates of pages is concentrated in the few first months of its creation.
- the number of updates is not correlated with Pagerank, indegree, outdegree and number of visits.

### 6.2 Related Work

The study of the temporal evolution of webgraphs has already been addressed in several previous works [5, 8, 7, 22, 38, 62]. Most of them traced a set of pages in order to compile some statistics about the frequency and rate of the changed pages and the percentage of pages that are deleted or created every year. The search engine perspective is dominant in all of them.

The paper [22] presents the results of an experiment conducted over 4 months. The authors daily crawled 270 sites in order to measure the rate of change and the lifespan of each page. A Poisson process was used to model the rate of change and compare the efficiency of different crawling strategies. The authors also described the architecture of an incremental crawler able to keep up the index with the evolving web.

Fetterly et al. [38] expanded the work of [22] both in terms of coverage and sensitivity to changes. They found out that good predictors of future changes in the web are the top-level domain pages, and relate document size and history to the freshness of a web page collection.

A search engine-centric approach is followed also in [62]. The authors crawled 154 ‘popular’ sites for a year and revealed a high dynamical behavior of the Web. But, despite of the high rate of newly created pages, the ‘new contents’ introduced are less than 5% of all changes introduced. They also observed that the Web link structure is even more dynamic with more than 75% of new links every year. Moreover they found out that, for pages with significant changes over the time, the degree of changes tends to be highly predictable and observed that this results can be used to crawl proper portions of the Web.
CHAPTER 6. TEMPORAL ANALYSIS OF WIKIGRAPHS

Models for analyzing the evolution of the webgraph were presented in [47], [48]. In particular Kraft et al. [47] defined the notion of *TimeLinks* and extract some statistics over the data. Kumar et al. [48] introduced the notion of *time graph* and conducted a series of experiments in order to trace the formation and the development of communities in the Blogspace and to detect burst of activity within them.

6.3 The Data Sets

Wikipedia is an on-line and free content encyclopedia. The first few English pages were published in January 15, 2001. Four and half years later, Wikipedia has more then 1 million articles, available in more then 100 languages:

- English dataset with more than 581 thousands articles;
- German dataset with 239 thousands articles;
- French and Japanese datasets with a bit more then 100 thousand articles;
- Dutch, Polish, Portuguese, Spanish and Swedish datasets with more then 50 thousand articles;
- 27 other datasets with more than 10 thousand articles;
- 41 other datasets with more than a thousand articles;
- 32 other datasets with more than a hundred articles.

The datasets of each language are available in two self-extracting files for *mysql* database. The table *cur* contains the current on-line articles, whereas the table *old* contains all previous versions of each article in the *cur* table. Old versions of an article are identified by the title, and not the same id. The dataset dumps are updated almost weekly, so the current graph is usually not more than a week old.

In order to generate a graph from the link structure of a dataset, each article corresponds to a node and each hyperlink between articles to an edge in this graph. In the Wikipedia datasets, each webpage is a single article, i.e. an item of the encyclopedia. An article also might contain some external links that point to pages outside the dataset. Very few Wikipedia articles have external links. These kind of links are not considered for generating wikigraphs, since we want to restrict the graph to pages into the set being analyzed.

The file *cur* is in fact a sql script able to create a mysql table with 16 fields. For our purpose, the fields of interest for each article are the *id*, the *title*, the *text*, a *timestamp* indicating its last change, and a *counter* that takes into the account the number

6.3. THE DATA SETS

of times the page was visited. Moreover, the fields namespace and is_redirect are important in order to identify real articles. The namespace field is set with values from 0 to 7, and just the value of namespace=0 indicates a proper article. Other settings of namespace refer to the user that last modified the article, discussions between the work group of Wikipedia, images, etc. The field is_redirect is set to 0 if it is active or to 1, if it was inserted in the content of another page, or if its title was modified. Pages set with is_redirect = 1 are pages that contain just one link to the page where its content was transferred. So, they are not real articles and we do not include such kind of nodes in the wikigraphs. The file old contains just 10 of the 16 fields existing in tables cur.

A graph with timestamps is generated making use of both tables. Besides of the original fields obtained from the table (cur), each node has a table with timestamps starting with the time the node was created, a list of timestamps indicating the time of each update and an end time, that is open for the current nodes. Similar temporal information is associated with links. A link is considered updated every time one of its endpoints is updated.

We generated six wikigraphs, wikiEN, wikiDE, wikiFR, wikiES, wikiIT and wikiPT, from the English, German, French, Spanish, Italian and Portuguese datasets, respectively. The graphs were obtained from an old dump of June 13, 2004. We are not using the most recent data due to disk space restrictions. The compressed English dataset we analyzed requires more than 36 GB, that correspond to about 200 GB after the extraction. In a future analysis we intend to extend the experiments for the current graphs and compare with the results found for these old datasets. Table 6.1 presents the dimensions of such graphs, as well as the average (#avg), and maximum values for the indegree (max_in) and outdegree (max_out), respectively.

Table 6.1: Dimensions of the wikigraphs used in the experiments.

| DB | |V| | |E| | |avg| |max_in| |max_out| |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| PT  | 8,645 | 51,231 | 5.92 | 2,264 | 379 |
| IT  | 13,132 | 159,965 | 12.18 | 1,747 | 1,244 |
| ES  | 27,262 | 288,766 | 10.59 | 2,973 | 612 |
| FR  | 42,987 | 660,401 | 15.36 | 7,570 | 2,247 |
| DE  | 116,251 | 2,163,405 | 18.61 | 5,580 | 3,136 |
| EN  | 339,834 | 5,278,037 | 15.53 | 46,992 | 3,524 |

Table 6.2 presents some statistics on the temporal information of the wikigraphs. The month and the year of the dataset creation is shown in the first column of the table (age). The average and maximum number of updates (#V_upd) and number of visits (#vst) of the articles are presented for each data set. The average number
of outdegree and indegree is the same, since both are obtained from the ratio of the number of edges by the number of nodes. The pages mostly visited are the main page of each dataset (the datasets wikiIT and wikiPT do not provide information on the number of visits).

Table 6.2: Statistics for some basic properties of the wikigraphs.

<table>
<thead>
<tr>
<th>DB</th>
<th>age</th>
<th>#V_upd</th>
<th>#vst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>max</td>
<td>avg</td>
</tr>
<tr>
<td>PT</td>
<td>Jun/01</td>
<td>2.49</td>
<td>239</td>
</tr>
<tr>
<td>IT</td>
<td>Aug/01</td>
<td>4.31</td>
<td>1505</td>
</tr>
<tr>
<td>ES</td>
<td>Aug/02</td>
<td>0.61</td>
<td>264</td>
</tr>
<tr>
<td>FR</td>
<td>Jun/01</td>
<td>9.52</td>
<td>1,985</td>
</tr>
<tr>
<td>DE</td>
<td>May/01</td>
<td>9.77</td>
<td>1,479</td>
</tr>
<tr>
<td>EN</td>
<td>Jan/01</td>
<td>8.76</td>
<td>4,986</td>
</tr>
</tbody>
</table>

6.4 Link Analysis of Wikigraphs

As a very first part of this work, we show the results of the same static analysis outlined in [18] by Broder et al. that we performed in order to emphasize the similarities between the common Webgraphs and the Wikigraphs. Consequently we carried out different kind of measurements. We present:

1. the distribution of local measures as indegree and outdegree;
2. the size of the bow-tie components;
3. the Pagerank values for all the pages.

All these experiments use a publicly available library of software tools for handling large networks [33].

The first set of measurements concerns the indegree and outdegree distributions. Figure 6.1 presents the indegree distribution plot in log scale. The distribution follows a power law. We found $\gamma = 2.1$, as it has been observed in the indegree distribution of Webgraphs [18, 32]. The outdegree distribution, shown by figure 6.2 also follows a power law with $\gamma = 2.4$.

The macroscopic connectivity structure of Wikipidia has been characterized by mapping the strongly connected components of Wikigraphs. The similar analysis applied to the Webgraph and that we presented in Chapter 2 has revealed the bow-tie structure [18]. We recall that the bow-tie structure is formed of four components.

\footnote{i.e. the probability that a node has in-degree $i$ is proportional to $\frac{1}{\gamma^i}$, for $\gamma > 1$}
6.4. LINK ANALYSIS OF WIKIGRAPHS

Figure 6.1: Indegree distribution for the graph wikiEN

Figure 6.2: Outdegree distribution for graph wikiEN
The main component is a large strongly connected component \textit{CORE}, comprised of all nodes that can reach each other along directed edges. The second and third components are the \textit{IN} and \textit{OUT} sets. The \textit{IN} is the set of nodes that can reach the \textit{CORE} but cannot be reached from it, whereas the \textit{OUT} is the set of nodes that are reached by the \textit{CORE} but cannot reach it. Finally, the set of nodes that cannot reach or be reached from the \textit{CORE} are the \textit{TENDRILS}. There are nodes that are reachable from portions of \textit{IN} or reach portions of \textit{OUT}. Those \textit{TENDRILS} that leave a set of nodes from \textit{IN} and enter a set of nodes in \textit{OUT} are called \textit{TUBES}. It can be observed that a significant portion of the nodes are in the large strongly connected component \textit{CORE}. We can also distinguish sets of nodes completely separated by the main bow-tie, called \textit{DISCS}. However the most of these disconnected components are arranged in small bow-tie shapes.

We measure the size of each component and the results are presented in table 6.3.

Table 6.3: Size of the bow-tie components of the Wikigraphs. Each entry in the table presents the percentage of nodes of the corresponding Wikigraph that belong to the indicated bow-tie component.

