Permuting Web Graphs*

Paolo Boldi Massimo Santini Sebastiano Vigna
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy

Abstract

Since the first investigations on web graph compression, it has been clear that the ordering of the nodes of the graph has a fundamental influence on the compression rate (usually expressed as the number of bits per link). The author of the LINK database [1], for instance, investigated three different approaches: an *extrinsic* ordering (URL ordering) and two *intrinsic* (or *coordinate-free*) orderings based on the rows of the adjacency matrix (lexicographic and Gray code); they concluded that URL ordering has many advantages in spite of a small penalty in compression. In this paper we approach this issue in a more systematic way, testing some old orderings and proposing some new ones. Our experiments are made in the WebGraph framework [2], and show that the compression technique and the structure of the graph can produce significantly different results. In particular, we show that for the *transpose* web graph URL ordering is significantly less effective, and that some new orderings combining host information and Gray/lexicographic orderings outperform all previous methods. In particular, in some large transposed graphs they yield the quite incredible compression rate of 1 bit per link.

1 Introduction

The web graph [3] is a directed graph whose nodes correspond to URLs, with an arc from x to y whenever page denoted by x contains a hyperlink toward page denoted by y; more loosely, the same term is sometimes used for the undirected version of the graph, when arc direction is not relevant. Web graphs are a huge source of information, and contain precious data that find applications in ranking, community discovery, and more. In many cases, results obtained and techniques applied for the web graph are also appropriate for the larger realm of *social networks*, of which the web graph is only a special case.

One nontrivial practical issue when dealing with web graphs is their size: a typical web graph contains millions, sometimes billions, of nodes and although sparse its adjacency matrix is way too big to fit in memory, even on large computers. To overcome this technical difficulty, one can access the graph in an offline fashion, which however requires to design special offline algorithms even for the most basic problems (e.g., finding connected components or computing shortest paths); alternatively, one can try to compress the adjacency matrix so that it can be loaded into memory and still be directly accessed without decompressing it (or, decompressing it only partially, on-demand, and efficiently).

*This work is partially supported by the EC Project DELIS, by MIUR PRIN Project "Automi e linguaggi formali: aspetti matematici e applicativi", and by MIUR PRIN Project "Web Ram: web retrieval and mining".

The latter approach, that can be referred to as *web graph compression*, can be traced back to the Connectivity Server [4] and to the LINK database [1]; more recently, it led to the development of the WebGraph framework [2], that still provides the best practical compression techniques.

Most web graph compression algorithms strongly rely on properties that are satisfied by the typical web graphs; in particular, the key properties that are exploited to compress the graph adjacency structure are locality and similarity. *Locality* means that most links from page x point to pages of the same host as x (and often share with x a long path prefix); *similarity* means that pages that are from the same host tend to have many links in common (this property tends to be more and more frequent with the widespread use of templates and generated content).

The fact that most compression algorithms make use of these (and similar) properties explains why such algorithms are so sensible to the way nodes are ordered. A technique that works incredibly well, and was adopted already in the Connectivity Server [4], consists in sorting nodes lexicographically by URL (node 0 is the one that corresponds to the lexicographically first URL, and so on). In this way, successors lists contain by locality URLs that are assigned close numbers, and *gap encoding*[1], a standard technique borrowed by inverted-index construction, makes it possible to store each successor using a small number of bits. This solution is usually considered good enough for all practical purposes, and has the extra advantage that even the URL list can be compressed very efficiently via prefix omission. Analogous techniques, which use additional information beside the web graph itself, are called *extrinsic*.

It is natural to wonder if there is an alternative way of finding a "good ordering" of the nodes that allow one to obtain the same (or maybe better) compression rate without having to rely on URLs; this is especially urgent for social network graphs, where nodes do not correspond themselves to URLs.[2] A general way to approach this problem is the following: take the graph with some ordering of its n nodes, and let A be the corresponding adjacency matrix; now, based on A, find some permutation π of its rows and columns such that, when applied to A, produces a new matrix A' in which two rows are similar (i.e., they contain 1's more or less in the same positions) iff they are close to each other (i.e., appear consecutively, or almost consecutively).

