

# The Web as a graph: how far we are.

DEBORA DONATO, LUIGI LAURA, STEFANO LEONARDI and STEFANO MILLOZZI

---

In this paper we present an experimental study of the properties of webgraphs. We study a large crawl from 2001 of 200M pages and about 1.4 billion edges, made available by the WebBase project at Stanford, as well as several synthetic ones generated according to various models proposed recently. We investigate several topological properties of such graphs, including the number of bipartite cores and strongly connected components, the distribution of degrees and PageRank values and some correlations; we present a comparison study of the models against these measures.

Our findings are that (i) the WebBase sample differs slightly from the (older) samples studied in the literature, and (ii) despite the fact that these models do not catch all its properties, they do exhibit some peculiar behaviors not found, for example, in the models from classical random graph theory.

Moreover we developed a software library able to generate and measure massive graphs in secondary memory; this library is publicly available under the GPL licence. We discuss its implementation and some computational issues related to secondary memory graph algorithms.

Categories and Subject Descriptors: H [3]: 5

General Terms: Experimentation, Measurement, Theory, Verification

Additional Key Words and Phrases: Graph structure, models, World-Wide-Web

---

## 1. INTRODUCTION

A first extensive study of “the Web as a graph” appeared, in 1999, in the work of Kleinberg et al.; here the authors, for the first time, explicitly focused on the directed graph induced by the hyperlink structure of the World Wide Web. Since then the term *Webgraph* addresses the graph whose nodes are the (static) html pages and edges are the (directed) hyperlinks among them.

The study of the Webgraph has been the subject of a large interest in the scientific community for several important reasons. The information provided by the Webgraph is for instance at the basis of link analysis algorithms for ranking Web documents, PageRank [Brin and Page 1998] and HITS [Kleinberg 1997] to mention the most popular such examples. In all these algorithms the insertion of a hyperlink between two documents is seen as the endorsement of authority from the first to the second document. The study of the

---

Authors' address: Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza”, Via Salaria 113, 00198 Roma Italy; email: {donato, laura, leon, millozzi}@dis.uniroma1.it Partially supported by the Future and Emerging Technologies programme of the EU under contracts number IST-2001-33555 COSIN “Co-evolution and Self-organization in Dynamical Networks” and IST-1999-14186 ALCOM-FT “Algorithms and Complexity in Future Technologies”, and by the Italian research project ALINWEB: “Algorithmica per Internet e per il Web”, MIUR – Programmi di Ricerca di Rilevante Interesse Nazionale.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 \$5.00

Web is also attracting 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.

The basic step to unearth the topological structure of the Webgraph is to collect the hyperlinked structure of large crawls spanning a good share of the whole Web. A first line of research has then focussed on performing meaningful measures to draw relevant conclusions on the statistical and topological properties of the Webgraph. This approach assumes that large samples of the Web will faithfully reproduce its properties, and this assumption has been supported by the work of Dill et al. [2001], that showed that no matter which criterion is used to sample the Web, all the samples exhibit the same properties.

A second important research line has concentrated on the development of models of the evolution of the Webgraph, able to simulate the emerging of global properties from the microscopic activity of link creation. Developing a realistic and accurate stochastic graph model for the Webgraph is also relevant for many practical purposes:

- Devising web applications, since many problems that are computationally difficult for general graphs could be considerably easier on graphs arising from specific stochastic processes.
- Testing web applications on synthetic benchmarks of small dimension.
- Detecting peculiar regions of the Webgraph, i.e. , local subsets that have different statistical properties with respect to the whole structure.
- Predicting the evolution of new phenomena in the Web.
- Dealing more efficiently with large scale computations (e.g. , by recognizing the possibility of compressing a graph generated according to such model [Adler and Mitzenmacher ; Boldi and Vigna 2004]).

The study of the topological structure of networks of several billion edges also poses new computational problems. We expect this branch of applied algorithmics to grow quickly in the next years [Witten et al. 1999; Abello et al. 2002] due to the increasing number of applications dealing with the mining of massive data. A third line of research has then focussed on the development of computational tools to manipulate graphs and perform web-related measures at very large scale.

Some of the experimental results we describe have been anticipated in [Donato et al. 2004a; 2004b]; in this paper we present a comprehensive summary of our work on the Webgraph.

This paper is organised as follows: in the following section we recall some basic definitions. In Section 3 the analysis of the WebBase sample is presented. In Section 4 we illustrate the results of a comparison study over several random webgraph models. Section 5 deals with the computational issues of generating and measuring massive webgraphs in secondary memory. We conclude (Section 6) with final remarks and comments.

## 2. 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*.

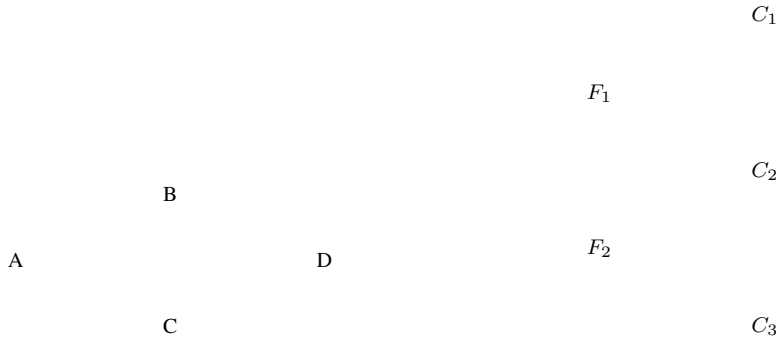


Fig. 1. A simple directed graph.