<table>
<thead>
<tr>
<th>DB</th>
<th>CORE</th>
<th>IN</th>
<th>OUT</th>
<th>TENDRILS</th>
<th>TUBES</th>
<th>DISC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT</td>
<td>67.14</td>
<td>6.79</td>
<td>15.85</td>
<td>1.65</td>
<td>0.03</td>
<td>7.50</td>
</tr>
<tr>
<td>IT</td>
<td>82.76</td>
<td>6.83</td>
<td>6.81</td>
<td>0.52</td>
<td>0.00</td>
<td>3.10</td>
</tr>
<tr>
<td>ES</td>
<td>71.86</td>
<td>12.01</td>
<td>8.15</td>
<td>2.76</td>
<td>0.07</td>
<td>6.34</td>
</tr>
<tr>
<td>FR</td>
<td>82.57</td>
<td>6.12</td>
<td>7.89</td>
<td>0.38</td>
<td>0.00</td>
<td>3.04</td>
</tr>
<tr>
<td>DE</td>
<td>89.05</td>
<td>5.61</td>
<td>3.95</td>
<td>0.10</td>
<td>0.00</td>
<td>1.29</td>
</tr>
<tr>
<td>EN</td>
<td>82.41</td>
<td>6.63</td>
<td>6.73</td>
<td>0.57</td>
<td>0.02</td>
<td>3.65</td>
</tr>
</tbody>
</table>

We can observe that the components sizes differ quite a bit from the previous measures of Webgraphs. About 30% of the nodes are in the core of Webgraphs \cite{18,32}, while the core of the different Wikigraphs contains a percentage of the nodes of the graph that ranges between 67% and 82%.

Hence we can conclude that the link structure of Wikipedia is well interconnected, in the sense that most of the nodes are in the core, and from any page it is possible to reach almost any other. Not surprisingly, this is probably due to the implicit aim of an online encyclopedia, that is driving the reader to related topics on the same encyclopedia during the topic description. In this way the content of each article can be fully understood while the surfer visits many articles.

We complete our link analysis by measuring the Pagerank distribution for wikiEN, presented in figure 6.3. It is a power law function with $\gamma = 2.1$. Previous measures for the Webgraphs \cite{32,63} also exhibit the same behavior for the Pagerank distribution.
Table 6.4 presents a list of the top pages considering the PageRank values, with the indegree \((\text{in})\) and outdegree \((\text{out})\) and number of visits \((\#\text{vst})\) of the pages. The indegree of such pages is high, as expected, while the outdegree is variable. We list the number of visits of the top ranked pages to show that this value is surprisingly not related with the PageRank values. This results was also presented in [67], in which very little correlation was found between the link analysis characteristics and the actual number of visits. We have to wonder if this finding is motivated by the nature of an encyclopedia or if it is a common feature of the Web. Indeed, since in Wikipedia, each item concerns a single topic and each topic is the content of exactly one article, it is not clear what we are going to measure with an algorithm like PageRank: each page is necessarily authoritative since it is the only one related with a particular issue. Instead, if this is a feature of the Web, then we might conclude that PageRank could not be a so good measure of the authoritativeness for the final users since they prefer to visit pages with lower PageRank values.

All the properties presented in this section, indegree and outdegree distributions, bow-tie measures and PageRank distribution, are similar on different datasets. Moreover, similar properties are also found in different snapshots of the same datasets. Some of the properties, mainly the PageRank distribution, are not so explicit when considering small graphs. For example, the PageRank of the graph \(\text{wikiPT}\) does not have a clear visible power law as it is found for the \(\text{wikiEN}\).

The next section explores statistics and properties associated with the temporal information of Wikigraphs. First we describe which kind of time information is iden-
Table 6.4: The ten top ranked pages and their corresponding indegree, outdegree and number of visits.

<table>
<thead>
<tr>
<th>Page title</th>
<th>in</th>
<th>out</th>
<th>#vst</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>25410</td>
<td>737</td>
<td>7</td>
</tr>
<tr>
<td>2000</td>
<td>46992</td>
<td>383</td>
<td>2</td>
</tr>
<tr>
<td>Asia</td>
<td>40807</td>
<td>97</td>
<td>6</td>
</tr>
<tr>
<td>Native America</td>
<td>39648</td>
<td>439</td>
<td>1</td>
</tr>
<tr>
<td>Hispanic</td>
<td>38638</td>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>Latino</td>
<td>38518</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>African American</td>
<td>39020</td>
<td>74</td>
<td>2</td>
</tr>
<tr>
<td>United States Census Bureau</td>
<td>35001</td>
<td>37</td>
<td>2</td>
</tr>
<tr>
<td>Asian</td>
<td>38633</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>France</td>
<td>13253</td>
<td>517</td>
<td>3</td>
</tr>
</tbody>
</table>

6.5 Temporal Information in the Wikigraphs

Before presenting results, we recall a few concepts that are used through this section. The first notion we introduce is the concept of timestamp. A timestamp is a temporal indication that we can associate with each graph component (i.e. nodes and edges). The timestamps are related to the different events that occur during the page life span. For example, the timestamps $ts_{begin}$, $ts_{end}$ and $ts_{upd}$ associated with a webpage indicate the time the page was generated, removed and updated, respectively. In this case, $ts_{upd}$ is a set of timestamps, and not just one reference.

In order to analyze the evolution of the graph over the time, we use the concept of snapshot. A snapshot is a picture of the graph in a specified moment in time. In other words, a snapshot is an old version of the current graph and such a version depends on the time it was taken. The precision on the timestamp is also an important information because it determines the granularity presented in the snapshots. For example, some users usually insert information of the last update of their page. For a home page, it would be enough indicate the day, month and year that was last updated. But if we consider an online timetable of an airport, we would need information at a finer level.

Wikipedia provides all the successive versions of each page. Each page is give with an identification (nodeID) and the three kinds of timestamps: $ts_{begin}$, $ts_{end}$ and $ts_{upd}$. The $ts_{begin}$ indicates the time the page was created whereas $ts_{end}$ indicates the time the page was removed from the graph. Otherwise, if the page is still alive in the current graph, $ts_{end}$ contains time set to infinity. The $ts_{upd}$ indicates the $i$-th update of the node and $ts_{begin} < ts_{upd} < ts_{end}$. In fact, due to
6.5. TEMPORAL INFORMATION IN THE WIKIGRAPHS

Figure 6.4: Temporal indegree distribution for the Wikigraph wikiEN

Figure 6.5: Temporal outdegree distribution for the Wikigraph wikiEN
the precision on the wikipedia timestamps, $t_{\text{begin}} \leq t_{\text{upd}} \leq t_{\text{end}}$. A wikipedia timestamp has indication of year, month, day, hour, minute and second, printed as a string on this sequence without separators. For example, the first English page has associated the timestamp 20010116200833. Having these temporal information, we can compute properties and statistics associated with the graph over time. Next section presents some computations considering the temporal information of the Wikigraphs.

### 6.6 Temporal Analysis of Wikigraphs

![Figure 6.6: Temporal Pagerank distribution for the Wikigraph wikiEN.](image)

In this section we present the temporal analysis of Wikigraphs. By temporal analysis we mean the measures that are related with the evolution of the graph over time. The analysis in these section aims to present measures about the frequency of page update and the distribution of the updates along the time life of a page.

We start the temporal analysis of Wikigraphs by plotting the indegree, outdegree and Pagerank distributions for temporal snapshots of Wikigraphs.

The Figures from 6.4 to 6.9 present these plots for the wikiEN e wikiDE datasets, respectively. For clarity, the figures present four plots per year, instead of once a month. For the indegree distribution of the wikiEN graph (figure 6.4), it is observed a gap on the number of vertexes with low indegree from July/2002 to October/2002. In this period the number of pages almost doubled. As expected, in their
6.6. TEMPORAL ANALYSIS OF WIKIGRAPHS

Figure 6.7: Temporal indegree distribution for the Wikigraph wikiDE

Figure 6.8: Temporal outdegree distribution for the Wikigraph wikiDE
CHAPTER 6. TEMPORAL ANALYSIS OF WIKIGRAPHS

Figure 6.9: Temporal Pagerank distribution for the Wikigraph wikiDE.

initial period of life they do not have many incoming links. This behavior is not observable in the wikiDE database. It is interesting to observe that in the same period, the number of pages with outdegree 8-10 increased considerably (figure 6.5). So, the peak around these values in the outdegree distribution is deriving from the new pages inserted in the dataset. The Pagerank distribution has high fluctuation when the database is small, but convergence to a power is observed as the dataset grows.

We proceed the temporal analysis of Wikigraphs by plotting the distribution of the number of the page updates. Figure 6.10 presents the distribution of number of nodes by the number of updates for the six Wikigraphs used for the tests. Each point presents the number of nodes (y axis) that were updated exactly $x$ times. This distribution is a power law with $\gamma = 1.9$ for the three larger datasets, and a slight different $\gamma$ for the other three. A power law also characterizes the distribution of the page updates when we concentrate on single snapshots. In this case the value of $\gamma$ depends on how close to the dataset creation each snapshot is taken. An example of this is shown in figure 6.11 for four snapshots of the graph wikiFR. Each point represents the number of nodes (y axis) that were updated exactly the corresponding number of times (x axis). The legend indicates the time the snapshot has been considered.

We draw a second experiment that aims to give an indication of the distribution of the updates over the time life of the pages. Consider the set of pages $U$ created in a given period. If we fix a percentage $p$, we can plot which is the percentage of the number of nodes (in that specific period of time) that had at least $p$ of their updates done at some specific time after the creation. Figure 6.12 presents the plots considering $p_1 = \epsilon\%$, $p_2 = 20\%$, $p_3 = 40\%$, $p_4 = 60\%$, $p_5 = 80\%$ and $p_6 = 100\%$. 
6.6. TEMPORAL ANALYSIS OF WIKIGRAPHS

Figure 6.10: Distribution of the number of updates per page for the Wikigraphs.

