# A Large Time–Aware Web Graph

Paolo Boldi
*boldi@dsi.unimi.it*

Massimo Santini
*santini@dsi.unimi.it*

Sebastiano Vigna
*vigna@dsi.unimi.it*

Dipartimento di Scienze dell'Informazione
Università di Milano, Italy

**Abstract**

We describe the techniques developed to gather and distribute in a highly compressed, yet accessible, form a series of twelve snapshot of the `.uk` web domain. *Ad hoc* compression techniques made it possible to store the twelve snapshots using just 1.9 bits per link, with constant-time access to temporal information. Our collection makes it possible to study the temporal evolution link-based scores (e.g., PageRank), the growth of online communities, and in general time-dependent phenomena related to the link structure.

## 1 Introduction

By now, several sources provide accessible snapshots of web data. The Stanford WebBase project, for instance, provides hundreds of Terabytes of such data. In the last years, the LAW (Laboratory for Web Algorithmics) focused its efforts on web graphs instead.

Within the DELIS project, interest rose about *temporal* link analysis, that is, studying how the web-graph nodes and arcs evolves in time. Since without proper bounds the amount of information required by this activity is staggering, we decided to concentrate our efforts on gathering twelve monthly 100 Mpages snapshots of the `.uk` domain and store them in a format that would make temporal link information accessible on a standard workstation.

The basis of our work is WebGraph [2], a framework for web graph compression that currently provides the best compression available (in terms of bits per link) and whose data can be accessed using free Java or C++ code [6]. One of the main challenges of this work was to extend WebGraph so that it would compress efficiently labels representing temporal information. Moreover, we wanted to extend the standard WebGraph flexible approach, that makes it always possible to load data into main memory or access it in offline form (with a performance drop, of course).

| | Pages | Size (GB) | GZip'd Size (GB) |
|---|---|---|---|
| June | 112 386 763 | 1 893 | 402 |
| July | 136 956 559 | 2 287 | 477 |
| August | 141 395 895 | 2 424 | 507 |
| September | 148 965 298 | 2 756 | 546 |
| October | 129 558 491 | 2 336 | 478 |
| November | 150 146 132 | 2 637 | 546 |
| December | 144 489 446 | 2 552 | 525 |
| January | 151 578 113 | 2 651 | 553 |
| February | 153 966 540 | 2 692 | 564 |
| March | 151 427 461 | 2 568 | 545 |
| April | 150 606 689 | 2 700 | 559 |
| May | 150 054 551 | 2 658 | 556 |

| | Nodes | Arcs | Size (GB) | bit/arc |
|---|---|---|---|---|
| June | 80 644 902 | 2 481 281 617 | 0.89 | 3.07 |
| July | 96 395 298 | 3 030 665 444 | 1.16 | 3.30 |
| August | 100 751 978 | 3 250 153 746 | 1.23 | 3.25 |
| September | 106 288 541 | 3 871 625 613 | 1.32 | 2.93 |
| October | 93 463 772 | 3 130 910 405 | 1.03 | 2.83 |
| November | 106 783 458 | 3 479 400 938 | 1.16 | 2.86 |
| December | 103 098 631 | 3 768 836 665 | 1.34 | 2.77 |
| January | 108 563 230 | 3 929 837 236 | 1.38 | 2.72 |
| February | 110 123 614 | 3 944 932 566 | 1.39 | 2.74 |
| March | 107 565 084 | 3 642 701 825 | 1.34 | 2.84 |
| April | 106 867 191 | 3 790 305 474 | 1.36 | 2.79 |
| May | 105 896 555 | 3 738 733 648 | 1.30 | 2.69 |

Table 1: Per-snapshot full-text and web-graph stats.

## 2 Gathering the Snapshots

The snapshots have been taken at the start of each month, during a period of $7-10$ days, using the bandwidth provided by the Università degli Studi di Milano and a cluster of PCs that has been funded by the DELIS project. Some basic information about the snapshots is shown in Table 1.

**Crawling parameters.** As in any limited-size crawl, it is essential to define the crawl parameters (the stopping criterion is clearly that of reaching about 100 Mpages, without counting duplicates). We highlight the main features:

**Crawl policy.** We use UbiCrawler's [1] built-in per-host breadth-first visit. A number of threads scan in parallel distinct hosts, and newly discovered URLs are added to a queue. When a thread completes its visit, it extracts from the queue the first URL whose host has an IP address that is not currently being visited, and starts visiting that host in breadth-first fashion.