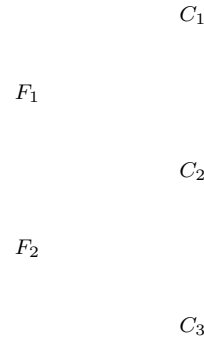


Fig. 2. A (2,3)-bipartite clique.

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

In a directed graph the *in-degree* (*out-degree*) of a node is the number of incoming (outgoing) edges. For example, if we refer to the simple digraph shown in Figure 1, the in-degree of vertex  $D$  is 2 (it is linked from  $B$  and  $C$ ) while its out-degree is 0 (it has no outgoing edges). Furthermore, we call *successors* of a node  $v$  all the nodes pointed by  $v$ , and *predecessors* of a node  $v$  all the nodes that point to  $v$ .

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: 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  $v_0, e_1, v_1, e_2, \dots, v_{n-1}, e_{n-1}, v_n$  such that each edge is incident to 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 orientation.

A *connected component* of an undirected graph  $G$  is a subset of nodes  $S$  such that for every pair of vertices  $u, v \in S$ ,  $v$  is reachable from  $u$  (i.e. there 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 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  $G$  that is obtained by removing the orientation of the edges in  $G$ .

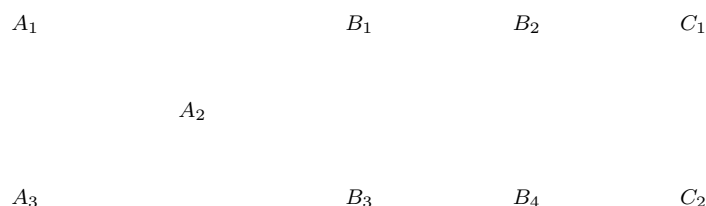


Fig. 3. An example of strongly connected components of a graph.

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 introductions to graph theory are the classical work of Harary [1969] and the more recent book of Diestel [1997].

**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 [Mitzenmacher 2003].

**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 [Cho and Garcia-Molina 2002] and [Boldi et al. 2002].

**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 in 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

Fig. 4. In-degree distribution of the WebBase crawl.

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 explicitly 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 [Vitter and Shriver 1994a; 1994b].

### 3. WEBBASE

In this section we present the results of our experiments, conducted on a 200M node crawl collected from the WebBase project at Stanford [webbase ] in 2001. The repository makes several crawls available to researchers; details about the crawler, including its implementation and its performance, can be found in [Cho et al. 2006]. 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 Gulli and Signorini [2005] in 2005 the size of the Web is more than 11.5 billion webpages. An older study done by cyvellance [] 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 million webpages per day, and this rate is consistent with the result of Gulli and Signorini [2005].

#### 3.1 In-degree and out-degree

Since the very first analysis, the Webgraph has shown the ubiquitous presence of power law distributions, a typical signature of scale-free properties. Barabasi and Albert [1999] and Kumar et al. [1999] suggested that the in-degree of the Webgraph follows a *power law* distribution. Later experiments by Broder et al. [2000] on a crawl of 200M pages from 1999 by Altavista confirmed it as a basic property: the probability that the in-degree of a

Fig. 5. Out-degree distribution of the WebBase crawl.

vertex is  $i$  is distributed as  $Pr_u[\text{in-degree}(u)=i] \propto 1/i^\gamma$ , for  $\gamma \approx 2.1$ . In [Broder et al. 2000] the out-degree 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 equal to about 7 times the number of vertices.

The in-degree distribution, shown in Figure 4, follows a power law with  $\gamma = 2.1$ . This confirms the observations done on the crawl of 1997 from Alexa [Kumar et al. 1999], the crawl of 1999 from Altavista [Broder et al. 2000] and the notredame.edu domain [Barabasi and Albert 1999].

We note a bump between the values 1,000 and 10,000, that has also been observed by [Broder et al. 2000] 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.

Figure 5 shows the out-degree distribution of the WebBase crawl. While the in-degree distribution is fitted with a power law, the out-degree 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 [Broder et al. 2000]. A possible explanation of this phenomenon is that writing a scale-free series of hyperlinks is seriously limited by the patience of webmasters. Another explanation is that the deviation is an artifact of the crawl policy: large sites and newly discovered sites usually get less coverage and seem smaller than they really are.

### 3.2 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. Brin and Page [1998] extend this idea further by observing that links from pages of high quality should confer more authority. It is not only important which

Fig. 6. PageRank distribution of the WebBase crawl.

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-to by many pages of high quality. We discuss the PageRank algorithm in Section 5, where we detail our implementation.

The correlation<sup>1</sup> between the distribution of PageRank and in-degree has recently been studied in a work of Pandurangan, Raghavan and Upfal [Pandurangan et al. 2002]. 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 matches exactly the in-degree distribution, but surprisingly very little correlation between these quantities is observed, i.e., pages with high in-degree may have low PageRank [Raghavan 2002].

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 correlation between PageRank and in-degree. We obtained a value of 0.3097, on a range of variation in  $[-1, 1]$  from negative to positive correlation. This result confirms the weak correlation between PageRank and in-degree values.

### 3.3 Bipartite cliques

A surprising number of specific topological structures such as bipartite cliques of relatively small size has been observed in [Kumar et al. 1999] 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-to by all the fans. Over 100,000 such communities have been recognized [Kumar et al. 1999] on a snapshot of 200M pages taken by Alexa in 1997.

