The WebGraph Framework: Compression Techniques

Paolo Boldi Sebastiano Vigna
DSI, Università di Milano, Italy

"The" Web graph

- Given a set U of URLs, the *graph* induced by U is the directed graph having U as set of nodes, and an arc from x to y iff the page with URL x has a link that points to URL y.
- The *transposed graph* can be obtained by reversing all arcs.
- The *symmetric graph* can be obtained by "forgetting" the arc orientation.
- The Web graph is *huge*.

What does it mean...

... "to store (part of) the Web graph"?

- Being able to know the successors of each node (the successors of x are those nodes y for which an arc $x \to y$ exists);
- this must be happen in a reasonable time (e.g., much less than 1 ms/link);
- having a simple way to know the node corresponding to a URL (e.g., minimal perfect hash).
- having a simple way to know the URL corresponding to a node (e.g., front-coded lists).

We shall denote all nodes using natural numers ($0, 1, \ldots, n-1$, where $n = |U|$).

Why...

...to store the Web graph?

- Many algorithms for ranking and community discovery require visits of the Web graph;
- Web graphs offer real-world examples of graphs with the *small-world* property, and as such they can be used to perform experiments to validate small-world theories.
- Web graphs can be used to validate Web graph models (not surprisingly).
- It's fun.
- It provides new, challenging mathematical and algorithmic problems.

WebGraph is...

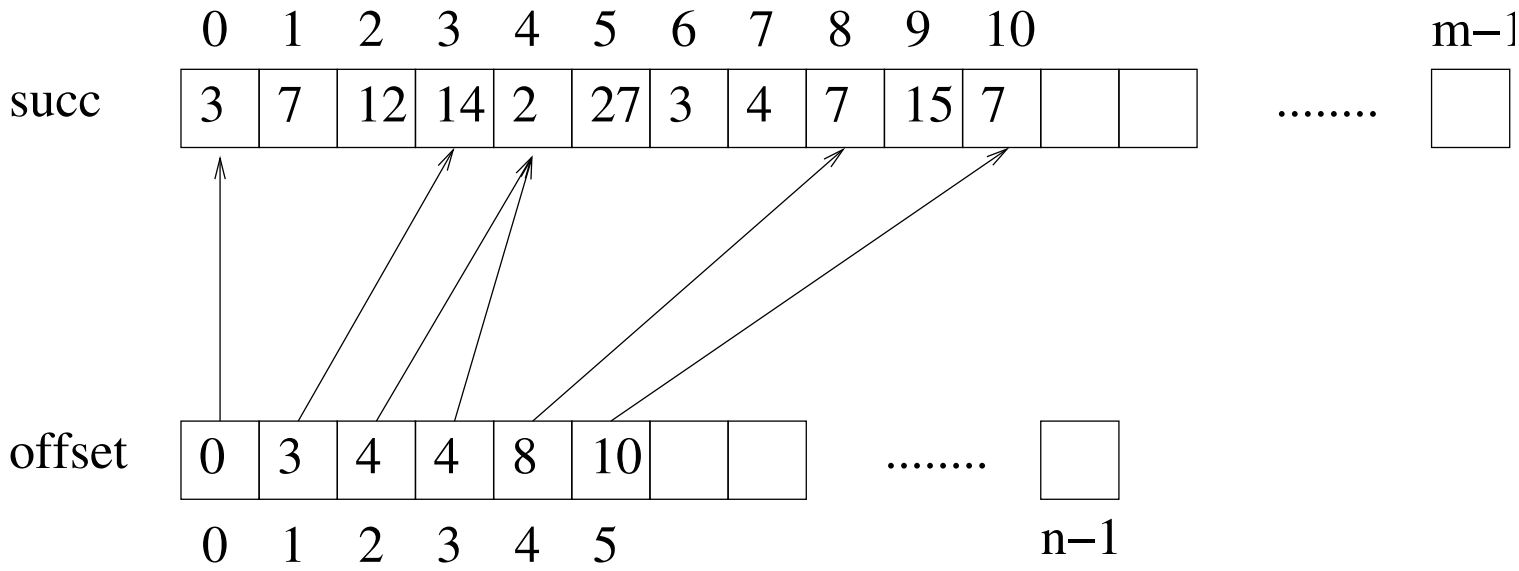
- Algorithms for compressing and accessing Web graphs.
- New instantaneous codes for distributions commonly found when compressing Web graphs.
- Java documented reference implementation (Gnu GPL'd) of the above (`http://webgraph.dsi.unimi.it/`).
- Freely available large graphs.
- Few such collections are publicly available, and, as a matter of fact, WebGraph was ./'d when it went public.

