Efficient and Simple Encodings for the Web Graph.

Jean-Loup Guillaume
LIAFA, Université Paris 7, case 7014
2, place Jussieu,
75005 Paris (France)
+33 (0) 1 44 27 28 37
guillaume@liafa.jussieu.fr

Matthieu Latapy
Projet Hipercom
INRIA Rocquencourt,
F-78153 Le Chesnay (France)
+33 (0) 1 39 63 52 25
latapy@liafa.jussieu.fr

Laurent Viennot
Projet Hipercom
INRIA Rocquencourt,
F-78153 Le Chesnay (France)
+33 (0) 1 39 63 52 25
Laurent.Viennot@inria.fr

Abstract: In this paper, we propose a set of simple and efficient methods based on standard, free and widely available tools, to store and manipulate large sets of URLs and large parts of the Web graph. Our aim is both to store efficiently the URLs list and the graph in order to manage all the computations in a computer central memory. We also want to make the conversion between URLs and their identifiers as fast as possible, and to obtain all the successors of an URL in the Web graph efficiently. The methods we propose make it possible to obtain a good compromise between these two challenges, and make it possible to manipulate large parts of the Web graph.

Keywords: Web graph, Web links, URLs, Compression.

1 Introduction.

One can view the Web as a graph whose vertices are Web pages, and edges are hyperlinks from one page to another. Understanding the structure of this graph is a key challenge for many important present and future applications. Information retrieval, optimized crawling and enhanced browsing are some of them. The first step to study the Web graph is to be able to store and manipulate it efficiently, both in

space and in time terms. The key element of this encoding is to associate a unique identifier to each URL which will then be used to encode the graph.

URLs are more than 60 bytes long on average and each vertex has an average outdegree at least seven (following a power law distribution [3]) depending on the considered domain [1, 8]. Storing a one million vertices subgraph of the Web graph without any compression would therefore need more than 100 MB of memory. When one is concerned with the Web graph, it is important to deal with much bigger graphs, classically several hundreds of millions vertices. Therefore, the efficient encoding of the graph becomes a crucial issue. The challenge is then to find a good balance between space and time requirements [2, 10].

Our aim is to provide an efficient and simple solution using only standard, free and widely available tools, namely `sort`, and `gzip` [4, 5]. We tested our methods on a 27 millions vertices and 133 millions links crawl performed with the larbin crawler [7] inside the "`.fr`" (4 millions URLs) and "`.edu`" (23 millions URLs) domains in November 2001. Although it may be considered as relatively small, this set of data is representative of the Web graph since it is consistent with the known statistics [1, 3].

All the experiments have been made on a Compaq™ Workstation AP 550, with a 800 MHz Pentium™ III processor, with 1 GB memory and a Linux 2.4.9 kernel. We obtained an encoding of each URL in 6.85 bytes on average with a conversion between URLs and identifiers (in both directions) in about 400μ s. One-way links can also be compressed to 1.6 byte on average with immediate access (lower than 20 μs), which can be improved to 1.2 byte if one allows slower access.

We describe in Section 2 our method to associate a unique identifier to each URL, based on the lexicographical order. We show how to compress the URLs set and how to obtain fast conversion between URLs and identifiers. In Section 3, we notice some properties on the graph itself, concerning a notion of distance between vertices and their successors. These properties explain the good results obtained when we compress the graph. Two different and opposite approachs are discussed concerning the compression: one of them optimizes space use, and the other one optimizes access time.

2 URLs Encoding

Given a large set of URLs, we want to associate a unique identifier (an integer) to each URL, and to provide a function which can make the mapping between identifiers and URLs. A simple idea consists in sorting all the URLs lexicographically. Then a URL identifier is its position in the set of sorted URLs. We will see that this choice for an identifier makes it possible to obtain efficient encoding.

Let us consider a file containing a (large) set of URLs obtained from a crawl. First notice that sorting this file improves its compression since it increases the local

redundancy of the data: we obtained an average of 7.27 bytes by URL before sorting (63 bytes for unsorted urls) and an average of 5.73 bytes after sorting. Using this compression method is very inefficient in terms of lookup time, since when one converts a URL into its identifier and conversely, one has to uncompress the entire file. On the other hand, random access compression schemes exist [6, 9], but their compression rate are much lower, too much for our problem.

Another solution consists in spliting the file into blocks and compressing independently each of them. We save this way a large amount of time since only one block has to be uncompressed to achieve the mapping. Moreover, since the URLs are sorted, the ones which share long common prefixes are in the same block, and so we do not damage the compression rate too much.

Experimentally, the average size for a compressed URL does not significantly increases as long as blocks length stays over one thousand URLs. In this case, URL average size is 5.62 bytes long. With blocks of one hundred URLs, the average size grows up to 6.43 bytes long. Notice that the method can be improved by taking blocks of different sizes, depending on the local redundancy of the URLs list (See Figure 1).
One can then convert a URL into its identifier as follows:

1. Find the block which contains the URL to convert: (dichotomic search based on the knowledge of the first URL of each block).