In Figure 7 is shown the distribution of the number of bipartite cliques  $(i, j)$ , with

<sup>1</sup>More precisely, with the term correlation we refer to the Pearson's correlation coefficient.

Fig. 7. Number of bipartite cores in the WebBase crawl.

$i, j = 1, \dots, 10$ . Note that the plot refers to the number of bipartite cliques of the pruned sample: since the objective is to detect cores of hidden communities, all the vertices with high degree are removed. In our case we removed all the vertices with in-degree (or out-degree) bigger than 50. 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 by 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. Another possible explanation is, again, the crawl policy. We also recall that the longevity of cyber-communities' websites is bigger as compared to other websites [Kumar et al. 1999]. A second possible explanation is that our algorithm for finding disjoint bipartite cliques, which is detailed in Section 5, is more efficient than the one implemented in [Kumar et al. 1999].

### 3.4 Strongly connected components

Broder et al. [2000] 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 I 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.

Figure 8 shows the global scc distribution of the Webbase sample and of its different

	SCC	IN	OUT	TENDR.	DISC.
Altavista (1999) [Broder et al. 2000]	28%	21%	21%	22%	9%
WebBase (2001)	33%	11%	39%	13%	4%

Table I. Size of regions in both Altavista and WebBase crawl

Fig. 8. SCC distribution of the Web Base crawl.

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 in-degree and the PageRank distributions.

#### 4. STOCHASTIC MODELS OF THE WEBGRAPH

As we said in the introduction, the topological properties observed in the Webgraph, such as the ones seen in the previous section, cannot be found in the traditional random graph model of Erdős and Rényi (ER) [1960]. Kleinberg et al. [1999], that first explicitly considered the Web as a graph, listed a number of *desiderata* for a Webgraph model:

- (1) It should have a succinct and fairly natural description.
- (2) It should be rooted in a plausible macro-level process for the creation of content of the Web.
- (3) It should not require some a priori static set of “topics”.
- (4) It should reflect many of the structural phenomena observed in the Web.

Several models appeared recently in the literature. A first comparison study has been presented by Laura et al. [2002], and the authors showed that three models, the Evolving, the Copying and the Multi-Layer were outstanding. These models show some common traits: they are *evolving*, in the sense that nodes (webpages) are added to the graph one

after the other; they have “natural” laws that rule the choice of which page to link to, as we see in the following; they reflect many of the structural phenomena of the Webgraph. In the following sections we start by addressing the main features of the Evolving, the Copying and the more recent Pagerank models; then we detail a “revised” version of the Multi-Layer model [Laura et al. 2002], and we conclude by showing the results of our experimental comparison of these models.

#### 4.1 Models of the Webgraph

Barabasi and Albert [1999] started the study of evolving networks by presenting a model in which at every discrete time step a new vertex is inserted in the graph. The new vertex connects to a constant number of previously inserted vertices chosen according to the *preferential attachment* rule, i.e. with probability proportional to the in-degree. This model shows a power law distribution over the in-degree of the vertices with exponent roughly 2 when the number of edges that connect every vertex to the graph is 7. In the following sections we refer to this model as the *Evolving* model.

The Copying model has been later proposed by Kumar et al. [2000] to explain other relevant properties observed in the Webgraph. For every new vertex entering the graph a prototype vertex  $p$  it is selected at random. A constant number  $d$  of links connect the new vertex to previously inserted vertices. The model is parameterized on a *copying factor*  $\alpha$ . The end-point of a link is either copied with probability  $\alpha$  from a link of the prototype vertex  $p$ , or it is selected at random with probability  $1 - \alpha$ . The copying event aims to model the formation of a large number of bipartite cliques in the Webgraph. The authors present two versions of the models with different growth factor: the *linear* model in which the graph grows by some absolute amount (usually one node) at every step and an *exponential* model in which the graph grows by an amount that is a function of its actual size. In our experimental study we consider the *linear* version, and we refer to it simply as the *Copying* model. The model has been analytically studied and showed to yield power law distributions on both the in-degree and the number of disjoint bipartite cliques for specific values of  $\alpha$ . In particular, the in-degree is distributed with a power law with exponent 2.1 when  $\alpha = 0.8$ .

More recently Pandurangan et al. [2002] proposed a model that complements the Evolving model by choosing the endpoint of a link with probability proportional to the in-degree and to the PageRank of a vertex. There are two parameters  $\alpha, \beta \in [0, 1]$  such that  $\alpha + \beta \leq 1$ . With probability  $\alpha$  a node is chosen as the end-point of the  $l$ th edge with probability proportional to its in-degree (preferential attachment), with probability  $\beta$  it is chosen with probability proportional to its PageRank value, and with probability  $1 - \alpha - \beta$  at random (uniform probability). The authors show by computer simulation that with an appropriate tuning of the parameters the generated graphs capture the distributional properties of both PageRank and in-degree. We refer to this model as the *PageRank* model.