Previous history

- Connectivity Server (Bharat, Broder, Henzinger, Kumar, and Venkatasubramanian), ≈ 32 bits/link.
- LINK database (Randall, Stata, Wickremesinghe, and Wiener), ≈ 4.5 bits/link.
- WebBase (Raghavan and Garcia–Molina), ≈ 5.6 bits/link.
- Suel and Yuan, ≈ 14 bits/link.
- Theoretical analysis and experimental algorithms (Adler and Mitzenmacher), ≈ 10 bits/link.
- Algorithms for separable graphs (Blandford, Blelloch, Kash), ≈ 5 bits/link.

Currently, WebGraph codes at ≈ 3 bits/link.

Naïf representation



The offset vector tells us from where successors of a given node start. Implicitly, it contains the outdegree of the node.

First simple idea

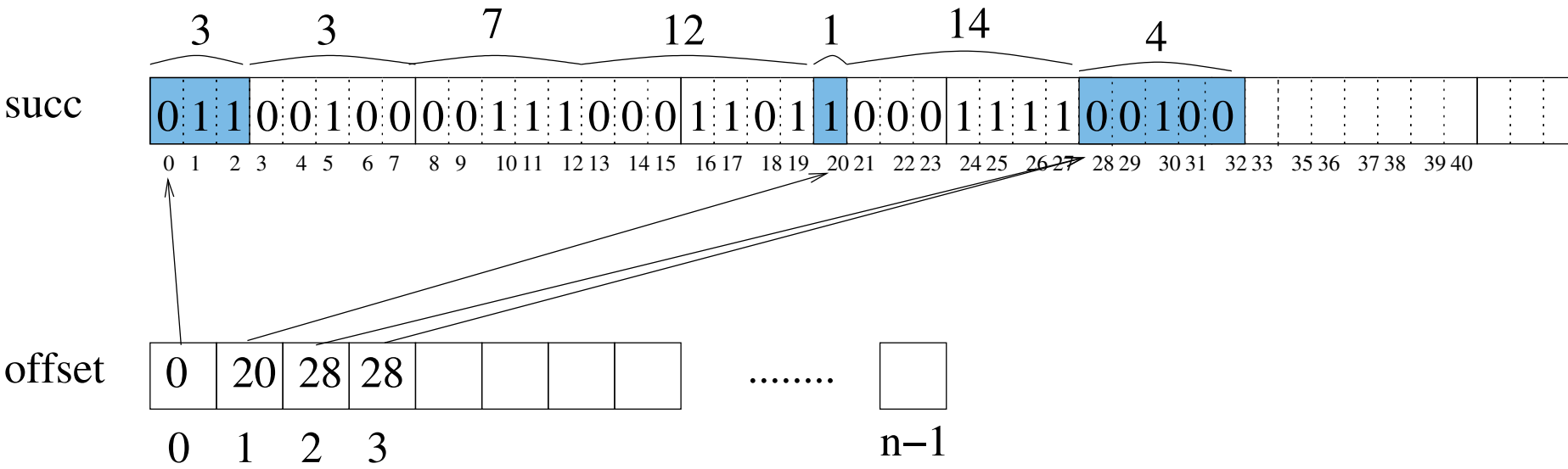
Use a variable-length representation, choosing it so that

- it is easy to decode;
- minimises the expected length.

And the offsets?

- bit displacement vs. byte displacement (with alignment)
- we must express explicitly the outdegree.

Variable-length representation



Variable-length representations are a basic technique in full-text indexing.

Instantaneous codes

- An *instantaneous code* for S is a mapping $c : S \to \{0,1\}^*$ such that for all x, $y \in S$, if $c(x)$ is a prefix of $c(y)$, then $x = y$.
- Let ℓ_x be the length in bits of $c(x)$.
- A code with lengths ℓ_x has *intended distribution*

$$p(x) = 2^{-\ell_x}.$$

- The choice of the code depends, of course, on the data distribution.

Unary coding

- If $S = \mathbf{N}$, we can represent $x \in S$ writing x zeroes followed by a one.
- Thus $\ell_x = x + 1$, and the intended distribution is

$$p(x) = 2^{-x-1} \quad \text{geometric distribution.}$$

0	1
1	01
2	001
3	0001
4	00001

γ coding

The γ *coding* of $x \in \mathbf{N}^+$ can be obtained by writing the index of the most significant bit of x in unary, followed by x (stripped of the MSB) in binary.
Thus

$$\ell_x = 1 + 2\lfloor \log x \rfloor \quad \Longrightarrow \quad p(x) \propto \frac{1}{2x^2} (\text{Zipf})$$

1	**1**
2	**01**0
3	**01**1
4	**001**00
5	**001**01

Degrees have a Zipf distribution with exponent ≈ 2.7: use γ!