Finding a good permutation π is an interesting problem in its own account; in [1], the authors propose to choose the permutation π that would sort the rows of A in lexicographic ordering. This is an instance of a more general approach: fix some total ordering $\prec$ on the set of n-bit vectors (e.g., the lexicographic ordering), and let π be the permutation[3] that sorts the rows of A according to $\prec$. Observe that the rows of A' are *not* $\prec$-ordered, because the permutation π is applied to *both* rows and columns. These approaches are called *intrinsic*, or *coordinate-free*, as they do not rely on external information (such as the URLs of each node).[4]

Another possible solution to the same problem, already briefly mentioned in [1], consists in letting $\prec$ be a Gray ordering, i.e., an ordering where adjacent vectors differ by exactly one bit. Although this solution may sound promising, one should carefully determine an efficient

[1] Instead of storing $x_0, x_1, x_2, \ldots$ we store, using a variable-length bit encoding, $x_0, x_1 - x_0, x_2 - x_1, \ldots$.

[2] We note that the same approach has been shown to be fruitful in the compression of inverted indices; see, for instance [5, 6, 7, 8].

[3] In this description we are ignoring the problem that π is not unique if A contains the same row many times.

[4] Of course, it is possible to devise coordinate-free methods that do not necessarily depend on some ordering; see, for instance, [5].

algorithm that finds the sorting permutation π with respect to (some) Gray ordering.

In this paper we explore experimentally, using the WebGraph framework, the improvements in compression due to permutations. Besides the classical permutations described above, we propose two new permutations based on the Gray ordering: however, we restrict the permutation to rows of the same host. Moreover, for the first time we provide experimental data on the *transposed* graph, showing that coordinate-free permutations provide a dramatic increase in compression, contrarily to what happens in the standard case.

2 Notation and Gray Code Basics

Von Neumann's notation. In the following we will adopt von Neumann's definition of natural numbers

$$x = \{ 0, 1, \ldots, x - 1 \},$$

which leads to a simple notation for sets of integers. We allow some ambiguity when writing exponentials: 2^n denotes the vectors of n bits, or, equivalently, the power set of n (interpreting the vectors as characteristic functions $n \to 2$). Since 2^n ambiguously denotes also the set $X = \{ 0, 1, \ldots, 2^n - 1 \}$ we assume the natural correspondence between the latter set in increasing order and the strings of n bits in lexicographic ordering. The mapping from strings to X is obviously given by base-2 evaluation.

In the following, if $x \in 2^n$, we use $x_0, x_1, \ldots$ to denote the bits of its binary expansion (x_0 being its least-significant bit). In other words, interpreting x as a characteristic function $n \to 2$, we let

$$x_k = x(k).$$

Gray codes and Gray orderings. An n-bit *Gray code* is an arrangement (i.e., a total ordering) of 2^n such that any two successive vectors[5] differ by exactly one bit; Gray codes, named after the physicist Frank Gray, find countless applications in computer science, physics and mathematics (we refer the interested reader to [9] for more information on this topic). The *ordering* imposed by a Gray code on 2^n is called a *Gray ordering*.

Even though there are many Gray codes, and thus many Gray orderings, one that is very simple to describe is the following one.

Lemma 1 *For every $x \in 2^n$, let $\bar{x} \in 2^n$ be defined recursively by*

$$\begin{aligned} \bar{x}_{n-1} &= x_{n-1} \\ \bar{x}_k &= x_{k+1} \oplus x_k, \end{aligned}$$

where $\oplus$ is the exclusive or. Then, $x \mapsto \bar{x}$ is a bijection, and the ordering $\prec$ defined by $\bar{x} \prec \bar{y}$ iff $x < y$ is a Gray ordering (the natural Gray ordering*).*

Table 1 shows the natural Gray code on 3 bits. Now, let us write $x \mapsto \hat{x}$ for the inverse

[5] We say that a' is the successor of a with respect to the ordering $<$ iff $a < a'$ and $a \leq b \leq a'$ implies either $a = b$ or $b = a'$.

x	$\bar{x}$
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Table 1: The natural Gray code on 3 bits: in the right-hand column, any two successive vectors differ in exactly one bit.