**Seed.** The seed is a large (190 000 elements) set of URLs obtained from the Open Directory Project. The reason for such a large seed is that of making the crawl more stable and repeatable, and reduce the amount of spam (as links in the Open Directory Project are judged by humans).

**Maximum number of pages per host.** We limited each host to a maximum of 50 000 pages. This guarantees that we shall crawl at least 2000 hosts, and limits the impact of web traps and database-driven sites.

**Maximum inter-host depth.** We do not delve more than 16 levels in a host. The main reason for a limit in depth is avoiding traps and also badly configured 404 pages, which sometimes generate an infinite number of links by prefix buildup.

**URL normalisation.** URLs are normalised following the strategy explained in the `BURL`[1] Java class. We apply all safe normalisations, escape all illegal characters, and treat in a special way square brackets as they are ubiquitously (although erroneously) used in an unescaped form.

**Duplicate detection.** Many pages are duplicates, and to detect their presence we maintain a set of 64-bit fingerprints obtained after stripping attributes (of HTML elements) and other non-relevant parts of the page. When a duplicate is detected we just store a pointer to the original page. About 25% of the overall pages happen to be duplicates.

## 3 Aligning the Snapshots

The web is constantly changing, and network errors can affect the presence of a site or page during a given crawl, Nonetheless, Table 2 shows that we have a significant host overlap in the chosen twelve-months time span.

[1] The class is available in bundle with the LAW software, downloadable at `http://law.dsi.unimi.it/`.

| | Jun | Jul | Aug | Sep | Oct | Nov | Dec | Jan | Feb | Mar | Apr | May |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Jun | 94.9 | 73.3 | 71.6 | 69.8 | 65.5 | 64.5 | 59.4 | 62.4 | 62.4 | 58.9 | 57.6 | 57.7 |
| Jul | | 130.7 | 102.2 | 99.4 | 89.9 | 90.9 | 81.4 | 86.7 | 88.1 | 84.0 | 82.7 | 82.1 |
| Aug | | | 128.5 | 102.8 | 84.9 | 90.3 | 81.0 | 86.4 | 86.7 | 81.9 | 80.0 | 79.6 |
| Sep | | | | 136.6 | 88.0 | 94.6 | 84.3 | 90.8 | 89.6 | 84.9 | 82.0 | 81.1 |
| Oct | | | | | 109.9 | 86.1 | 75.8 | 81.1 | 81.6 | 76.6 | 75.6 | 75.1 |
| Nov | | | | | | 121.2 | 86.7 | 91.4 | 91.5 | 84.1 | 82.3 | 81.6 |
| Dec | | | | | | | 113.4 | 88.8 | 84.3 | 79.2 | 76.2 | 75.8 |
| Jan | | | | | | | | 125.1 | 94.2 | 86.4 | 84.4 | 83.1 |
| Feb | | | | | | | | | 122.9 | 91.0 | 87.8 | 86.7 |
| Mar | | | | | | | | | | 122.5 | 84.9 | 83.8 |
| Apr | | | | | | | | | | | 113.1 | 91.6 |
| May | | | | | | | | | | | | 114.5 |

Table 2: Host overlap, in thousands.

| | Jun | Jul | Aug | Sep | Oct | Nov | Dec | Jan | Feb | Mar | Apr | May |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Jun | 31.3 | 19.0 | 18.3 | 17.2 | 15.3 | 15.0 | 13.8 | 13.7 | 13.2 | 12.3 | 11.9 | 11.1 |
| Jul | | 35.2 | 23.3 | 21.7 | 18.5 | 18.3 | 16.2 | 16.4 | 16.0 | 15.2 | 14.5 | 13.6 |
| Aug | | | 37.3 | 24.3 | 19.4 | 19.4 | 17.1 | 17.3 | 16.7 | 15.5 | 15.0 | 13.8 |
| Sep | | | | 39.9 | 21.2 | 21.2 | 18.7 | 19.0 | 18.1 | 16.9 | 16.3 | 14.7 |
| Oct | | | | | 33.8 | 22.2 | 19.0 | 19.1 | 18.3 | 16.6 | 16.4 | 15.0 |
| Nov | | | | | | 37.3 | 22.3 | 21.9 | 21.3 | 18.8 | 18.3 | 16.6 |
| Dec | | | | | | | 36.6 | 23.5 | 21.0 | 19.0 | 17.9 | 16.6 |
| Jan | | | | | | | | 39.0 | 23.7 | 20.8 | 19.9 | 18.4 |
| Feb | | | | | | | | | 37.7 | 23.1 | 22.2 | 20.0 |
| Mar | | | | | | | | | | 38.1 | 22.4 | 20.2 |
| Apr | | | | | | | | | | | 36.9 | 24.2 |
| May | | | | | | | | | | | | 36.9 |

Table 3: Static URLs overlap, in millions.