Figure 6.11: Number of pages (y axis) that were updated the corresponding number of times (x axis) plot for four snapshots of the graph wikiFR.
CHAPTER 6. TEMPORAL ANALYSIS OF WIKIGRAPHS

Figure 6.12: Plot of the percentage of updates over time, considering the graph wikiEN.

Figure 6.13: Percentage of distinct pages updated per month for wikiEN, wikiDE, wikiFR.
Figure 6.14: Percentage of the number of updates per month for wikiEN, wikiDE, wikiFR.

Figure 6.15: Community evolution on the wikiEN graph.
We use $0 < \epsilon \ll 1$ (red line) such that it plots the time when the first update was executed. The data is from the graph wikiEN.

Many pages, once created, are never updated, or just updated a few times. About 20% of the pages are never updated, as we can see from the plot on Feb/03 for $p_0 = 100\%$. About 20% of the pages are fully updated in their first month of existence. It could be the case that many pages are updated soon after they are created, but the reason in fact is that many pages are never updated. By the plot $p_1 = \epsilon\%$, we outline that many pages have an update soon after they are created. For example, 70% of the pages received the first update in their first month of life.

Figures 6.13 and 6.14 show the number of distinct pages that receive updates and the number of different updates for every month. As you can observe, the two functions show several peaks and seemingly very little correlation. The percentage of pages that are updated seems to grow with time, while the number of updates decrease with time.

We calculated the number of bipartite cliques $k(i,i)$ for $i = 1, 2, ..., \text{maxI}$, where \text{maxI}, in our case, is the maximum number of $i$ such that we found at least a web-community. Following the experiment reported in [38], we plot the total number of web communities, and the total number of nodes involved in webcommunities, for monthly snapshots. We used the heuristic algorithm for calculating cliques presented in [51]. Figure 6.15 presents the evolution of communities over time.

One further experiment was to measure the correlation between the number of updates of a page and the Pagerank, indegree, outdegree and number of visits of a page. From this experiment we concluded that there is no correlation between the number of updates and any one of the other measures. Motivated by the previous experiments, we plotted the sequence of updates for the five most frequently updated pages and for the five pages with higher Pagerank. Figures 6.16 to 6.20 present the plots for the most frequently updated pages, whereas Figures from 6.21 to 6.25 show the sequence of updates for the most highly ranked pages. We can observe that the frequency of updates increases with time and show several peaks. The peaks seen to be less predictable in the Pagerank plots than in the most updated pages.

6.7 Conclusions

In this Chapter we presented link and temporal analysis of Wikigraphs. We performed a series of measurements and observed that Wikigraphs resemble many characteristics of Webgraphs. The core of this study was the temporal analysis of Wikigraphs, where we made a large number of experiments on the evolution over time of the topological and statistical properties of Wikigraphs and made several observations on the frequency of update of the articles of Wikipedia. We devised a power
6.7. CONCLUSIONS

Figure 6.16: Frequency of the 4986 updates of the most updated page entitled “October_2003”.

Figure 6.17: Frequency of the 3289 updates of the second most updated page entitled “Main_Page”.
Figure 6.18: Frequency of the 2460 updates of the third most updated page entitled “George_W_Bush”.

Figure 6.19: Frequency of the 1429 updates of the fourth most updated page entitled “Jesus”.

law distribution on the number of vertices receiving a given number of updates and found out that the number of updates of a page is not correlated with the Pagerank, indegree, outdegree and the number of visits of that page. Finally, we conclude that we are not able to describe with a simple function the rate of updates of a page. Many pages are created and the respective updates are performed soon afterward, while for others the updates are spread over time, and there are also pages that have bunches of updates in some periods of life.

The observation that the Pagerank is not related with the number of visits opens a rather interesting research direction aimed to establish whether Pagerank is a good measure of the authoritativeness of the pages in Wikigraphs and which modifications should be introduced in order to tune its performances.
CHAPTER 6. TEMPORAL ANALYSIS OF WIKIGRAPHS

Figure 6.21: Frequency of the 1032 updates of the most ranked page entitled “United States”.

Figure 6.22: Frequency of the 444 updates of the second most ranked page entitled “2000”.
6.7. CONCLUSIONS

Figure 6.23: Frequency of the 148 updates of the third most ranked page entitled “Asia”.

Figure 6.24: Frequency of the 543 updates of the fourth most ranked page entitled “Native America”.

Figure 6.25: Frequency of the 48 updates of the fifth most ranked page entitled “Hispanic”
Part II

Link Analysis Ranking Algorithms
Chapter 7

Stability and Similarity of Link Analysis Ranking Algorithms

The area of Link Analysis Ranking (LAR) algorithms was introduced by the seminal works of Kleinberg [45], who proposed the HITS algorithm and of Brin and Page [16] with the well-known PAGERANK, used by Google. These algorithms use the hyper-linked structures to rank the results of search queries. The need of having results more and more accurate have been led, during the last seven years, to a number of modification, generalizations and improvements of these two original algorithms (see [14] and references within). This proliferating of LAR algorithms motivated the work of Borodin et al. [14] that introduced a theoretical framework to analyse and compare the properties of various ranking algorithms. In this chapter we undertake a theoretical analysis of the properties of the HITS algorithms on a broad class of random class. We present and extend the theoretical work of Borodin et al. [14] in section 7.3. The theoretical results concerning the stability and the similarity on the class of product graph are illustrated in section 7.4. Finally, in section 7.5 we study experimentally the similarity of HITS and INDEGREE on the WebBase sample. From now on, we use the term indegree to indicate the number of incoming links of a page whereas INDEGREE is used for the heuristic that allows to rank the pages based on their indegree value.

7.1 The Stability and Similarity Properties

The multitude of LAR algorithms creates the need for a formal framework for assessing and comparing their properties. Borodin et al., introduced such a theoretical framework in [14]. In this framework an LAR algorithm is defined as a function from a class of graphs of size $n$ to an $n$-dimensional real vector that assigns an authority weight to each node in the graph. The nodes are ranked in decreasing order of their weights. Borodin et al. [14] define various properties of LAR algorithms. In
this work we focus on stability and similarity. Stability considers the effect of small changes in the graph to the output of an LAR algorithm. Similarity studies how close the outputs of two algorithms are on the same graph.

Borodin et al. [14] considered the question of stability and similarity over an unrestricted class of graphs. They studied a variety of algorithms, and they proved that no pair of these algorithms is similar, and almost all algorithms are unstable. It appears that the class of all possible graphs is too broad to allow for positive results. This raises naturally the question whether it is possible to prove positive results if we restrict ourselves to a smaller class of graphs. Since the explosion of the Web, various stochastic models have been proposed for the Web graph [6, 9, 49, 3]. The model we consider, which was proposed by Azar et al. [6], is the following: assume that every node \( i \) in the graph comes with two parameters \( a_i \) and \( h_i \) which take values in \([0, 1]\). For some node \( i \), the value \( h_i \) can be thought of as the probability of node \( i \) to be a good hub, while the value \( a_i \) is the probability of the node \( i \) to be a good authority. We then generate an edge from \( i \) to \( j \) with probability proportional to \( h_i a_j \). We will refer to this model as the product model, and the corresponding class of graphs as the class of product graphs. The product graph model generalizes the traditional random graph model of Erdős and Rényi [36] to include graphs where the expected degrees follow specific distributions. This is of particular interest since it is well known [49, 18] that the indegrees of the nodes in the Webgraph follow a power law distribution.

In this Chapter we study the behavior of the HITS algorithm, proposed by Kleinberg [45], on the class of product graphs. The study of HITS on product graphs was initiated by Azar et al. [6] who showed that under some assumptions the HITS algorithm returns weights that are very close to the authority parameters. We formalize the findings of Azar et al. [6] in the framework of Borodin et al. [14]. We extend the definitions of stability and similarity for classes of random graphs, and we demonstrate the link between stability and similarity. We then prove that, with high probability, under some restrictive assumptions, the HITS algorithm is stable on the class of product graphs, and similar to the INDEGREE heuristic that ranks pages according to their indegree. This similarity result is the main contribution of this study. The implication of the result is that on product graphs, with high probability, the HITS algorithm reduces to simple indegree count. We show that our assumptions are general enough to capture graphs where the expected degrees follow a power law distribution as the one observed on the real Web. We also analyze the correlation between INDEGREE and HITS on a large sample of the Webgraph. The experimental analysis reveals that similarity between HITS and INDEGREE can also be observed on the real Web. We conclude with a discussion on the conditions that guarantee similarity of HITS and INDEGREE for the class of all possible graphs.
7.2 Preliminaries

**Link Analysis Ranking Algorithms**: Let $P$ be a collection of $n$ Web pages that need to be ranked. This collection may be the whole Web, or a query dependent subset of the Web. We construct the underlying hyperlink graph $G = (P, E)$ by creating a node for each Web page in the collection, and a directed edge for each hyperlink between two pages. The input to a LAR algorithm is the $n \times n$ adjacency matrix $W$ of the graph $G$. The output of the algorithm is an $n$-dimensional authority weight vector $w$, where $w_i$, the $i$-th coordinate of $w$, is the authority weight of node $i$.

We now describe the two LAR algorithms we consider in this Chapter: the INDEGREE algorithm, and the HITS algorithm. The INDEGREE algorithm is the simple heuristic that assigns to each node weight equal to the number of incoming links in the graph $G$. The HITS algorithm was proposed by Kleinberg [45] in the seminal paper that introduced the hubs and authorities paradigm. In this framework, every page can be thought of as having a hub and an authority weight. Let $h$ and $a$ denote the $n$-dimensional hub and authority weight vectors. Kleinberg proposed an iterative algorithm, termed HITS, for computing the vectors $h$ and $a$; the algorithm is essentially a power method computation of the principle eigenvectors of the matrices $WW^T$ and $W^TW$ respectively. These are the principal singular vectors of the matrix $W$. The HITS algorithm returns the vector $a$, the right singular vector of matrix $W$.