of $x \mapsto \bar{x}$; starting from the definition given in Lemma 1 one can easily see that $\hat{x}$ can be recursively defined as follows

$$\begin{aligned} \hat{x}_{n-1} &= x_{n-1} \\ \hat{x}_k &= x_k \oplus \hat{x}_{k+1}, \end{aligned}$$

This observation gives a nice and simple way to compute $\hat{x}$ from x: indeed, let $x \downarrow k$ be the number of 1's in x preceding position k (i.e., the number of 1's in bits that are not less significant than the k-th). Then

$$\hat{x}_k = x \downarrow k \bmod 2. \tag{1}$$

3 On Gray ordering and graphs

An obvious application of Gray ordering to graphs is that of permuting node labels so that the resulting adjacency matrix changes "slowly" from row to row. Indeed, intuitively if we permute the rows of the adjacency matrix following a Gray ordering rows with a small number of changes should appear nearby. (This intuition is only partially justified—e.g., in Table 1 the first and last word of the second column differ by just one bit, yet they are as far apart as possible). Of course, to maintain correctly the graph we also need to permute columns in the same way: but this process will not change the number of differences between adjacent rows.

Indeed, already some of the first investigation of web graph compression experimented with Gray orderings [1][6]. However, the authors reported a very small improvement in compression with respect to URL ordering; this fact, coupled with the obvious advantages of the latter, pushed the authors to discard Gray ordering altogether. The main question we try to answer in this paper is that this small difference is actually an absolute property or it is an artifact strongly depending on the compression algorithms used, and whether it also applies to transposed graphs.

To this purpose, we first develop a very simple algorithm that makes it possible to decide the Gray code ordering inspecting the successor lists in parallel.

When manipulating web graphs using the WebGraph framework, successors are returned under the form of an iterator providing an increasing sequence of integers. This makes it possible

[6]Note that in the paper the codes are incorrectly spelt as "Grey".

to compare the position in the Gray ordering of two rows of the adjacency matrix by iterating in parallel over the adjacency lists of two nodes. Until the lists coincide, we skip, and keep a variable recording the parity of the number of arcs seen so far (note that the value of formula (1) depends only on the *parity* of $x \downarrow k$). As soon as the lists differ, we can use formula (1) to compute the first different bit of the ranks of the adjacency rows in the Gray ordering: assuming the first list returns j and the second list returns k (for sake of simplicity, we assume that the end-of-list is marked by ∞), we have the following scenario:

- if the parity is odd, the order of the lists is the order of j and k;
- if the parity is even, the order of the lists is the order of j and k *reversed*.

This can be easily seen as $j < k$ implies that the first difference in the rows of the adjacency matrix is at position j, where the first list has a one whereas the second list has a zero. If the parity is odd, this means that the rank of the first list has a zero in position j, whereas the rank of the second list has a one. The situation is reversed if the parity is even. Algorithm 1 describes formally this process.

Algorithm 1 The procedure for deciding the Gray natural order of two rows of the adjacency matrix, represented by means of iterators i and j that return the position of the next nonzero element. Note that end-of-list is denoted by ∞ and that we used Iverson's notation: $[a < b]$ has value one if $a < b$, zero otherwise. The meaning of the return value is the same as that of the C standard function `strcmp`.

```
0   p ← false;
1   forever begin
2     a ← next(i);
3     b ← next(j);
4     if a = ∞ and b = ∞ then return 0;
5     if a ≠ b then begin
6       if p ⊕ [a < b] then return 1
7       else return −1
8     end;
9     p ← ¬p
10  end;
```

Once this simple consideration is made, it is trivial to implement Gray code (or lexicographic) graph permutation using WebGraph's facilities. The idea is that of using a standard exchange-based sorting algorithm using a lazy comparator based on the considerations above. As a result, we can compute the Gray permutation of the `uk` graph (see Table 2) in about one hour on an Opteron at 2.8 GHz.