#### 4.2 A Multi-Layer model

An interesting point of view on the structure of the Webgraph has been presented in the work of Dill et al. [2001]. Here the authors explain how the Web shows a fractal structure in many different ways. The Webgraph can be viewed as the outcome of a number of similar and independent stochastic processes. At various scales we have that there are “cohesive collections” of web pages (for example pages on a site, or pages about a topic) and these collections are structurally similar to the whole Web. The central regions of

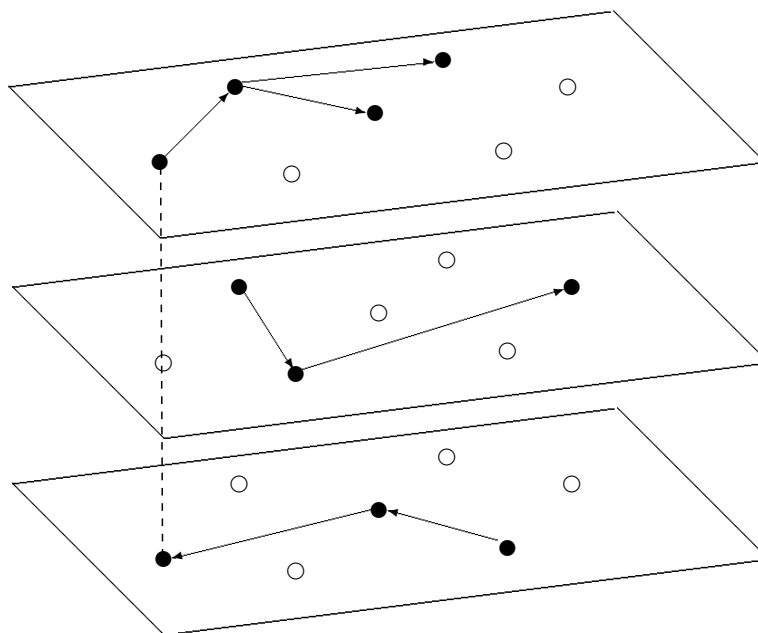


Fig. 9. A Multi-Layer view of a graph. The leftmost node is active on the top and bottom layer, and not active in the middle layer. Note that, on each layer, links are between active (black) nodes.

such collections are called “Thematically Unified Clusters” (TUCs) and they provide a navigational backbone of the Web. In a different work, Pennock et al. [2002] argue that the Web is the sum of stochastic independent processes that share a common (fractal) structure, but this structure sometimes can vary. Indeed they provide examples where the distributions exhibit large deviation from power laws.

Motivated by the above works, Laura et al. [2002] proposed a *Multi-Layer* model of the Webgraph in which every new page that enters into the graph (i) is assigned to a number of regions (layers) it belongs to and (ii) it is allowed to link only to vertices of these regions. Inside each region, the links are chosen according to some graph model, i.e. the Evolving, the Copying or a combination of both. This model showed some nice properties such as a distribution of the in-degree with a power law of exponent 2.1 for a wide range of variation of the parameters.

In the following we detail the *Multi-Layer* model in both its extra-layer behavior, i.e. how the nodes belong to the layers, and the intra-layer behavior, i.e. how nodes are connected in a single layer.

**Extra-layer.** The model evolves at discrete time steps. The graph will be formed by the union of  $L$  regions, also denoted as *layers*. At each time step  $t$  a new page  $x$  enters the graph and it is assigned to a fixed number  $l$  of regions; then the page connects to a total number of  $d$  pages that belong to its regions.

Let  $X_i(t)$  be the number of pages assigned to region  $i$  at time  $t$ . Let  $L(x)$  be the set of regions which the page  $x$  belongs to.

We repeat  $l$  times the following random choice:

—  $L(x) = L(x) \cup \{i\}$ , where region  $i$  is chosen in  $L \setminus L(x)$  with probability proportional

to  $X_i(t)$  with a suitable normalization factor.

The stochastic process above clearly defines a Zipf's distribution over the size of the population of the regions, i.e., the value  $X_i(t)$ .

The  $d$  edges are evenly distributed (up to 1) between the  $l$  regions. Let  $c = \lfloor d/l \rfloor$  and  $\alpha$  be the copying factor. Consider each region  $i$  to which vertex  $x$  is assigned. Vertex  $x$  will be connected by  $c$  or  $c + 1$  edges to other vertices of region  $i$ . Denote by  $\mathcal{X}$  the set of  $X_i(t)$  vertices assigned to region  $i$  not later than time  $t$ . The layer  $i$  graph, denoted by  $G_i(t)$ , is formed by the vertices of  $\mathcal{X}$  and by the edges inserted not later than time  $t$  between edges of  $\mathcal{X}$ .

So once we assign pages to layers, every layer is generated independently (in particular, when we deal with preferential attachment, only the edges that connect vertices in the same layer are considered). This completely defines the extra-layer behaviour (how nodes belong to layers).

**Intra-layer.** We considered several intra-layer behaviours (how nodes are connected in a single layer): in particular, in the next section we measure the following Multi-Layer models where the single layer is generated according to:

- LCE: this is the original Multi-Layer model presented in [Laura et al. 2002]; every layer is a hybrid between the *Evolving* model and the *Copying* model: a link is either copied from a prototype vertex (as in the *Copying* model) or chosen according to the preferential attachment rule, as in the *Evolving* model.
- LE: the *Evolving* model is used in each layer.
- LC: each layer is generated according to the *Copying* model.

### 4.3 Large Scale Simulation

In this section we present the experimental results on the four stochastic graph models we discussed in the previous sections: the *Evolving*, *Copying*, PageRank and Multi-Layer models.