Independently from Kleinberg, Brin and Page developed the celebrated PAGERANK algorithm [16], which outputs the stationary distribution of a random walk on the Webgraph. The works of Kleinberg [45] and Brin and Page [16] were followed by numerous modifications and extensions (see [14] and references within). Of particular interest is the SALSA algorithm by Lempel and Moran [55], which performs a random walk that alternates between hubs and authorities.

**Theoretical study of LAR algorithms**: Borodin et al. [14], in the paper that introduced the theoretical framework for the analysis of LAR algorithms, considered various algorithms, including HITS, SALSA, INDEGREE, and variants of HITS defined in their paper. They proved that, on the class of all possible graphs, no pair of algorithms is similar, and only the INDEGREE algorithm is stable. They also defined the notion of rank stability and rank similarity, where they considered the ordinal rankings induced by the weight vectors. The same results carry over in this case. Their work was extended by Lempel and Moran [56], and Lee and Borodin [54]. The stability of HITS and PAGERANK has also been studied elsewhere [61][10].

**The product graph model**: Product graphs (also known as random graphs with given expected degrees) were first considered as a model for the Webgraph by Azar et al. [6]. The undirected case, where the $h_i = a_i$ and the edges are undirected, has been studied extensively [57][24][25][26]. The focus of these works is on the case where the parameters follow a power law distribution, as it is the case with most real-life networks.
CHAPTER 7. STABILITY AND SIMILARITY OF LINK ANALYSIS RANKING ALGORITHMS

7.3 The theoretical framework

In this section we review the definitions of Borodin et al. [14], and we extend them for classes of random graphs. Let \( G_n \) denote the set of all possible graphs of size \( n \). The size of a graph is the number of nodes in the graph. Let \( G_n \subseteq G_n \) denote a collection of graphs in \( G_n \). Following the work of Borodin et al. [14], we define a link analysis algorithm \( A \) as a function \( A: G_n \rightarrow \mathbb{R}^n \) that maps a graph \( G \in G_n \) to an \( n \)-dimensional real vector. The vector \( A(G) \) is the authority weight vector produced by the algorithm \( A \) on graph \( G \). The weight vector \( A(G) \) is normalized under some chosen norm \( L \), that is, the algorithm maps the graphs in \( G_n \) onto the unit \( L \)-sphere. Typically, the weights are normalized under some \( L^p \) norm. The \( L^p \) norm of a vector \( w \) is defined as \( \|w\|_p = \left( \sum_{i=1}^{n} |w_i|^p \right)^{1/p} \).

Distance measures: In order to compare the behavior of different algorithms, or the behavior of the same algorithm on different graphs, Borodin et al. [14] defined various distance measures between authority weight vectors. The distance functions we consider are defined using the \( L^q \) norm. The \( d_q \) distance between two weight vectors \( w_1, w_2 \) is defined as follows.

\[
d_q(w_1, w_2) = \min_{\gamma_1, \gamma_2 \geq 1} \| \gamma_1 w_1 - \gamma_2 w_2 \|_q
\]

The constants \( \gamma_1 \) and \( \gamma_2 \) serve the purpose of alleviating differences due to different normalization factors. When using distance \( d_q \) we will assume that the vectors are normalized in the \( L^q \) norm. In this study we consider mainly the \( d_2 \) distance measure. In the following lemma, we prove that the \( d_2(a, b) = \|a - b\| \), and thus the \( d_2 \) distance is a metric.

Lemma 7.3.1 Let \( a \) and \( b \) be two unit vectors in the \( L_2 \) norm. For the distance measure \( d_2 \), we have that

\[
d_2(a, b) = \min_{\gamma_1, \gamma_2 \geq 1} \| \gamma_1 a - \gamma_2 b \| = \|a - b\|.
\]

Proof. By definition of the \( d_2 \) distance measure for any two weight vectors \( a \) and \( b \), we have that \( d_2(a_1, a_2) \leq \|a - b\| \). We will now prove that \( d_2(a, b) \geq \|a - b\| \), which implies that \( d_2(a, b) = \|a - b\| \).

Borodin et al. [14] prove that for every distance measure \( d_p \), at least one of the constants \( \gamma_1, \gamma_2 \) should be equal to 1. Without loss of generality, assume that this is the constant that is associated with the vector \( a \). Therefore, we have that

\[
d_2(a, b) = \min_{\gamma \geq 1} ||a - \gamma b||
\]

Given two vectors \( a \) and \( b \), let \( a^T b \) denote the dot product of the two vectors, and \( \cos(a, b) \) denote the cosine of the angle of the vectors \( a \) and \( b \). We have that
7.3. THE THEORETICAL FRAMEWORK

\[ a^T b = ||a|| ||b|| \cos(a, b) \]. For two unit vectors \( a \) and \( b \) we have the following.

\[
\|a - b\|^2 = (a - b)^T (a - b) = a^T a + b^T b - 2a^T b
\]
\[
= \|a\|^2 + \|b\|^2 - 2\|a\|\|b\| \cos(a, b)
\]
\[
= 2 - 2 \cos(a, b)
\]

Also we have that

\[
\|a - \gamma b\|^2 = \|a\|^2 + \|\gamma b\|^2 - 2\|a\|\|\gamma b\| \cos(a, \gamma b)
\]
\[
= 1 + \gamma^2 - 2\gamma \cos(a, b)
\]
\[
\geq 2\gamma - 2\gamma \cos(a, b)
\]
\[
\geq \|a - b\|
\]

The last two inequalities follow from the fact that \( 1 + \gamma^2 \geq 2\gamma \), and \( \gamma \geq 1 \).

**Similarity:** Borodin et al. [14] give the following general definition of similarity for any distance function \( d \) and any normalization norm \( L \). In the following we define

\[
M_n(d, L) = \sup_{\|w_1\| = \|w_2\| = 1} d(w_1, w_2)
\]

to be the maximum distance between any two \( n \)-dimensional vectors with unit norm \( L = || \cdot || \).

**Definition 7.3.1** Algorithms \( A_1 \) and \( A_2 \) are \((L, d)\)-similar on the class \( \overline{G}_n \) if as \( n \to \infty \)

\[
\max_{G \in \overline{G}_n} d(A_1(G), A_2(G)) = o(M_n(d, L))
\]

Consider now the case that the class \( \overline{G}_n \) is a class of random graphs, generated according to some random process. That is, we define a probability space \( (\overline{G}_n, P) \), where \( P \) is a probability distribution over the class \( \overline{G}_n \). We extend the definition of similarity on the class \( \overline{G}_n \) as follows.

**Definition 7.3.2** Algorithms \( A_1 \) and \( A_2 \) are \((L, d)\)-similar with high probability on the class of random graphs \( \overline{G}_n \) if for a graph \( G \) drawn from \( \overline{G}_n \), as \( n \to \infty \)

\[
d(A_1(G), A_2(G)) = o(M_n(d, L))
\]

with probability \( 1 - o(1) \).

We note that when we consider \((L_q, d_q)\)-similarity we have that \( M_n(d_q, L_q) = \Theta(1) \). Furthermore, if the distance function \( d \) is a metric, or a near metric\(^1\), then the transitivity property holds. It is easy to show that if algorithms \( A_1 \) and \( A_2 \) are similar

\(^1\)A near metric is a distance function that is reflexive, and symmetric, and there exists a constant \( c \) independent of \( n \), such that for all \( k > 0 \), and all vectors \( u, w_1, w_2, \ldots, w_k, v \), \( d(u, v) \leq c(d(u, w_1) + d(w_1, w_2) + \cdots + d(w_k, v)) \).
CHAPTER 7. STABILITY AND SIMILARITY OF LINK ANALYSIS RANKING ALGORITHMS

(with high probability), and algorithms \( A_2 \) and \( A_3 \) are similar (with high probability), then algorithms \( A_1 \) and \( A_3 \) are also similar (with high probability).

**Stability:** Let \( \mathcal{G}_n \) be a class of graphs, and let \( G = (P, E) \) and \( G' = (P, E') \) be two graphs in \( \mathcal{G}_n \). The link distance \( d^\ell \) between graphs \( G \) and \( G' \) is defined as
\[
d^\ell(G, G') = |(E \cup E') \setminus (E \cap E')|
\]
That is, \( d^\ell(G, G') \) is the minimum number of links that we need to add and/or remove so as to change one graph into the other.

Given a class of graphs \( \mathcal{G}_n \), let \( \mathcal{C}_k(G) = \{ G' \in \mathcal{G}_n : d^\ell(G, G') \leq k \} \) denote the set of all graphs that have link distance at most \( k \) from graph \( G \). Borodin et al. [14] give the following generic definition of stability.

**Definition 7.3.3** An algorithm \( A \) is \((L, d)\)-stable on the class of graphs \( \mathcal{G}_n \) if for every fixed positive integer \( k \), we have as \( n \to \infty \)
\[
\max_{G \in \mathcal{G}_n} \max_{G' \in \mathcal{C}_k(G)} d(A(G), A(G')) = o(M_n(d, L))
\]

Given a class of random graphs \( \mathcal{G}_n \), we define stability with high probability as follows.