Note that from a complexity viewpoint this approach is far from optimal. Indeed, a simple way to permute words in Gray code ordering is to apply a modified radix sort in which, at each recursive call, we have a parity bit that tells us whether $0 \prec 1$ or $1 \prec 0$. We apply a standard

Name	Year	Nodes	Edges
cnr	2000	325 557	3 216 152
webbase	2001	118 142 155	1 019 903 190
it	2004	41 291 594	1 150 725 436
eu	2005	862 664	19 235 140
uk	2007	105 896 555	3 738 733 648

Table 2: Basic properties of graphs used as dataset.

radix sort algorithm, dividing words in two blocks depending on the first bit, and then recurse on each block: however, when we recurse on the block of words starting with one, we invert the parity bit.

Now, this approach is theoretically optimal *if measured of the space occupancy of the adjacency matrix*. However, the adjacency matrix of a web graph is very sparse, and never represented explicitly.

Alternatively, we could develop a radix sort that picks up lazily successors from successor lists (for all nodes) and deduces implicitly the zeroes and the ones of the adjacency matrix. Albeit in principle such an algorithm would iterate optimally (i.e., it would extract from each iterator the minimum number of elements that are necessary to compute the ordering), it would require to build at the same time the iterators for *all* nodes—a task that would require a preposterous amount of core memory.

4 Experiments

We ran a number of experiments, on web graphs of different sizes (ranging from 300K nodes up to almost 120M nodes) and collected at different times, and on their transpose version (i.e., the graph obtained reversing the direction of all arcs). The graphs used are described in Table 2, and they are all publicly available, as well as the code used in the experiments.

We started with the standard URL ordering of nodes and permuted the nodes in different ways, taking note of the number of bits/link occupied if the graph is compressed in WebGraph format. Six node orderings were considered:

- URL: URL ordering (to avoid confusion, we use "URL ordering" instead of "URL lexicographic ordering");
- lex: lexicographic row ordering;
- Gray: Gray ordering;
- lhbhGray: loose host-by-host Gray ordering (i.e., keep URLs from the same host adjacent, and order them using Gray ordering);
- shbhGray: strict host-by-host Gray ordering (like before, but Gray ordering is applied considering only local links, i.e., links to URLs of the same host).

We remark that we devised the latter two orderings trying to combine external and internal information.

Figure 1 shows how the adjacency matrix changes when re-ordering is applied. To obtain this figure, we actually divided the matrix into smaller 100×100-square submatrices, computed the fraction of 1's found in each submatrix, and plotted a square using a grayscale that maps 0 to white and that becomes exponentially darker (1 is black).

WebGraph compression depends not only on the graph to be compressed and the ordering of its nodes, but also on a number of other parameters that control its behaviour. Two parameters that happen to be important for our experiments are:

- Window size: when compressing a certain matrix row, WebGraph compares it with a number of previous rows, and tries to compress it differentially with respect to (i.e., as a difference from) each such row, choosing at the end the row that gave the best compression (or none, if representing the row non-differentially gives better compression); the number of rows considered in this process is called the *window size*. Of course, larger window sizes produce slower compression, but usually guarantee better compression: the default window size is 7.

- Maximum reference count: Compressing a row x differentially with respect to some previous row $y < x$ makes it necessary, at access time, to decompress row y before decompressing row x; if also y is compressed differentially with respect to some other row $z < x$, z must be decompressed first, and so on, producing what we call a reference chain. This recursive process must be somehow limited, or access becomes extremely inefficient (and may even overflow the recursion stack). The *(maximum) reference count* is the maximum length of a reference chain (we simply avoid to compress a row differentially with respect to one that already has a maximum-length reference chain); the default reference count is 3, but this value may be pushed up to ∞ (meaning that we just don't care of creating too long chains: this makes sense if we plan to access the graph only sequentially, in which case we just need to keep the last w uncompressed rows, where w is the window size used for compression).

Tables 3 and 4 show the number of bits/link occupied by the various graphs using ∞ and 3, respectively, as reference count (the window size was fixed at 8). Not surprisingly, the number of bits/link for the random permutation, that we present here only for comparison, is very large.

A more thorough, and visually understandable, presentation of the experimental results in shown in Figure 2.

5 Discussion

Our experiments highlight a number of very interesting issues.

- **The effectiveness of intrinsic methods depends on the graph.** As we discussed, the fact that lex or Gray ordering should improve compression is intuitive, but has little formal justification. Indeed, on some graphs we have a visible decrease in compression.