We will compare these models on the following measures:

- (1) in-degree;
- (2) bipartite cliques;
- (3) size of the largest SCC;
- (4) PageRank distribution;
- (5) PageRank/in-degree correlation;
- (6) in-degree/Average in-degree of predecessors correlation;

All the synthetic models we study do not contain cycles. Therefore we added a number of edges whose endpoints are chosen at random. In particular, we introduce random edges for a number equal to 50%, 100% and 200% of the number of vertices in the graph.

All simulation have been carried out on graphs of 1 M vertices and average degree of each vertex equal to 7. We consider the variant of the *Copying Linear* model with a copying factor  $\alpha = 0.8$ , the value for which the in-degree distribution meets a power law with slope 2.1, the one observed in the real Web. The *Copying Linear* model with copying factor smaller than  $\alpha = 0.5$  does not follow a power law distribution.

Table II summarizes our main experimental findings; note that in the last line we added, for comparison, also the measures of the WebBase sample.

Fig. 10. Number and size of SCCs – (Evolving Network Model)

All simulated models, with the exception of the copying model, do not exhibit any clique. This can be explained from an intuitive point of view because it is very unlikely that, after pruning, two nodes with a small in-degree share a subset of predecessors.

We also observed that in all models the largest SCC spans almost the whole graph when the number of edges that are added to the graph is about 200% of the number of vertices. PageRank and in-degree are positively correlated in all models. Moreover, PageRank always follows a power law distribution with slope similar to that of the in-degree.

The in-degree of a vertex and the average in-degree of its predecessors are weakly correlated in all models with the exception of Multi-Layer-LC. In this version the in-degree distribution does not follow a power law.

Multi-Layer-LCE and Multi-Layer-LE present several features that can also be found in real samples of the Web. In-degree and PageRank follow a power law with slope 2.1 for a wide range of the parameters, such as the copying factor and the number of layers. Differently from the real Web the graphs generated according to this model do not contain small cliques and have in-degree and PageRank highly correlated.

We emphasize that the prominent aspect of the Multi-Layer model is a relative independence from the generating model used in each single layer (and its parameters): we see that many versions of the Multi-Layer exhibit properties resembling the ones of real samples of the Web. It seems therefore that summing up various stochastic processes creates a graph whose characteristics do not depend strongly on the underlying processes. This can confirm on an experimental base the observation made by Dill et al. [2001] that the Web is the sum of independent stochastic processes.

The most remarkable observation for all these stochastic models is that we do not observe any threshold phenomenon in the emerging of a large SCC. We observe that the size of the largest SCC increases smoothly with the number of edges that are rewired up to span a big part of the graph. We also observe that the number of SCCs decreases smoothly with the increase of the percentage of rewired edges. This can be observed in Figure 10 for the Evolving model on a graph of 10M vertices. We therefore conclude that stochastic

model	number of added edges	$\gamma$	number of cliques (4,3)	max scc size	max scc percent	$\pi$	correlation indeg - PR
Evolving	0	2	0	1	0,0001	2,2	0,935804612
	0.5M	2	0	404907	40,4907	2,2	0,941112952
	1M	2	0	692714	69,2714	2,2	0,948764657
	2M	2	0	910955	91,0955	2,2	0,84754201
Copying	0	2,1	2673	1	0,0001	2,1	0,650691063
	0.5M	2,1	2661	647942	64,7942	2,2	0,738855883
	1M	2,1	2647	824165	82,4165	2,3	0,764060435
	2M	2,1	2628	947255	94,7255	2,4	0,798681635
PageRank	0	3	0	1	0,0001	2,1	0,1977538252
	0.5M	3	0	814.552	81,4552	2,1	0,2392437130
	1M	3	0	908605	90,8605	2,1	0,2917959368
	2M	3	0	971854	97,1854	2,1	0,3299212975
multiLayer Copying (LC)	0M	No	0	1	0,0001	2,1	0,141732415
	0.5M	No	0	397645	39,7645	2,2	0,317389402
	1M	No	0	680906	68,0906	2,2	0,501913657
	2M	No	0	901211	90,1211	2,23	0,708833201
multiLayer Copy+Evol (LCB)	0	2,1	0	1	0,0001	2,1	0,345893784
	0.5M	2,1	0	298801	29,8801	2,1	0,390800789
	1M	2,1	0	620923	62,0923	2,05	0,531627042
	2M	2,1	0	883783	88,3783	2	0,692679365
multiLayer Evolving (LB)	0	2,1	0	1	0,0001	2,1	0,679171342
	0.5M	2,1	0	657848	65,7848	2,1	0,67020433
	1M	2,1	0	829717	82,9717	2,1	0,69044359
	2M	2,1	0	948958	94,8958	2,1	0,745121517
<b>WebBase crawl</b>	-	2,1	2949486	44713193	32,9	2,1	0.3076062794

Table II. Summary of measures of webgraph models

models for power law graphs behave very differently from the Erdős-Renyi model [Erdős and R. 1960] where the emerging of a giant connected component is observed when the number of edges grows over a threshold that is slightly more than linear in the number of vertices. This can be intuitively explained with the presence in the graph of a number of vertices of high degree that form the skeleton of the strongly connected component but this phenomenon certainly needs a deeper understanding. A similar conclusion is also formally obtained for undirected graphs in a recent paper of B. Bollobas [2003], where they study the LCD model, that is a variation of the Evolving model.

## 5. ALGORITHMIC TECHNIQUES FOR GENERATING AND MEASURING WEBGRAPHS

In this section 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. 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.