**Definition 7.3.4** An algorithm \( A \) is \((L, d)\)-stable with high probability on the class of random graphs \( \mathcal{G}_n \) if for every fixed positive integer \( k \), for a graph \( G \) drawn from \( \mathcal{G}_n \) we have as \( n \to \infty \)
\[
\max_{G' \in \mathcal{C}_k(G)} d(A(G), A(G')) = o(M_n(d, L))
\]
with probability \( 1 - o(1) \).

**Stability and Similarity:** The following lemma shows the connection between stability and similarity. The lemma is a generalization of a lemma by Borodin et al. [14].

**Lemma 7.3.2** Let \( d \) be a metric or near metric distance function, \( L \) a norm, and \( \mathcal{G}_n \) a class of random graphs. If algorithm \( A_1 \) is \((L, d)\)-stable with high probability on the class \( \mathcal{G}_n \), and algorithm \( A_2 \) is \((L, d)\)-similar to \( A_1 \) with high probability on the class \( \mathcal{G}_n \), then \( A_2 \) is \((L, d)\)-stable with high probability on the class \( \mathcal{G}_n \).

**Proof.** Let \( G \in \mathcal{G}_n \) be a graph drawn from the class \( \mathcal{G}_n \). Also let \( M = M_n(d, L) \). Since \( A_1 \) and \( A_2 \) are \((L, d)\)-similar with high probability on the class \( \mathcal{G}_n \), it follows that \( p_1 = Pr[d(A_2(G), A_1(G)) = \Omega(M)] = o(1) \). Furthermore, since \( A_1 \) is \((L, d)\)-stable with high probability on the class \( \mathcal{G}_n \), we have that \( p_2 = Pr[\max_{G' \in \mathcal{C}_k(G)} d(A_1(G), A_1(G')) = \Omega(M)] = o(1) \). Define graph \( G_1 \) as
\[
G_1 = \arg \max_{G' \in \mathcal{C}_k(G)} d(A_1(G), A_1(G'))
\]
7.4. STABILITY AND SIMILARITY ON THE PRODUCT GRAPHS

and graph $G_2$ as

$$G_2 = \arg \max_{G' \in \mathcal{G}_k(G)} d(A_2(G), A_2(G'))$$

. By definition of the graph $G_1$, we have that $d(A_1(G), A_1(G_2)) = \Omega(M)$, thus $p_3 = \Pr[d(A_1(G), A_1(G_2))] = o(1)$.

From the metric or near metric property of the function $d$, we have that

$$d(A_2(G), A_2(G_2)) \leq c \left( d(A_2(G), A_1(G)) + d(A_1(G), A_1(G_2)) + d(A_1(G_2), A_2(G_2)) \right).$$

Therefore, $\Pr[d(A_2(G), A_2(G_2))] = \Omega(M) \leq p_1 + p_2 + p_3 = o(1)$. Therefore, $A_2$ is $(L, d)$-stable with high probability.

7.4 Stability and similarity on the product graphs

The class of product graphs $\mathcal{G}_n^p(h, a)$ (or, for brevity, $\mathcal{G}_n^p$) is defined with two parameters $h$ and $a$, which are two $n$-dimensional real vectors, with $h_i$ and $a_i$ taking values in $[0, 1]$. These can be thought of as the latent hub and authority vectors. A link is generated from node $i$ to node $j$ with probability $h_i a_j$.

Let $G \in \mathcal{G}_n^p$, and let $W$ be the adjacency matrix of the graph $G$. The matrix $W$ can be written as $W = ha^T + R$, where $R$ is a random matrix, such that

$$R[i, j] = \begin{cases} -h_i a_j & \text{with probability } 1 - h_i a_j \\ 1 - h_i a_j & \text{with probability } h_i a_j \end{cases}$$

We refer to matrix $R$ as the rounding matrix, that rounds the entries of $M$ to 0 or 1. We can think of the matrix $W$ as a perturbation of the matrix $M = ha^T$ by the rounding matrix $R$. The matrix $M$ is a rank-one matrix. If we run HITS on the matrix $M$ (assuming a small modification of the algorithm so that it runs on weighted graphs), the algorithm will reconstruct the latent vectors $a$ and $h$, which are the singular vectors of matrix $M$. Note also that if we run the INDEGREE algorithm on the matrix $M$ (assuming again that we take the weighted indegrees), the algorithm will also output the latent vector $a$. So, on rank-one matrices the two algorithms are identical. The question is how the addition of the rounding matrix $R$ affects the output of the two algorithms. We will show that it has only a small effect, and the two algorithms remain similar.

More formally, let LATENT denote the (imaginary) LAR algorithm which, for any graph $G$ in the class $\mathcal{G}_n^p(h, a)$, outputs the vector $a$. We will show that both HITS and INDEGREE are similar to LATENT with high probability. This implies that the two algorithms are similar with high probability. Furthermore, we will show that it also implies the stability of the HITS algorithm.
7.4.1 Mathematical Tools

We now introduce some mathematical tools that we will use for the remaining of this section.

**Perturbation Theory:** Perturbation theory studies how adding a perturbation matrix $E$ to a matrix $M$ affects the eigenvalues and eigenvectors of $M$. Let $G$ and $G'$ be two graphs, and let $W$ and $W'$ denote the respective adjacency matrices. The matrix $W'$ can be written as $W' = W + E$, where $E$ is a matrix with entries in $\{-1, 0, 1\}$. The entry $E[i, j]$ is 1 if we add a link from $i$ to $j$, and $-1$ if we remove a link from $i$ to $j$. Therefore, we can think of the matrix $W'$ as a perturbation of the matrix $W$ by a matrix $E$. Note that if we assume that only a constant number of links is added and removed, then both the Frobenius and the $L_2$ norms of $E$ are bounded by a constant.

We now introduce an important lemma that we will use in the following.

**Lemma 7.4.1** Let $W$ be a matrix, and let $W + E$ be a perturbation of the matrix. Let $u$ and $v$ denote the left and right principal singular vectors of the matrix $W$, and $u'$ and $v'$ the principal singular vectors of the perturbed matrix. Let $\sigma_1, \sigma_2$ denote the first and second singular values of the matrix $W$. If $\sigma_1 - \sigma_2 = \omega(\|E\|_2)$, then $\|u' - u\|_2 = o(1)$ and $\|v' - v\|_2 = o(1)$.

For the proof of this lemma we use results from perturbation theory [66] to study how the principal singular vectors of a matrix $W$ change when we add the matrix $E$. The theorems that we use assume that both the matrix $W$ and the perturbation $E$ are symmetric, so instead of using the matrices $W$ and $E$ we will consider the matrices $B$ and $F$ which are defined as follows.

$$B = \begin{bmatrix} 0 & W^T \\ W & 0 \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} 0 & E^T \\ E & 0 \end{bmatrix} \tag{7.1}$$

If $\sigma_i$ is the $i$-th singular value of $W$, and $(u_i, v_i)$ is the corresponding pair of singular vectors, then the matrix $B$ has eigenvalues $\pm \sigma_i$, with eigenvectors $[v_i, u_i]^T$ for the eigenvalue $\sigma_i$, and $[v_i, -u_i]^T$ for the eigenvalue $-\sigma_i$. Therefore, instead of studying the perturbation of the singular values and vectors of matrix $W + E$, we will study the eigenvalues and eigenvectors of matrix $B + F$. Note also that $\|F\|_2 = \|E\|_2$, and that $\|F\|_F = \sqrt{2}\|E\|_F$.

We make use of the following theorem by Stewart (Theorem V.2.8 in [66] for the symmetric case).

**Theorem 7.4.2** Suppose $B$ and $B + F$ are $n$ by $n$ symmetric matrices and that 

$$Q = [q, Q_2]$$

is a unitary matrix, such that the vector $q$ is an eigenvector for the matrix $B$. Partition the matrices $Q^TBQ$ and $Q^TFQ$ as follows

$$Q^TBQ = \begin{bmatrix} \lambda & 0 \\ 0 & B_{22} \end{bmatrix} \quad \text{and} \quad Q^TFQ = \begin{bmatrix} f_{11} & f_{21}^T \\ f_{21} & F_{22} \end{bmatrix}$$
Let
\[ \delta = \min_{\mu \in \lambda(B_{22})} |\lambda - \mu| - |f_{11}| - \|F_{22}\|_2 \]
where \( \lambda(B_{22}) \) denotes the set of eigenvalues of \( B_{22} \). If \( \delta > 0 \), and \( \delta > 2\|f_{21}\|_2 \), then there exists a vector \( p \) such that
\[ \|p\|_2 < \frac{2\|f_{21}\|_2}{\delta} \]
and
\[ q' = q + Q_2p \]
is an eigenvector of the matrix \( B + F \). For the eigenvalue \( \lambda' \) that corresponds to the eigenvector \( q' \), we have that
\[ \lambda' = \lambda + f_{11} + f_{21}^T p \]
We now give the proof of Lemma 7.4.1.

**Proof.** In the following, we will argue that under condition \( \sigma_1 - \sigma_2 = \omega(\|E\|_2) \), perturbing matrix \( W \) by \( E \) causes only a small perturbation of the principal left and right singular vectors of \( W \). Moreover, we will prove that the perturbed singular vectors remain the principal singular vectors of \( W \) since the perturbation does not change the relative order of the first and the second singular values.

In Theorem 7.4.2, define matrices \( B \) and \( F \) as in the Equation (7.1). Now, set \( q = [u, v]^T \), where \( u \) and \( v \) are the left and right singular vectors of \( W \) respectively. We have that \( \lambda = \sigma_1 \). We have that
\[ \delta = \sigma_1 - \sigma_2 - |f_{11}| - \|F_{22}\|_2 \]
Note that \( f_{11} = q^TFq \), \( F_{22} = Q_2^TFQ_2 \), and \( f_{21} = Q_2^TFq \). Since \( \|AB\|_2 \leq \|A\|_2 \|B\|_2 \), and unitary matrices have \( L_2 \) norm 1, we have that \( |f_{11}| \leq \|F\|_2 \), \( \|F_{22}\|_2 \leq \|F\|_2 \), and \( \|f_{21}\|_2 \leq \|F\|_2 \).