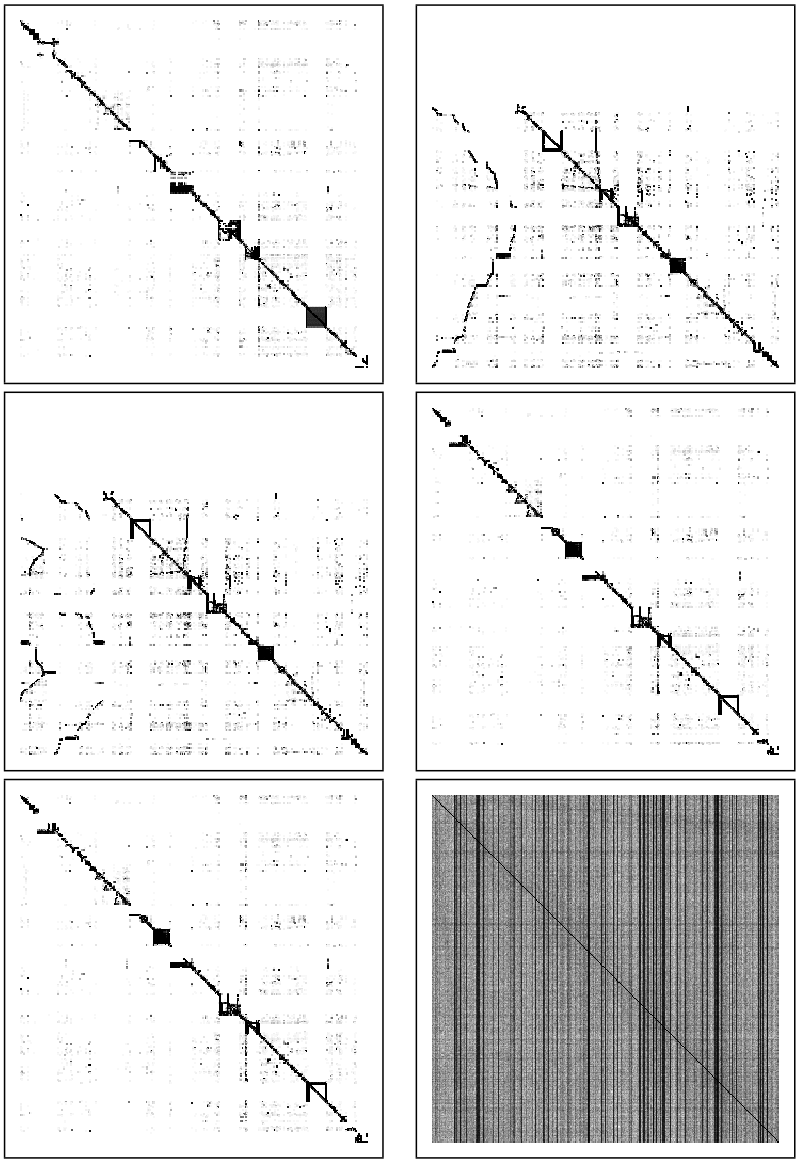

Figure 1: The picture shows the adjacency matrix of the `cnr` graph (325 557 nodes), when nodes are ordered as follows (left-to-right, top-to-bottom): lexicographically by URL, lexicographically by row, Gray ordering, loose host-by-host Gray ordering, strict host-by-host Gray ordering, randomly.