2. Uncompress the block.

3. Find the identifier of the URL inside the (uncompressed) block (linear search in the list).

Conversely, one can convert an identifier to a URL as follows:

1. Find the block which contains the identifier to convert (the block number is $\frac{\text{Identifier}}{\text{BlocksLength}}$).

2. Uncompress the block.

3. Find the URL in the (uncompressed) block (*i.e.* line number Identifier $-$ BlocksLength $\cdot$ BlockNumber in block.

Notice that the third step of each of these conversions implies a linear search in a block. To improve this, we add at the end of all the URLs in a given block as many occurrences of a special character as necessary to make it as long as the longest URL in the block. In each block, the fixed length is then the length of the longest URL. Therefore, the third point of the URL to identifier conversion becomes a dichotomic search in the block, and the third point of the identifier to URL conversion can be done in constant time since the URL is at position UrlsLength $\cdot$ (Identifier $-$ BlocksLength $\cdot$ BlockNumber) in the block.

This optimization must be done carefully to ensure both a good compression of the URLs and a fast expansion (Step 2). If the blocks size is too low, compression rate will be naturally low. On the opposite, if the size is too important, the probability that a very long URL lies in the file will increase, adding a lot of unused character, which are going to increase the average URL size. Expansion time is linear with respect to the blocks length, so we must use as small blocks as possible to get fast mapping. Using median blocks length will result in very good compression rate but median expansion speed. Results showing these phenomena can be found in Figure 1.

In conclusion, we obtained a coding of the URLs in 6.85 bytes in average, with conversion between URLs and their identifiers in about 400μ s in both directions. This coding associates to each URL its position in the entire list with respect to the lexicographic order. We will now see how this encoding can be used to represent large parts of the Web graph.

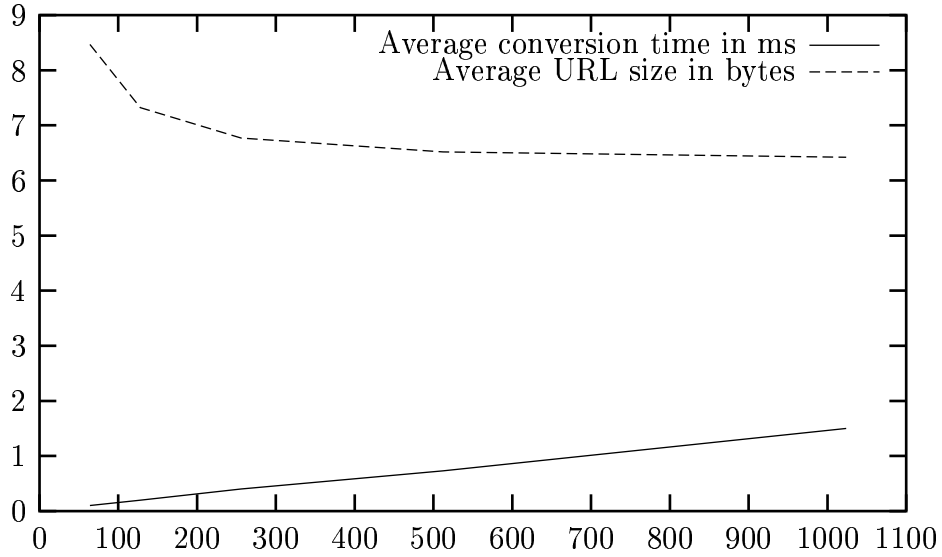

Figure 1: Average URL size and conversion times with respect to the size of the considered blocks, using fixed-length URLs.

3 Graph Encoding

As soon as the mapping between URLs and identifiers is defined, we can try to compress all links as much as possible. A link is defined by a couple of integers, each of them being the identifier of a URL as defined in Section 2. The graph is then stored in a file such that line number k contains the identifiers of all the successors of vertex

k (in a textual form). Using `gzip` to compress this file, we obtain a very compact encoding: 0.83 byte by link on average. Again, these values may be considered as lower bounds for the space needed to represent a link.

Using the same kind of block compression used in the previous section, one can obtain an encoding of each link in 1.24 byte in average with a lookup time of 0.45 ms, using 32 lines blocks. However, most of the operations made on the graph concern the exploration of successors or predecessors of vertices (during breadth-first search for instance). In this case, successors lookup time becomes a crucial parameter, and the encoding should be improved in terms of time. We are going to present another compression method which uses a strong property of the Web graph, the locality, to improve lookup time.