Note that \( \|F\|_2 = \|E\|_2 \). If \( \sigma_1 - \sigma_2 = \omega(\|E\|_2) \), then \( \delta = \omega(\|E\|_2) \) and obviously \( \delta > 0 \) and \( \delta > 2\|f_{21}\|_2 \). Therefore, there exists a vector \( p \) with \( \|p\|_2 < \|f_{21}\|_2/\delta \), such that the vector
\[ q' = q + Q_2p \]
is an eigenvector of the matrix \( B + F \). We also have that \( \|p\| = o(1) \) since \( \|f_{21}\|_2 \leq \|E\|_2 \) and \( \delta = \omega(\|E\|_2) \).

The eigenvalue associated with the vector \( q' \) is \( \lambda' = \lambda + f_{11} + f_{21}^T p \). Therefore,
\[
|\lambda - \lambda'| = |f_{11} + f_{21}^T p| \leq |f_{11}| + \|f_{21}^T p\|_2 \\
\leq \|E\|_2 + o(\|E\|_2) = O(\|E\|_2)
\]
CHAPTER 7. STABILITY AND SIMILARITY OF LINK ANALYSIS RANKING

ALGORITHMS

The first and second inequalities follow from the well known property of the absolute value and the properties of the $L_2$ vector norm. The last inequality follows from the fact that $\|F_{\tilde{\omega}}^2\|_2 = O(\|E\|_2)$, and $\|p\|_2 = o(1)$.

Note that $\lambda = \sigma_1$ is the principal singular value of the matrix $W$. Let $\sigma'_i$ denote the $i$-th singular value of the matrix $W' = W + E$. We know that for any singular value $\sigma_i$, $|\sigma_i - \sigma'_i| \leq \|E\|_2$. We have that $|\sigma_1 - \sigma'_1| \leq \|E\|_2$ and $|\sigma_2 - \sigma'_2| \leq \|E\|_2$. We have assumed that $\sigma_1 - \sigma_2 = \omega(\|E\|_2)$. Therefore, it must be that $\sigma'_1 - \sigma'_2 = \omega(\|E\|_2)$. Since $|\lambda - \lambda'| = O(\|E\|_2)$, it follows that $\lambda' = \sigma'_1$. Thus, the vector $q'$ is the principal eigenvector of the matrix $B + F$, and $q' = [u', v']^T$, where $u'$ and $v'$ are the left and right singular vectors of $W'$. Since $\|Q_2 p\|_2 \leq \|p\|_2$, it follows that $\|q - q'\|_2 = o(1)$.

Therefore,

$$\|u' - v\|_2 = o(1) \quad \text{and} \quad \|u' - u\|_2 = o(1)$$

\[\square\]

Norms of random matrices: We also make use of the following theorem for concentration bounds on the $L_2$ norm of random symmetric matrices. We state the theorem as it appears in [11].

**Theorem 7.4.3** Given an $m \times n$ matrix $A$ and any $\epsilon > 0$, let $\hat{A}$ be any random matrix such that for all $i, j$: $E[\hat{A}_{ij}] = A_{ij}$, $\text{Var}(\hat{A}_{ij}) \leq \sigma^2$, and $|\hat{A}_{ij} - A_{ij}| \leq K$, where

$$K = \left(\frac{4\epsilon}{4 + 3\epsilon}\right)^3 \frac{\sigma \sqrt{m + n}}{\log^2(m + n)}$$

For any $\alpha > 0$, and $m + n \geq 20$, with probability at least $1 - (m + n)^{-\alpha^2}$,

$$\|\hat{A} - A\|_2 < (2 + \alpha + \epsilon)\sigma \sqrt{m + n}$$

Chernoff bounds: We will make use of standard Chernoff bounds. The following theorem can be found in the textbook of Motwani and Raghavan [59].

**Theorem 7.4.4** Let $X_1, X_2, \ldots, X_n$ be independent Poisson trials such that, for $1 \leq i \leq n$, $\text{Pr}[X_i = 1] = p_i$, where $0 \leq p_i \leq 1$. Let $X = \sum_{i=1}^n X_i$, $\mu = E[X] = \sum_{i=1}^n p_i$. Then, for $0 < \delta \leq 1$, we have that

$$\text{Pr}[X < (1 - \delta)\mu] < \exp(-\mu\delta^2/2) \quad (7.2)$$

$$\text{Pr}[X > (1 + \delta)\mu] < \exp(-\mu\delta^2/4) \quad (7.3)$$

7.4.2 Conditions for the stability of HITS

We first provide general conditions for the stability of the HITS algorithm. Let $\mathcal{G}_n^\omega$ denote the class of graphs with adjacency matrix $W$ that satisfies $\sigma_1(W) - \sigma_2(W) = \omega(1)$. The proof of the following theorem follows directly from Lemma 7.4.1 and the fact that the perturbation matrix $E$ has $L_2$ norm bounded by a constant.
Theorem 7.4.5  The HITS algorithm is $(L_2,d_2)$-stable on the class of graphs $G_n^a$.

Theorem 7.4.5 provides a sufficient condition for the stability of HITS on general graphs and it will be useful when considering stability on the class of product graphs. The class $G_n^a$ is actually a subset of the class defined by the result of Ng et al. 61]. Translating their result in the framework of Borodin et al. 14], they prove that the HITS algorithm is stable on the class of graphs with $\sigma_1(W)^2 - \sigma_2(W)^2 = \omega(\sqrt{d})$, where $d$ is the maximum outdegree.

7.4.3 Similarity of HITS and Latent

We now turn our attention to product graphs, and we prove that HITS and Latent are similar on this class. A result of similar spirit is shown in the work of Azar et al. 6]. We make the following assumption for the vectors $a$ and $h$.

Assumption 1  For the class $G_n^a(h,a)$, the latent vectors $a$ and $h$ satisfy $\|a\|_2\|h\|_2 = \omega(\sqrt{n})$.

As we show below, Assumption 1 places a direct lower bound on the principal singular value of the matrix $M = ha^T$. Also, let $A = \sum_{i=1}^n a_i$, denote the sum of the authority values, and let $H = \sum_{j=1}^n h_j$ the sum of the hub values. Since the values are positive, we have $A = \|a\|_1$ and $H = \|h\|_1$. The product $HA$ is equal to the expected number of edges in the graph. We have that $HA \geq \|a\|_2\|h\|_2$, thus, from Assumption 1 $HA = \omega(\sqrt{n})$. This implies that the graph is not too sparse.

Lemma 7.4.6  The algorithms HITS and Latent are $(L_2,d_2)$-similar with high probability on the class $G_n^a$, subject to Assumption 1.

Proof. The singular vectors of the matrix $M$ are the $L_2$ unit vectors $a_2 = a/\|a\|_2$ and $h_2 = h/\|h\|_2$. The matrix $M$ can be expressed as $M = h_2^T\|h\|_2\|a\|_2^2 a_2$. Therefore, the principal singular value of $M$ is $\sigma_1 = \|h\|_2\|a\|_2 = \omega(\sqrt{n})$. Since $M$ is rank-one, $\sigma_i = 0$, for all $i = 2,3,\ldots,n$. Therefore, for matrix $M$ we have that $\sigma_1 - \sigma_2 = \omega(\sqrt{n})$.

Matrix $R$ is a random matrix, where each entry is a independent random variable with mean 0, and maximum value and variance bounded by 1. Using Theorem 7.3.3 we observe that $K = 1$, and $\sigma = 1$. Setting $\epsilon = 1$ and $\alpha = 1$, we get that $Pr[\|R\|_2 \leq 8\sqrt{n}] \geq 1 - o(1/n)$, thus $\|R\|_2 = O(\sqrt{n})$ with high probability.

Therefore, we have that $\sigma_1 - \sigma_2 = \omega(\|R\|_2)$ with probability $1 - o(1)$. If $w_2$ is the right singular vector of matrix $W$ normalized in the $L_2$ norm, then, using Lemma 7.3.1 we have that $\|w_2 - a_2\|_2 = o(1)$ with probability $1 - o(1)$. 

Assumption 1 guarantees also the stability of HITS on $G_n^a$. The proof follows from the fact that if $G \in G_n^a$, then $G \in G_n^a$, with high probability.

Theorem 7.4.7  The HITS algorithm is $(L_2,d_2)$-stable with high probability on the class of graphs $G_n^a$, subject to Assumption 1.
7.4.4 Similarity of INDEGREE and LATENT

We now consider the \((L_q, d_q)\)-similarity of INDEGREE and LATENT, for all \(1 \leq q < \infty\). Again, let \(A = \sum_{i=1}^{n} a_i\), and let \(H = \sum_{j=1}^{n} h_j\). Also, let \(d\) denote the vector of the INDEGREE algorithm before any normalization is applied. That is, \(d_i\) is the indegree of node \(i\). For some node \(i\), we have that

\[
d_i = \sum_{j=1}^{n} W[j, i] = \sum_{j=1}^{n} M[j, i] + \sum_{j=1}^{n} R[j, i]
\]

We have that \(\sum_{j=1}^{n} M[j, i] = Ha_i\). Furthermore, let \(r_i = \sum_{j=1}^{n} R[j, i]\), and let \(r = [r_1, \ldots, r_n]^T\). Vector \(d\) can be expressed as \(d = Ha + r\).

We first prove the following auxiliary lemma.