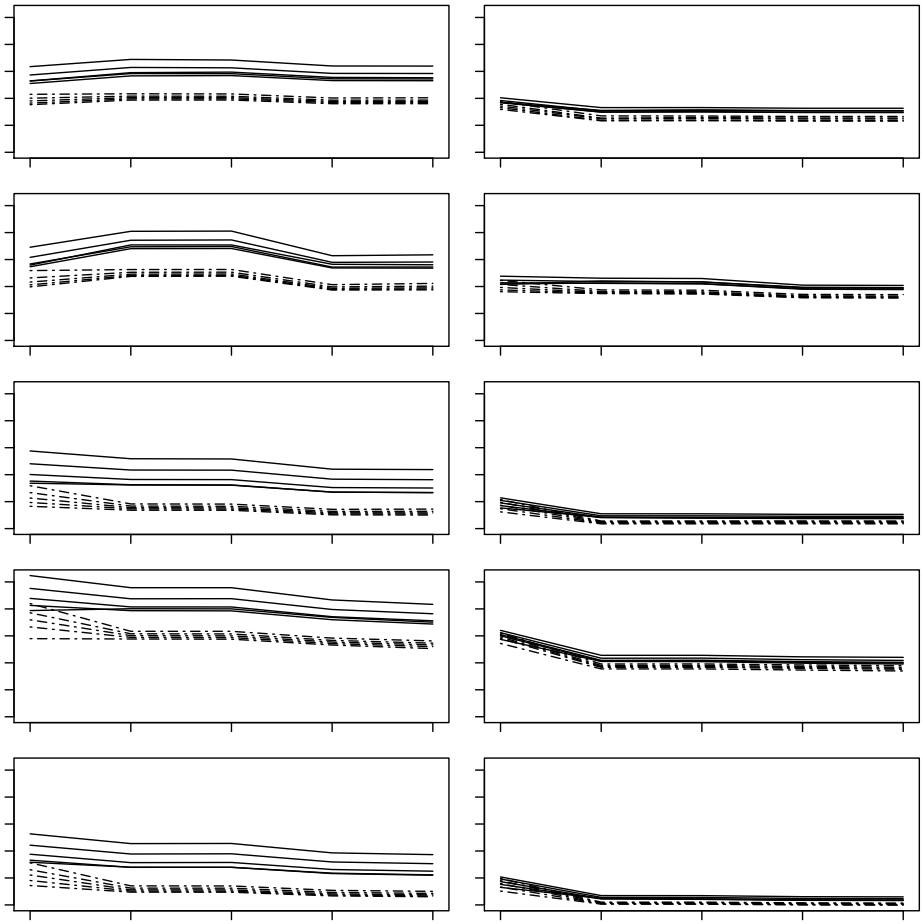

Figure 2: The picture shows the bits/link compression rate for various node orderings and compression parameters. Each row of two boxes corresponds (top to bottom) to one of the graphs in the dataset (see Table 2), the box on the left relating to the graph itself and the one on the right to its transpose. In every box, the abscissa corresponds to different orderings and the ordinate to the number of bits per link; the considered orderings are (left to right): URL, lex, Gray, lhbhGray, and shbhGray. Note that the random ordering is not shown since its compression rate is an order of magnitude larger than the one of other orderings. Every group of five lines corresponds to different values of the maximum number of (maximum) reference count: the above (continous) one has such parameter set to 3, whereas the below (dashed) one has such parameter set to ∞. In every group of five lines, each line corresponds to different values of the window size parameter; from top to bottom, the values are 1, 2, 4, 8, 16.

Graph	URL	lex	Gray	shbhGray	lhbhGray	Random
cnr	2.823	2.981	2.983	2.845	2.843	17.986
	2.654	2.185	2.192	2.176	2.177	15.084
webbase	3.059	3.410	3.416	2.907	2.895	30.937
	2.876	2.753	2.740	2.589	2.598	28.236
it	1.969	1.733	1.723	1.541	1.545	26.430
	1.737	1.206	1.207	1.206	1.209	21.717
eu	4.331	3.944	3.938	3.600	3.715	19.859
	3.903	2.832	2.833	2.761	2.795	16.445
uk	1.906	1.513	1.509	1.332	1.367	27.576
	1.662	1.042	1.040	1.007	1.014	21.682

Table 3: Compression rate summary, window size set to 8, and (maximum) reference count set to ∞.

- **Intrinsic methods are extremely effective on transposed graphs.** Our data for standard web graphs confirm what has been previously reported [1], but our new data for transposed graphs show that here the situation is reversed: intrinsic methods improve very significantly the compression of transposed web graphs. With infinite reference chains, the transpose of uk requires just *one bit per link*. This is a phenomenon that clearly needs a combinatorial explanation.