Successors & locality

- Since many link are *navigational*, the URLs they point to share a large prefix.
- Thus, if we order lexicographically URLs, for many arcs $x \to y$ often $|x - y|$ will be small.
- So, we represent the successors $y_1 < y_2 < \cdots < y_k$ using their *gaps*

$$y_1 - x, y_2 - y_1 - 1, \ldots, y_k - y_{k-1} - 1$$

which are distributed as a Zipf with exponent ≈ 1.2.
- Commonly used: *variable-length nibble coding*, a list of 4-bit blocks whose MSB specifies whether the list has ended (it is redundant).
- WebGraph uses by default ζ_k, a new family of non-redundant codes with intended distribution close to a Zipfian with exponent < 1.6 (ζ_3 is the default choice).

Similarity

URL that are close in lexicographic order are likely to have similar successor lists, as they belong to the same site, and probably to the same level of the site hierarchy. So, we code a list by *referentiation*:

- an integer r (reference): if $r > 0$, the list is described as a difference from the list of $x - r$: a bit string tells us which successors must be copied, and which not;
- a list of *extra nodes*, for the remaining nodes.

Referentiation: an example

Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

Node	Outd.	Ref.	Copy list	Extra nodes
...	...	...	...	...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	01110011010	22, 316, 317, 3041
17	0			
18	5	3	11110000000	50
...	...	...	...	...

Differential compression

WebGraph pushes much farther this idea: we code use a list of *copy blocks*, which specify by inclusion/exclusion the sublists that must be alternatively copied or discarded.

Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

Node	Outd.	Ref.	# blocks	Copy blocks	Extra nodes
...	...	...	...	...	...
15	11	0			13, 15, 16, 17, 18, 19, 23, ...
16	10	1	7	0, 0, 2, 1, 1, 0, 0	22, 316, ...
17	0				
18	5	3	1	4	50
...	...	...	...	...	...

Consecutivity

- WebGraph exploits the fact that many links within a page are consecutive (with respect to the lexicographic order). This is due to at least two distinct phenomena.
- First of all, most pages contain sets of navigational links which point to a fixed level of the hierarchy.
- Second, in the transposed Web graph pages that are high in the site hierarchy (e.g., the home page) are pointed to by most pages of the site.
- More in general, *consecutivity is the dual of distance-one similarity*. If a graph is easily compressible using similarity at distance one, its transpose must sport large intervals of consecutive links, and viceversa.

Intervalisation

To exploit consecutivity, WebGraph uses a special representation for extra nodes.

- if there are enough large intervals, they are coded using their left extreme and their length;
- the remaining extra nodes, called *residuals*, are represented separately.

Intervalisation: an example

Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

Node	Outd.	Ref.	# bl.	Copy bl.s	# int.	Lft extr.	Lth	Residuals
...	...	...	...	...	...	...	...	...
15	11	0			2	0, 2	3, 0	5, 189, 111, 718
16	10	1	7	0, 0, ...	1	600	0	12, 3018
17	0							
18	5	3	1	4	0			50
...	...	...	...	...	...	...	...	...

Choices in the reference scheme

- How do you choose the reference node for x?
- You consider the successor lists of the last W nodes, but... you do not consider lists which would cause a recursive reference of more than R chains.
- The parameter R is essential for deciding the ratio compression/speed. W essentially decreases compression time only.

Implementation

- Random access to successor lists is implemented *lazily* through a cascade of *iterators*.
- Each series of interval and each reference cause the creation of an iterator; the same happens for references.
- The results of all iterators are then merged.
- The advantage of laziness is that we never have to build an actual list of successors in memory, so the overhead is limited to the number of *actual reads*, not to the number of successors lists that would be necessary to re-create a given one.

Access speed

- Access speed to a compressed graph is commonly measured in the time required to access a link (≈ 300 ns for WebGraph).
- This quantity, however, is strongly dependent on the architecture (e.g., cache size), and, even more, on low-level optimisations (e.g., hard-coding of the first codewords of an instantaneaous code).
- To compare speeds reliably, we need *public data*, that anyone can access, and a *common framework* for the low-level operations.
- A first step is `http://webgraph-data.dsi.unimi.it/`. We provide freely available data to compare compression techniques.

Conclusions

- WebGraph combines new codes, new insights on the structure of the Web graph and new algorithmic techniques to achieve a very high compression ratio, while still retaining a good access speed (but it could be better).
- Our software is highly tunable: you can experiment with dozens of codes, algorithmic techniques and compression parameters, and there is a large unexplored space of combinations.
- A theoretically interesting question is how to combine optimally differential compression and intervalisation: we do not know whether is current greedy approach (first copy as much as you can, then intervalise) is necessarily the best one.