Lemma 7.4.8 For every \(q \in [1, \infty)\), if \(H\|a\|_q = \omega(n^{1/q}\ln n)\), then \(\|r\|_q = o(H\|a\|_q)\) with high probability.

Proof. For the following we will use \(\| \cdot \|\) to denote the \(L_q\) norm, for some \(q \in [1, \infty)\). We will prove that \(\|r\|_q = o(H\|a\|_q)\) with probability at least \(1 - 1/n\).

We have assumed that \(H\|a\|_q = \omega(n^{1/q}\ln n)\), so it is sufficient to show that \(\|r\|_q = O(n^{1/q}\ln n)\), or equivalently that for all \(1 \leq i \leq n\), \(|r_i| = O(\ln n)\) with probability at least \(1 - 1/n^2\). Note that \(r_i = d_i - Ha_i\), so essentially we need to bound the deviation of \(d_i\) from its expectation.

We partition the nodes into two sets \(S\) and \(B\). Set \(S\) contains all nodes such that \(Ha_i = O(\ln n)\), that is, nodes with “small” expected indegree, and set \(B\) contains all nodes such that \(Ha_i = \omega(\ln n)\), that is, node with “big” expected indegree.

Consider a node \(i \in S\). We have that \(Ha_i \leq c\ln n\), for some constant \(c\). Using Equation 7.3, we set \(\delta = k\ln n/(Ha_i)\), where \(k\) is a constant such that \(k \geq \sqrt{8c}\), and we get that \(Pr[|d_i - Ha_i| \geq k\ln n] \leq \exp(-2\ln n)\). Therefore, for all nodes in \(S\) we have that \(|r_i| = O(\ln n)\) with probability at least \(1 - 1/n^2\). This implies that \(\sum_{i \in S} |r_i|^q = O(n\ln^q n) = o(H^q\|a\|^q)\), with probability \(1 - 1/n\).

Consider now a node \(i \in B\). We have that \(Ha_i = \omega(\ln n)\), thus, \(Ha_i = (\ln n)/s(n)\), where \(s(n)\) is a function such that \(s(n) = o(1)\). Using Equation 7.3, we set \(\delta = k\sqrt{s(n)}\), where \(k\) is a constant such that \(k \geq \sqrt{8}\), and we get that \(Pr[|d_i - Ha_i| \geq \delta Ha_i] \leq \exp(-2\ln n)\). Therefore, for the nodes in \(B\), we have that \(|r_i| = o(Ha_i)\) with probability at least \(1 - 1/n^2\). Thus, \(\sum_{i \in B} |r_i|^q = o(H^q\|a\|^q)\), with probability \(1 - 1/n\).

Putting everything together we have that \(\|r\|^q = \sum_{i \in S} |r_i|^q + \sum_{i \in B} |r_i|^q = o(H^q\|a\|^q)\), with probability \(1 - 2/n\). Therefore, \(\|r\| = o(H\|a\|)\) with probability \(1 - 2/n\). This concludes our proof.

We are now ready to prove the similarity of INDEGREE and LATENT. The following lemma follows from Lemma 7.4.8.
Lemma 7.4.9 For every \( q \in [1, \infty) \), the InDegree and Latent algorithms are \((L_q, d_q)\)-similar with high probability on the class \( G_n^q \), when the latent vectors \( a \) and \( h \) satisfy \( H\|a\|_q = \omega(n^{1/q} \ln n) \).

Proof. For the following we will use \( \| \cdot \| \) to denote the \( L_q \) norm, for some \( q \in [1, \infty) \). Let \( d_q \) and \( a_q \) denote the \( d \) and \( a \) vectors when normalized under the \( L_q \) norm. We will now bound the difference \( \| \gamma_1 a_q - \gamma_2 d_q \| \) for \( \gamma_1, \gamma_2 \geq 1 \).

First we observe that since \( d = Ha + r \), using norm properties, we can easily show that

\[
H\|a\| - \|r\| \leq \|d\| \leq H\|a\| + \|r\|
\]

Since we have that \( \|r\| = o(H\|a\|) \), it follows that \( \|d\| = \Theta(H\|a\|) \).

Now consider two cases. If \( \|d\| \geq H\|a\| \), then let \( \gamma_1 = 1 \) and \( \gamma_2 = \frac{\|d\|}{H\|a\|} \geq 1 \). We have that

\[
\|\gamma_1 a_q - \gamma_2 d_q\| = \left\| \frac{H\|a\|}{\|d\|} \frac{a}{\|a\|} - \frac{H\|a\|}{\|d\|} \frac{H a + r}{\|d\|} \right\| = \left\| \frac{r}{H\|a\|} \right\|
\]

If \( \|d\| \leq H\|a\| \), then let \( \gamma_1 = \frac{H\|a\|}{\|d\|} > 1 \) and \( \gamma_2 = 1 \). We have that

\[
\|\gamma_1 a_q - \gamma_2 d_q\| = \left\| \frac{H\|a\|}{\|d\|} \frac{a}{\|a\|} - \frac{H\|a\|}{\|d\|} \frac{H a + r}{\|d\|} \right\| \leq \frac{\|r\|}{\|d\|} \leq c \frac{\|r\|}{H\|a\|}
\]

for some constant \( c \), such that \( \|d\| \geq c H\|a\| \).

Therefore, we have that \( \|\gamma_1 a_q - \gamma_2 d_q\| \leq c \frac{\|r\|}{H\|a\|} \). When \( H\|a\| = \omega(n^{1/q} \ln n) \), we have that \( \|r\| = o(H\|a\|) \). Therefore \( \|\gamma_1 a_q - \gamma_2 d_q\| = o(1) \) which concludes the proof.

We now make the following assumption for vectors \( a \) and \( h \).

Assumption 2 For the class \( G_n^q(h, a) \), the latent vectors \( a \) and \( h \) satisfy \( H\|a\|_2 = \omega(\sqrt{n} \ln n) \).

Assumption 2 implies that the expected number of edges in the graph satisfies \( HA = \omega(\sqrt{n} \ln n) \). Note that we can satisfy Assumption 2 by requiring \( HA = \omega(n \ln n) \), that is, the graph is dense enough. We can satisfy both Assumption 1 and 2 by requiring that \( \sigma_1(M) = \|h\|_2 \|a\|_2 = \omega(\sqrt{n} \ln n) \).

The InDegree and Latent algorithms are \((L_2, d_2)\)-similar subject to Assumption 2. The following theorem follows from the transitivity property of similarity.

Theorem 7.4.10 The Hits and InDegree algorithms are \((L_2, d_2)\)-similar with high probability on the class \( G_n^q \), subject to Assumptions 1 and 2.
CHAPTER 7. STABILITY AND SIMILARITY OF LINK ANALYSIS RANKING ALGORITHMS

7.4.5 Power law graphs

A discrete random variable $X$ follows a power law distribution with parameter $\alpha$, if $Pr[X = x] \propto x^{-\alpha}$. Closely related to the power-law distribution is the Zipfian distribution, also known as Zipf’s law [74]. Zipf’s law states that the $r$-th largest value of the random variable $X$ is proportional to $r^{-\beta}$. It can be proved [2] that if $X$ follows a Zipfian distribution with exponent $\beta$, then it also follows a power law distribution with parameter $\alpha = 1 + 1/\beta$. We will now prove that Assumptions 1 and 2 are general enough to include graphs with expected in-degrees that follow Zipf’s law with parameter $\beta < 1$.

Without loss of generality we assume that $a_1 \geq a_2 \geq \cdots \geq a_n$. For some constant $c \leq 1$ the $i$-th authority value is defined as $a_i = ci^{-\beta}$, for $\beta < 1$. This implies a power law distribution on the expected in-degrees with exponent $\alpha > 2$. This is typical for most real-life graphs. The exponent of the in-degree distribution for the Webgraph is 2.1 [18]. For the hub values we assume that $h_i = \Theta(1)$, for all $1 \leq i \leq n$. Therefore, we have that $H = \Theta(n)$, and $\|h\|_2 = \Theta(\sqrt{n})$. Furthermore, it is easy to show that for $\beta < 1$, $\|a\|_2^2 = \sum_{i=1}^{n} \frac{c}{i^\beta} = \omega(1)$.

Therefore, $\|a\|_2^2\|h\|_2 = \omega(\sqrt{n})$, and $H\|a\|_2 = \omega(n)$, thus satisfying Assumptions 1 and 2. Therefore, we can conclude that HITS and INDEGREE are similar with high probability when the expected degrees follow a power law distribution. Note that on this graph we have that the expected number of edges is $HA = \omega(n \ln n)$.

7.5 Experimental analysis

In this section we study experimentally the similarity of HITS and INDEGREE on the WebBase sample [2] at Stanford. Figures 7.1(a) and 7.1(b) show the distributions of

\[\text{(a) INDEGREE} \quad \text{(b) HITS Authority}\]

Figure 7.1: INDEGREE and HITS Authority distributions on the Webgraph.

\[^{2}\text{http://www-diglib.stanford.edu/~testbed/doc2/WebBase/}\]
7.6. SIMILARITY OF HITS AND INDEGREE

the INDEGREE and HITS authority values. The indegree distribution, as it is well known, follows a power law distribution. The HITS authority weights also follow a “fat” power law distribution in the central part of the plot. Table 7.1 summarizes our findings on the relationship between INDEGREE and HITS. Since we only have a single graph and not a sequence of graphs, the distance measures are not very informative, so we also compute the correlation coefficient between the two weight vectors. We observe a strong correlation between the authority weights of HITS and the indegrees, while almost no correlation between the hub weights and the outdegrees. Similar trends are observed for the $d_2$ distance, where the distance between hub weights and outdegrees is much larger than that between authority weights and indegrees. These results suggest that although the Web, as expected, is not a product graph, the HITS authority weights can be well approximated by the indegrees.