- Semi-external random graph generators that create webgraphs according to the random models described in Section 4 (Copying, Evolving, Multi-Layer and PageRank).
- 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 for each node in the graph. This traversal has been developed following the approach suggested by [Sibeyn et al. 2002].
- 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 algorithms listed above are publicly available, together with other routines, in a software library that can be downloaded from <http://www.dis.uniroma1.it/~cosin/>. A detailed description of some of these algorithms and of their performance analysis appears in [Donato et al. 2004c].

## 5.1 Data representation and multifiles

The first problem when dealing with massive data in secondary memory is the size limit of a single file<sup>2</sup>. 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 about the successors of each node. We refer to all these files related to a single graph as the `.ips` multifile. Figure 11 shows a simple graph together with its representation, details can be found in Donato et al. [2004c].

## 5.2 Generating webgraphs

In this section we discuss some algorithmic issues related to the generation of massive random webgraphs according to the models described above. The input for a random graph generator is  $N$ , the number of nodes of the graph, together with specific parameters

<sup>2</sup>This limit can be changed but we preferred, for portability reasons, to use a multifile format.

```

0          3          4
1          2          5          6

```

Fig. 11. An example graph and its representation in the .ips format

for each model. We assume that the graph cannot be stored in main memory. We focus on the Evolving and Copying model, and later discuss how to extend the techniques to the other models presented.

**Evolving model.** For the Evolving model we need to generate the end-point of an edge with probability proportional to the in-degree of a vertex<sup>3</sup>.

The straightforward approach is to keep in main memory an  $N$ -element array  $i$  where we store the in-degree for each generated node, so that  $i[k] = \text{indegree}(v_k) + 1$  (the plus 1 is necessary to give to each vertex an initial non-zero probability to be chosen as an end-point). Assume that we are currently generating the links from the  $g + 1$ -th vertex;

<sup>3</sup>The problem of generating efficiently random discrete variables according to general distributions has been for instance considered by Walker [1977]; see also Knuth [1997].

we denote by  $I$  the total in-degree of the vertices  $v_1 \dots v_g$  plus  $g$ , i.e.  $I = \sum_{j=1}^g i[j]$ . We randomly (and uniformly) generate a number  $r$  in the interval  $(1 \dots I)$ ; then, we search for the smallest integer  $k$  such that  $r \leq \sum_{j=1}^k i[j]$ : we add the link from the  $g + 1$ -th vertex to the  $k$ -th. In this way, using a uniform random number generator we are able to generate numbers according to the preferential attachment rule. For massive graphs, this approach has two main drawbacks: i.) We need to keep in main memory the whole in-degree array to speed up operations; ii.) It can be difficult to quickly identify the integer  $k$ .

To overcome both problems the idea is to partition the set of vertices in a certain number of blocks, let's say  $B$ . We use a  $B$ -element array  $S$  to keep the sum of the in-degrees of all the vertices in the block. Then we alternate the following 2 phases:

*Phase I.* We store in main memory tuples corresponding to pending edges, i.e. edges that have been decided but not yet stored. Tuple  $t = \langle g + 1, k', r - \sum_{j=1}^{k'-1} S[j] \rangle$  associated with vertex  $g + 1$ , maintains the block number  $k'$  and the relative position of the endpoint within the block. We also group together the tuples referring to a specific block. We switch to phase II when a sufficiently large number of tuples has been generated.

*Phase II.* In this phase we generate the edges and we update the information on disk. This is done by considering, in order, all the tuples that refer to a single block when this is moved to main memory. For every tuple, we find the pointed node and we update the information stored in  $i$ . The list of successors is also stored as the graph is generated.

In the real implementation we use multiple levels of blocks, instead of only one, in order to speed up the process of finding the endpoint of an edge. An alternative is the use of additional data structures to speed up the process of identifying the position of the node inside the block, like dyadic rangesums.

**Copying model.** The Copying model is parameterized with a copying factor  $\alpha$ . As we recalled in Section 4, every new vertex  $u$  inserted in the graph by the Copying model is connected with  $d$  edges to previously existing vertices. A random prototype vertex  $p$  is also selected. The endpoint of the  $l$ th outgoing edge of vertex  $u$ ,  $l = 1, \dots, d$ , is either copied with probability  $\alpha$  from the endpoint of the  $l$ th outgoing link of vertex  $p$ , or chosen uniformly at random among the existing nodes with probability  $1 - \alpha$ .

A natural strategy would be to generate the graph with a batch process that, alternately, i) generates edges and writes them to disk and ii) reads from disk the edges that need to be "copied". This clearly requires two accesses to disk for every newly generated vertex.

Instead we generate for every node  $1 + 2 \cdot d$  random integers: one for the choice of the prototype vertex,  $d$  for the endpoints chosen at random, and  $d$  for the values of  $\alpha$  drawn for the  $d$  edges. We store the seed of the random number generator at fixed steps, say every  $x$  generated nodes.

When we need to copy an edge from a prototype vertex  $p$ , we step back to the last time when the seed has been saved before vertex  $p$  has been generated, and let the computation progress until the outgoing edges of  $p$  are recomputed; for an appropriate choice of  $x$ , this sequence of computations is still faster than accessing the disk. Observe that  $p$  might also have copied some of its edges. In this case we recursively refer to the prototype vertex of  $p$ . We store the generated edges in a memory buffer and write it to disk when complete.

**PageRank model.** The PageRank model is a generalization of the Evolving model, and therefore the only difference in the algorithm sketched above is that we have to store (in main memory) also the preassigned PageRank values for each node.