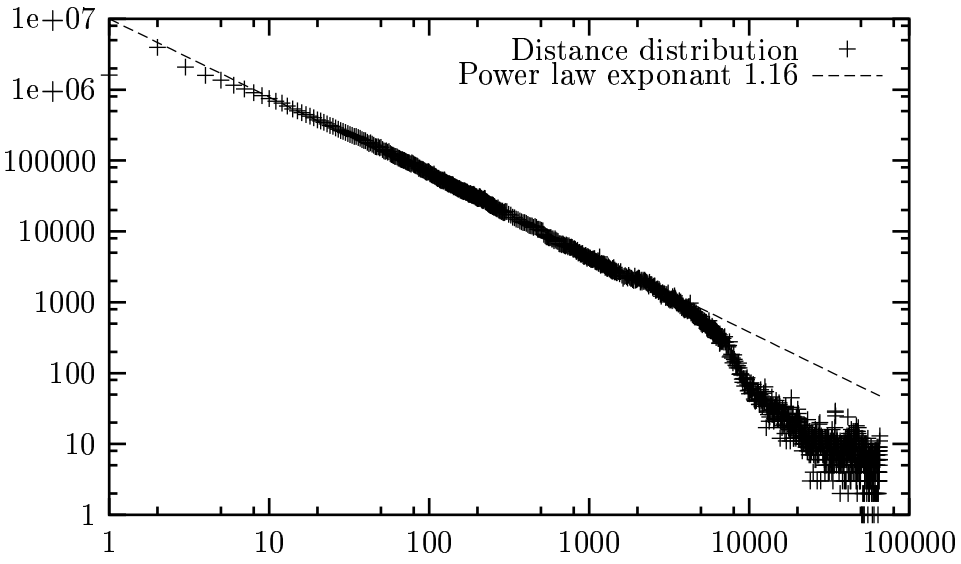

Figure 2: Distance distribution between vertices and their successors.

Let us define the *distance* between two URLs as the (signed) difference between their identifiers, and the *length* of a link between two URLs as the unsigned distance between these two URLs. Now, let us consider the distances distribution. This distribution follows a power law: the probability for the distance between two given vertices to be i is proportional to $i^{-1.16}$. See Figure 2.

Our experiments show that 68 percent of the URLs which are linked together are at distance between -255 and 255. We call these links *short* links. They can be encoded on 1 byte, plus 1 bit for the sign of the difference. Going further, one can distinguish short links (encoded on 1 byte), medium links (26.75 percent of the links, encoded on 2 bytes) and long (5.25 percent of the links, encoded on 4 bytes) links. We therefore use one bit per link to give the sign of the distance, and a prefix to know the type of the link (0 for short links, 10 for medium links and 11 for long links). This

	Average link size	Average lookup time for all the successors
identifiers	8 bytes	–
`gzip`ped identifiers	0.83 byte	–
distances	4.16 bytes	–
`gzip`ped distances	1.1 byte	–
`gzip`ped identifiers, blocks of 32 lines	1.24 byte	0.45 ms
short, medium, long links	1.71 byte	20 μs
short (Hufmann), medium, long links	1.58 byte	20 μs

Table 1: The average space needed to store one link, depending on the method used. The first four lines are just here to serve as references, since they imply either a very low compression ratio, or very slow elementary operations.

way, a link can be stored using 1.71 byte on average. If one use Huffman compression for short links, one more bit can be save.

4 Conclusion.

We described in this paper a simple and efficient method to encode large sets of URLs and large parts of the Web graph. We gave a way to compute the position of a URL in the sorted list of all the considered URLs, and conversely, which makes it possible to manipulate large data sets in RAM, avoiding disk usage. Using this encoding, the conversion between identifiers and URLs takes around 400μ s on our computer, in both directions, and finding all the successors of a given URL takes around 0.5 ms. We can improve the link lookup to around 20 μs by using the second method we proposed, but with an increase of the space requirements.

We therefore obtained results which are comparable to the best results known in the literature, using only standard, free, and widely available tools like `sort` and `gzip`. Notice that the good performances of our method rely on the performances of these tools, which have the advantage of being strongly optimized. Moreover, let us emphasize on the fact that our method is scalable: it relies on *local* properties of the list of URLs which remain true independently of the size of the considered crawl.

References

[1] R. Albert, H. Jeong, and A.-L. Barabasi, *Diameter of the world wide web*, Nature **401** (1999), 130–131.

[2] Krishna Bharat, Andrei Broder, Monika Henzinger, Puneet Kumar, and Suresh Venkatasubramanian, *The Connectivity Server: fast access to linkage information on the Web*, Computer Networks and ISDN Systems **30** (1998), no. 1–7, 469–477.

[3] A. Z. Broder, S. R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener, *Graph structure in the web*, WWW9 / Computer Networks **33** (2000), no. 1-6, 309–320.

[4] P. Deutsch, *Deflate compressed data format specification version 1.3*, Aladdin Enterprises, May 1996, RFC 1951.

[5] ———, *Gzip file format specification version 4.3*, Aladdin Enterprises, May 1996, RFC 1952.

[6] Jirí Dvorsky, *Text compression with random access.*

[7] Larbin home page, *http://larbin.sourceforge.net/index-eng.html.*

[8] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins, *The Web as a graph: Measurements, models, and methods*, Proc. 5th Annual Int. Conf. Computing and Combinatorics, COCOON (T. Asano, H. Imai, D. T. Lee, S. Nakano, and T. Tokuyama, eds.), no. 1627, Springer-Verlag, 1999.

[9] Haris Lekatsas and Wayne Wolf, *Random access decompression using binary arithmetic coding*, Data Compression Conference, 1999, pp. 306–315.

[10] Rajiv Wickremesinghe, Raymie Stata, and Janet Wiener, *Link compression in the connectivity server*, Tech. report, Compaq systems research center, 2000.