<table>
<thead>
<tr>
<th></th>
<th>authority/indegree</th>
<th>hub/outdegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_2$ distance</td>
<td>0.36</td>
<td>1.23</td>
</tr>
<tr>
<td>correlation coefficient</td>
<td>0.93</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Table 7.1: Similarity between HITS and INDEGREE

7.6 Similarity of HITS and INDEGREE

In this section we study the general conditions under which the HITS and INDEGREE algorithms are similar. Consider a graph $G \in \mathcal{G}_n$ and the corresponding adjacency matrix $W$. Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ be the singular values of $W$, and let $a_1, \ldots, a_n$ and $h_1, \ldots, h_n$ denote the right (authority) and left (hub) singular vectors respectively. All vectors are unit vectors in the $L^2$ norm. The HITS algorithm outputs the vector $a = a_1$. Let $w$ denote the output of the INDEGREE algorithm (normalized in $L^2$). Also, let $H_i = \sum_{j=1}^n h_i(j)$ be the sum of the entries of the $i$-th hub vector. We prove the following proposition.

Proposition 7.6.1 For a graph $G \in \mathcal{G}_n$, the $d_2$ distance between HITS and INDEGREE is

$$d_2(a, w) = \sqrt{\frac{(\sigma_2 H_2)}{\sigma_1 H_1}^2 + \cdots + \frac{(\sigma_n H_n)}{\sigma_1 H_1}^2} \quad (7.4)$$

Proof. The adjacency matrix $W$ of graph $G$ can be decomposed as $W = \sigma_1 h_1 a_1^T + \cdots + \sigma_n h_n a_n^T$. Let $d$ denote the vector such that the $i$-th entry $d(i)$ of this vector is the in-degree of node $i$ (not normalized). We have that $d(i) = \sigma_1 H_1 a_1(i) + \cdots + \sigma_n H_n a_n(i)$, and $d = \sigma_1 H_1 a_1 + \cdots + \sigma_n H_n a_n$. Note that

$$\|d\|^2 = (\sigma_1 H_1 a_1 + \cdots + \sigma_n H_n a_n)(\sigma_1 H_1 a_1 + \cdots + \sigma_n H_n a_n)^T = \sigma_1^2 H_1^2 + \cdots + \sigma_n^2 H_n^2 \geq \sigma_1^2 H_1^2$$

where the last equation follows from the fact that $a_i^T a_i = 1$ and $a_i^T a_j = 0$. 

The output of INDEGREE is $w = d/\|d\|$, and the output of HITS is $a = a_1$. We are interested in bounding $\|a - \gamma w\|$, where $\gamma = \|d\|/\sigma_1 H_i \geq 1$. We have that

$$ \|a - \gamma w\|^2 = \left( \frac{\sigma_2 H_2}{\sigma_1 H_1} a_2 + \cdots + \frac{\sigma_n H_n}{\sigma_1 H_1} a_n \right)^2 $$

$$ = \left( \frac{\sigma_2 H_2}{\sigma_1 H_1} a_2 + \cdots + \frac{\sigma_n H_n}{\sigma_1 H_1} a_n \right)^T \left( \frac{\sigma_2 H_2}{\sigma_1 H_1} a_2 + \cdots + \frac{\sigma_n H_n}{\sigma_1 H_1} a_n \right) $$

Therefore,

$$ d_2(a, w) = \sqrt{\left( \frac{\sigma_2 H_2}{\sigma_1 H_1} \right)^2 + \cdots + \left( \frac{\sigma_n H_n}{\sigma_1 H_1} \right)^2} $$

We now study the conditions under which $d_2(a, w) = o(1)$. Since the values of $h_1$ are positive, we have that $H_1 = \|h_1\|_1$, and $1 \leq H_1 \leq \sqrt{n}$. For every $i > 1$, we have that $|H_i| \leq \|h_i\|_1$ and $|H_i| \leq \sqrt{n}$. The following conditions guarantee the similarity of HITS and INDEGREE: (a) $\sigma_2/\sigma_1 = o(1/\sqrt{n})$, and there exists a constant $k$ such that $\sigma_{k+1}/\sigma_1 = o(1/n)$; (b) $H_1 = \Theta(\sqrt{n})$, and $\sigma_2/\sigma_1 = o(1)$, and there exists a constant $k$ such that $\sigma_{k+1}/\sigma_1 = o(1/n)$; (c) $H_1 = \Theta(\sqrt{n})$, and $\sigma_2/\sigma_1 = o(1/\sqrt{n})$.

Assume now that $|H_i|/(\sigma_1 H_1) = o(1)$, for all $i \geq 2$. One possible way to obtain this bound is to assume that $\sigma_1 = \omega(\sqrt{n})$, or that $H_1 = \Theta(\sqrt{n})$ and $\sigma_1 = \omega(1)$. Then, we can obtain the following characterization of the distance between HITS and INDEGREE. From Equation (7.4) we have that $d_2(a, w) = o(\sqrt{\sigma_2^2 + \cdots + \sigma_n^2})$.

Let $W_1 = \sigma_1 h_1 a_1^T$ denote the rank-one approximation of $W$. The matrix $R = W - W_1$ is called the residual matrix, and it has singular values $\sigma_2, \ldots, \sigma_n$. We have that

$$ d_2(a, w) = o(\|W - W_1\|_F) \quad \text{and} \quad d_2(a, w) = o\left( \sqrt{\|W\|_F^2 - \|W_1\|_F^2} \right) $$

Equation (7.5) says that the similarity of HITS and INDEGREE algorithms depends on the Frobenius norm of the residual matrix. Furthermore, the similarity of the HITS and INDEGREE algorithms depends on the difference between the Frobenius and the spectral ($L_2$) norm of matrix $W$. The $L_2$ norm measures the strength of the strongest linear trend in the matrix, while the Frobenius norm captures the sum of the strengths of all linear trends in the matrix. The similarity of the HITS and INDEGREE algorithms depends upon the contribution of the strongest linear trend to the sum of linear trends.
7.7 Conclusions

In this chapter we studied the behavior of the HITS algorithm on the class of product graphs. We proved that under some assumptions the HITS algorithm is stable, and it is similar to the INDEGREE algorithm. Our assumptions include graphs with expected degrees that follow a power law distribution.

Our work opens a number of interesting directions for future work. First, it would be interesting to determine a necessary condition for the stability of the HITS algorithm. Also, it would be interesting to study the stability and similarity of other LAR algorithms on product graphs, such as the PAGERANK and the SALSA algorithms. Finally, it would be interesting to study other classes of random graphs \[9, 49\].
Chapter 8

Conclusions

In this thesis we addressed some of the open issues concerning the Web and its structure. Far from answering all questions posed by this complex, continuously expanding object, we gave some contributions to the characterization of its structure. In doing this, we also analyzed and developed effective and efficient techniques for the study of its properties. The importance of the contributions goes "in crescendo" with the chapters. The first three chapters present "the technical foundations" we develop in order to perform all the measurements. The main contributions of this work can be summarized in the following:

**Measures of the main properties on more recent and wider crawls.** Our extensive study allowed to confirm the ubiquitous presence of power law in all the measures, as indegree, outdegree, PageRank, and the very low correlation between indegree and PageRank.

**Algorithmic technologies and methodologies.** In order to measure the properties outlined by previous papers and a number of new ones, we developed a software library able to work in secondary memory. The implementation of such a library required the design of efficient, external memory algorithms. Moreover, we applied data stream techniques for the approximate computation of the indegree distribution and the number of bipartite cliques. We plan to extend these techniques to the computation of other properties of the Webgraph, such as the number of triangles.

**A fine-grain picture of the Webgraph.** The measures we accomplished allowed us to evidence the inner structure of the bow-tie. We discovered that the IN and the OUT sets are shallow and highly fragmented whereas the CORE is well-connected. These features led us to present a new abstraction of the Web: the daisy structure, a dense center with a large number of small petals. This outcome is not only a fascinating result by itself but it has a number of practical implications in devising better crawling strategies and in improving of performance of search engines.
A temporal analysis of the Web. A dynamic analysis of the Web is an hard task first of all for the lack of temporal data, i.e. sequential snapshots of the same set of Web pages along a significant interval of time. We limited this study to the Wikipedia encyclopaedia, where it is possible to derive snapshots of different subsets. This analysis focused on the evolution of the topological and statistical properties of Wikigraphs over the time and led to several observations on the update frequency of the articles of Wikipidia. We devised a power law distribution on the number of vertices receiving a given number of updates and found out that the number of updates of a page is not correlated with the PageRank, indegree, outdegree and the number of visits of that page.

A theoretical study of LAR algorithms. The stability is a favorable feature for algorithms devised to operate on a highly dynamic object as the Web. The results we presented in this thesis are only an initial step towards the study of stability and similarity of Link Analysis Ranking (LAR) algorithms on the class of random graphs. Our study was limited to the behaviour of HITS on the class of product graphs, nevertheless we obtained some interesting results: we outlined the sufficient conditions for its stability (it remains an open questions to determine a necessary condition) and proved the similarity between HITS and INDEGREE.
Bibliography


[57] MIHAIL, M., AND PAPADIMITRIOU, C. On the eigenvalue power law.

[58] MITZNMACHER, M. A brief history of generative models for power law and lognormal distributions. Internet Mathematics 1, 2 (2003).


Università La Sapienza  
Dottorato di Ricerca in Ingegneria Informatica  

Collection of Theses


VIII-96-1 Luca Cabibbo. *Querying and Updating Complex-Object Databases.* May 1996.


XII-00-1 Marco Daniele. Advances in Planning as Model Checking. April 2000.


XII-00-3 Leonardo Tininini. Querying Aggregate Data. April 2000.