**Multi-Layer model.** The Multi-Layer model presents more difficulties, and we implemented it by generating each layer by itself and then merging the result. This required the use of an additional array for each layer to keep track of the correspondence between vertices in the single layer and in the “whole” graph.

### 5.3 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 *reached*[ $u$ ] is true if  $u$  has already been reached, and false otherwise.
- (2) The second bit *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 *reached* are also *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.

### 5.4 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.

### 5.5 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 [Sibeyn et al. 2002]. 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 [Sibeyn et al. 2002] 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.

## 5.6 Computation of the SCCs

It is well known [Cormen et al. 1992] 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 [Tarjan 1972]. 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 5.3, 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 the all pages, using a starting set of just 20 nodes, the probability of not finding the CORE is only  $(3/4)^{20} \approx 0,3\%$ .

## 5.7 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:

$$\begin{aligned} \text{TENDRILS} &= \text{TENDRILS\_IN} \cup \text{TENDRILS\_OUT} \\ \text{TUBES} &= \text{TENDRILS\_IN} \cap \text{TENDRILS\_OUT} \end{aligned}$$

**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.

## 5.8 Disjoint bipartite cliques

Kumar et al. [1999] 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.

Our algorithm, that is detailed in [Donato et al. 2004c], is an external one that stores the graph in secondary memory in a number of blocks. Every block  $b$ ,  $b = 1, \dots, \lceil N/B \rceil$ , contains the list of successors and the list of predecessors of  $B$  vertices of the graph. The idea is that, at each iteration, we load in the main memory a single block, and we try to compute all the bipartite cliques (BC) inside that one; at the same time, we keep track (in a buffer file), of the *partially* computed BC, i.e. the ones that could become BC depending on the part of the graph that either we do not have seen yet or we have seen “without knowing it could have been part of a BC”. Our algorithm need to bring in main memory each block at most twice; the first when it builds the partially computed BC, and the second when it looks for the missing parts.

## 5.9 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 out-degree 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

*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

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

We implement the external memory algorithm proposed by Haveliwala [1999]. Note that several approaches to compute PageRank have been proposed, including parallel ones [Gleich et al. 2004]. The algorithm uses a list of successors *Links*, and two arrays *Source* and *Dest* that store the vector Rank at iteration  $i$  and  $i + 1$ . The computation proceeds until either the error  $r = ||Source - Dest||$  drops below a fixed value  $\tau$  or the number of iterations exceeds a prescribed value.

Arrays *Source* and *Dest* are partitioned and stored into  $\beta = \lceil N/B \rceil$  blocks, each holding the information on  $B$  vertices. *Links* is also partitioned into  $\beta$  blocks, where *Links<sub>l</sub>*,  $l = 0, \dots, \beta - 1$ , contains for every vertex of the graph only those successors directed to vertices in block  $l$ , i.e. in the range  $[lB, (l + 1)B - 1]$ . We bring to main memory one block of *Dest* per time. Say we have the  $i$ th block of *Dest* in main memory. To compute the new PageRank values for all the nodes of the  $i$ th block we read, in a streaming fashion, both arrays *Source* and *Links<sub>i</sub>*. From array *Source* we read previous PageRank values, while from *Links<sub>i</sub>* we have the list of successors (and the out-degree) 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.

## 6. CONCLUSIONS

In this paper we presented an overview of modeling the Webgraph. We first detailed its known properties, then we discussed and compared several random graph generators that try to model it. We examined also the algorithmic aspects of this research, and we detailed several algorithms to generate and measure massive webgraphs. It should be clear that we made only a first step in modeling the Webgraph: several aspects are missing and, among them, we mention that no model captures the out-degree distribution observed in samples, and the nature of the “rewiring” process, i.e. how a page changes its out-going links, is still obscure. 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 first step towards this direction has been presented in [Kraft et al. 2003], 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.