The first important step in getting a temporally labelled collection is *alignment*: identifying URLs in different snapshots that correspond to the same web page. Alignment is a non-trivial issue because if a URL is not static it might contain session-generated data (e.g., a session ID) that makes *de facto* identical URLs appear as syntactically distinct.

For the present collection, a radical choice was made: the only allowed URLs are static URLs (i.e., URLs that do not contain a question mark[2]), although we plan to develop some reasonably sound alignment technique for dynamic URLs in the future. Table 3 shows that, as a first attempt, this choice is not unreasonable.

[2]This choice, unfortunately, cannot prevent *opaque* session-dependent URLs from generating noise in the collection.

# 4 Temporal Labelling a Graph

Once URLs are aligned, it is possible to build a *global* graph $G$ that includes all static pages (and related links) appearing in each snapshot. The graph $G$ must be labelled so that, for each node and each link, we can detect whether it was present or not in any a given snapshot. Essentially, we need to store twelve bits of information per node and per arc.

We decided to use the labelling facility implemented in WebGraph to store a twelve-bit label for each node and arc. To reduce significantly the space occupied, we generated $2^{12}$ canonical Huffman coders [5], one for each possible node label. Each coder contains an optimal, canonical code for the distribution of the labels of the arcs going out of nodes with a fixed label. The distribution on the outgoing arc is strongly dependent on the label of the source, and we exploit this fact to increase the compression ratio.

# 5 Memory and disk footprint

Memory and disk occupation depend on which components are loaded in memory (as opposed to being accessed directly on disk). The underlying graph (representing each node and arc ever met during the twelve crawls) has 133.6 millions of nodes and 5.5 billions of arcs. WebGraph uses in this case $\approx 2.6$ bits per link, resulting in a bitstream with a memory footprint of 1.7 GiB.

As we discussed previously, we use $2^{12}$ canonical Huffman decoders, which require around 66 MiB of core memory. Node and arc labels occupy around 1.2 GiB, implying a cost of just 2.16 bits per label. Since the overall graph represents 13.8 billion links, the cost per link of the overall representation is just 1.9 bits per link.

We still have to consider the about 267 million pointers that are necessary to access the graph bitstream and the arc-label bit stream. WebGraph exploits a broadword implementation [7] of the Elias–Fano representation of monotone functions [3, 4], which provides constant-time pointer access with minimal space occupancy. As a result, we use 9 bits per graph bitstream pointer and 8.3 bits per label pointer—a memory footprint of about 274 MiB.

Finally, random access to a labelled arc requires about 250 ns on an Opteron at 2.8 Ghz, making it possible to apply standard algorithmic techniques requiring random access to the graph (for instance, visits over the whole graph) using a commodity workstation.

# 6 Conclusions

We have presented the first large-scale time-aware publicly available web graph. Our software and data representation make it possible to access randomly structural and temporal data quickly on a standard workstation. There are however several possible improvements:

- dynamic URL alignment is difficult, but not impossible, and it is clearly necessary if one wants to conduct a more precise analysis;
- it is imperative to provide statistical evidence of the bias due to crawling policies and to influence of the crawling limits imposed.

Work is under way to tackle both issues. We are providing, in the mean time, the raw data we gathered. The twelve graphs are also available independently, and a separate note in this SIGIR Forum issue describes a method and Java implementation for distributing part of the page content, which, due to its very large size (7 TB) would be very difficult to serve on the network.

# References


[1] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[2] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[3] P. Elias. Efficient storage and retrieval by content and address of static files. *J. Assoc. Comput. Mach.*, 21(2):246–260, 1974.

[4] R. M. Fano. On the number of bits required to implement an associative memory. Memorandum 61, Computer Structures Group, Project MAC, MIT, Cambridge, Mass., n.d., 1971.

[5] D. S. Hirschberg and D. A. Lelewer. Efficient decoding of prefix codes. *Comm. ACM*, 33(4):449–459, 1990.

[6] J. Ratkiewicz. WebGraph++, 2006. `http://homer.informatics.indiana.edu/~nan/webgraph/`.

[7] S. Vigna. Broadword implementation of rank/select queries. In *WEA 2008: Proc. of the 7th International Workshop on Experimental Algorithms*, number 5038 in Lecture Notes in Computer Science, pages 154–168. Springer–Verlag, 2008.