  We modified WebGraph so that we could get access to detailed statistics about which compression technique is responsible for the increase in compression. Moving from URL to Gray ordering, the number of *copied* arcs of the uk graph (arcs that are not recorded explicitly, but rather represented differentially) jumps from $\approx 2\,\mathrm{G}$ to $\approx 3\,\mathrm{G}$. Thus, more than 80 % of the arcs of the graph is not represented explicitly. Another visible effect is that of shifting the distribution of *gaps*—very small values (say, below 10) are much more frequent, which also increases compression.

  We believe that such a major improvement in the transpose depends on the repetition in patterns of predecessors being much more frequent than in patterns of successors. For instance, all pages of level k are often pointed to by pages of level $k+1$ (e.g., general topics of a site). This makes the predecessor list of level-k pages large and very similar. The same phenomenon does not happen for successors because usually at level k the pointers at level $k+1$ are distinct. Moreover, URL ordering does not gather level k pages nearby—rather, it sorts URLs so that a level k page is followed by all its subpages. On the contrary, manual inspection of the URL permutation induced by strict host-by-host Gray ordering shows that pages on the same level of the hierarchy are grouped together.

- **Mixed methods work better.** Essentially in all cases, our new orderings outperform old methods.

6 Conclusions

We have presented some experiments about the effect of permuting nodes on web graph compression. Albeit our results are clearly preliminary, they highlight a number of issues that have not been tackled in the literature. First of all, we have provided two new permutations that outperform all previous methods. Second, we have shown that transposed graphs behave in a radically different manner when permuted with our techniques, giving rise to extreme compression rates.

References

[1] Randall, K., Stata, R., Wickremesinghe, R., Wiener, J.L.: The LINK database: Fast access to graphs of the Web. Research Report 175, Compaq Systems Research Center, Palo Alto, CA (2001)

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

[3] Kumar, R., Raghavan, P., Rajagopalan, S., Sivakumar, D., Tompkins, A., Upfal, E.: The Web as a graph. In: PODS '00: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM Press (2000) 1–10

[4] Bharat, K., Broder, A., Henzinger, M., Kumar, P., Venkatasubramanian, S.: The Connectivity Server: fast access to linkage information on the Web. Computer Networks and ISDN Systems **30**(1-7) (1998) 469–477

[5] Blandford, D.K., Blelloch, G.E.: Index compression through document reordering. In: Data Compression Conference, IEEE Computer Society (2002) 342–351

[6] Shieh, W.Y., Chen, T.F., Shann, J.J.J., Chung, C.P.: Inverted file compression through document identifier reassignment. Inf. Process. Manage **39**(1) (2003) 117–131

[7] Silvestri, F.: Sorting out the document identifier assignment problem. In Amati, G., Carpineto, C., Romano, G., eds.: Advances in Information Retrieval, 29th European Conference on IR Research, ECIR 2007, Rome, Italy, April 2-5, 2007, Proceedings. Volume 4425 of Lecture Notes in Computer Science., Springer (2007) 101–112

[8] Blanco, R., Barreiro, A.: Document identifier reassignment through dimensionality reduction. In Losada, D.E., Fernández-Luna, J.M., eds.: Advances in Information Retrieval, 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005, Proceedings. Volume 3408 of Lecture Notes in Computer Science., Springer (2005) 375–387

[9] Knuth, D.E.: The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations (Art of Computer Programming). Addison-Wesley Professional (2005)

Graph	URL	lex	Gray	shbhGray	lhbhGray	Random
cnr	3.551	3.833	3.844	3.654	3.659	18.008
	2.839	2.489	2.495	2.472	2.474	15.084
webbase	3.732	4.404	4.409	3.680	3.688	30.937
	3.092	3.132	3.105	2.902	2.916	28.236
it	2.763	2.626	2.618	2.334	2.353	26.430
	1.852	1.407	1.393	1.378	1.379	21.717
eu	5.130	4.935	4.927	4.438	4.599	19.872
	4.002	3.063	3.072	2.993	3.019	16.445
uk	2.659	2.394	2.396	2.101	2.163	27.576
	1.761	1.222	1.205	1.162	1.170	21.682

Table 4: Compression rate summary, window size set to 8, and (maximum) reference count set to 3.