## REFERENCES

- ABELLO, J., PARDALOS, P. M., AND RESENDE, M. G. C. 2002. *Handbook of massive data sets*. Kluwer Academic Publishers.
- ADLER, M. AND MITZENMACHER, M. Towards compressing web graphs. Tech. Rep. 00-39, U.of Mass.
- B. BOLLOBAS, O. R. 2003. Robustness and vulnerability of scale-free random graphs. *Internet Mathematics* 1, 1, 1–35.
- BARABASI, A. AND ALBERT, A. 1999. Emergence of scaling in random networks. *Science* 286, 509.
- BOLDI, P., CODENOTTI, B., SANTINI, M., AND VIGNA, S. 2002. Ubicrawler: A scalable fully distributed web crawler.
- BOLDI, P. AND VIGNA, S. 2004. The webgraph framework i: compression techniques. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*. ACM Press, 595–602.
- BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* 30, 1–7, 107–117.
- BRODER, A., KUMAR, R., MAGHOUL, F., RAGHAVAN, P., RAJAGOPALAN, S., STATA, S., TOMKINS, A., AND WIENER, J. 2000. Graph structure in the web. In *Proceedings of the 9th WWW conference*.
- CHO, J. AND GARCIA-MOLINA, H. 2002. Parallel crawlers. In *Proc. of the 11th International World-Wide Web Conference*.
- CHO, J., GARCIA-MOLINA, H., HAVELIWALA, T., LAM, W., PAEPCKE, A., RAGHAVAN, S., AND WESLEY, G. 2006. Stanford WebBase Components and Applications. *ACM Transactions on Internet Technology* 6, 2.
- CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1992. *Introduction to algorithms*, 6th ed. MIT Press and McGraw-Hill Book Company.
- cyveillance. Cyveillance. [www.cyveillance.com](http://www.cyveillance.com).
- DIESTEL, R. 1997. *Graph Theory*. Springer, New York.
- DILL, S., KUMAR, R., MCCURLEY, K., RAJAGOPALAN, S., SIVAKUMAR, D., AND TOMKINS, A. 2001. Self-similarity in the web. In *Proceedings of the 27th VLDB Conference*.
- DONATO, D., LAURA, L., LEONARDI, S., AND MILLOZZI, S. 2004a. Large scale properties of the webgraph. *European Journal of Physics B* 38, 2, 239–243. DOI: 10.1140/epjb/e2004-00056-6.
- DONATO, D., LAURA, L., LEONARDI, S., AND MILLOZZI, S. 2004b. Simulating the webgraph: A comparative analysis of models. *Computing in Science and Engineering* 6, 6, 84–89.
- DONATO, D., LAURA, L., LEONARDI, S., AND MILLOZZI, S. 2004c. A software library for generating and measuring massive webgraphs. Tech. Rep. D13, COSIN European Research Project. <http://www.dis.uniroma1.it/~cosin/html/pages/COSIN-Tools.htm>.
- ERDÖS, P. AND R., R. 1960. *Publ. Math. Inst. Hung. Acad. Sci* 5.
- GLEICH, D., ZUCHOV, L., AND BERKHIN, P. 2004. Fast Parallel PageRank: A Linear System Approach. Tech. Rep. 038, Yahoo! Research.
- GULLI, A. AND SIGNORINI, A. 2005. The Indexable Web is More than 11.5 Billion Pages. In *Proceedings of WWW-05, International Conference on the World Wide Web*.
- HARARY, F. 1969. *Graph Theory*. Addison-Wesley, Reading, MA.
- HAVELIWALA, T. H. 1999. Efficient computation of PageRank. Tech. rep., Stanford University.
- KLEINBERG, J. 1997. Authoritative sources in a hyperlinked environment. *Journal of the ACM* 46, 5, 604–632.
- KLEINBERG, J., KUMAR, R., RAGHAVAN, P., RAJAGOPALAN, S., AND TOMKINS, A. 1999. The web as a graph: measurements, models and methods. In *Proc. Intl.Conf. on Combinatorics and Computing*. 1–18.
- KNUTH, D. E. 1997. *Seminumerical Algorithms*, Third ed. The Art of Computer Programming, vol. 2. Addison-Wesley, Reading, Massachusetts.
- KRAFT, R., HASTOR, E., AND STATA, R. 2003. Timelinks: Exploring the link structure of the evolving web. In *Second Workshop on Algorithms and Models for the Web-Graph (WAW2003)*.
- ACM Journal Name, Vol. V, No. N, Month 20YY.

- KUMAR, R., RAGHAVAN, P., RAJAGOPALAN, S., SIVAKUMAR, D., TOMKINS, A., AND UPFAL, E. 2000. Stochastic models for the web graph. In *Proc. of 41st FOCS*. 57–65.
- KUMAR, R., RAGHAVAN, P., RAJAGOPALAN, S., AND TOMKINS, A. 1999. Trawling the web for emerging cyber communities. In *Proc. of the 8th WWW Conference*. 403–416.
- LAURA, L., LEONARDI, S., CALDARELLI, G., AND DE LOS RIOS, P. 2002. A multi-layer model for the webgraph. In *On-line proceedings of the 2nd International Workshop on Web Dynamics*.
- MITZENMACHER, M. 2003. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics* 1, 2.
- PANDURANGAN, G., RAGHAVAN, P., AND UPFAL, E. 2002. Using pagerank to characterize web structure. In *Proc. of the 8th Annual International Conference on Combinatorics and Computing (COCOON)*, Springer-Verlag, Ed. LNCS 2387. 330–339.
- PENNOCK, D., FLAKE, G., LAWRENCE, S., GLOVER, E., AND GILES, C. 2002. Winners don't take all: Characterizing the competition for links on the web. *Proc. of the National Academy of Sciences* 99, 8 (April), 5207–5211.
- RAGHAVAN, S. 2002. Personal communication.
- SIBEYN, J., ABELLO, J., AND MEYER, U. 2002. Heuristics for semi-external depth first search on directed graphs. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*. 282–292.
- TARJAN, R. E. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1, 2, 146–160.
- VITTER, J. AND SHRIVER, E. 1994a. Algorithms for parallel memory i: Two-level memories. *Algorithmica* 12, 2-3, 107–114.
- VITTER, J. AND SHRIVER, E. 1994b. Algorithms for parallel memory ii: Hierarchical multilevel memories. *Algorithmica* 12, 2-3, 148–169.
- WALKER, A. 1977. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Mathematical Software* 3, 3, 253–256.
- webbase. The stanford webbase project. [http://www-diglib.stanford.edu/\\$sim\\$testbed/doc2/WebBase/](http://www-diglib.stanford.edu/$sim$testbed/doc2/WebBase/).
- WITTEN, I. H., MOFFAT, A., AND BELL, T. C. 1999. *Managing gigabytes (2nd ed.): compressing and indexing documents and images*. Morgan Kaufmann Publishers Inc.