(Software Heritage) Graph Compression

in 5 minutes!

. . . ok, maybe 10

Thibault Allançon, Stefano Zacchiroli

Software Heritage

17 July 2019
Software Heritage, Inria
Paris, France

Context — webgraph compression

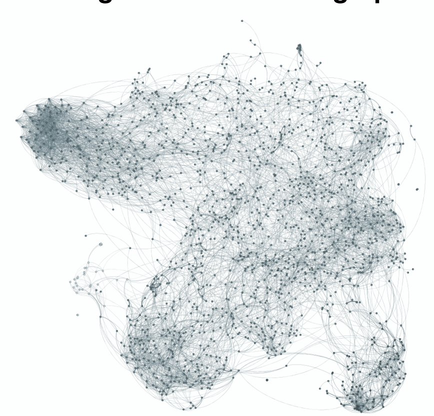
Reusing slides shamelessly from:

Giulio Ermanno Pibiri
Effective Web Graph Representations, 2018
http://pages.di.unipi.it/pibiri/slides/webgraphs_compression.pdf

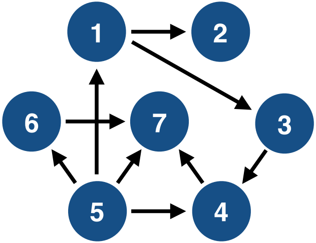
Context - Web Graphs

Web graphs are directed graphs of pages pointing to other pages on the Web.

We focus on compression effectiveness on **large real-world Web graphs**.



Context - Web Graphs



Conceptual graph

0	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	1
1	0	0	1	0	1	1
0	0	0	0	0	0	1
0	0	0	0	0	0	0

Adjacency **matrix**

1: 2,3
2: -
3: 4
4: 7
5: 1,4,6,7
6: 7
7: -

Adjacency **lists**

Many results are known for compressing integer sequences.

The WebGraph Framework

Java/C++ framework consisting in algorithms and compression codes for managing large Web Graphs.

http://webgraph.di.unimi.it/

The WebGraph Framework I: Compression Techniques, Boldi-Vigna, WWW 2004

Locality - pages links to pages whose URL is lexicographically similar. URLs share long common prefixes.

Use *d-gap* compression.

Similarity - pages that are close together in lexicographic order, tend to have many common successors.

Use *reference* compression.

The WebGraph Framework

Exploiting **locality**.

If we have: x: $[y_1,\ldots,y_k]$, then we represent
$[y_1 - x, y_2 - y_1 - 1, y_3 - y_2 - 1, \ldots, y_k - y_{k-1} - 1]$

First gap $d = y_1 - x$ is represented as $2d$ if $d \geq 0$ or $2|d|-1$ if $d < 0$

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
...	...	...

Adjacency lists

Node	Outdegree	Successors
...	...	...
15	11	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0	
18	5	9, 1, 0, 0, 32
...	...	...

***d*-gapped** adjacency lists

The WebGraph Framework

Exploiting **similarity**.

Idea: use reference compression, i.e., represent a list with respect to another one called its **reference list**.

	Node	Outdegree	Successors
	...	...	...
1	15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
2	16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
3	17	0	
4	18	5	13, 15, 16, 17, 50
	...	...	...

Adjacency lists

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
...	...	...	...	...

Copy lists

The WebGraph Framework

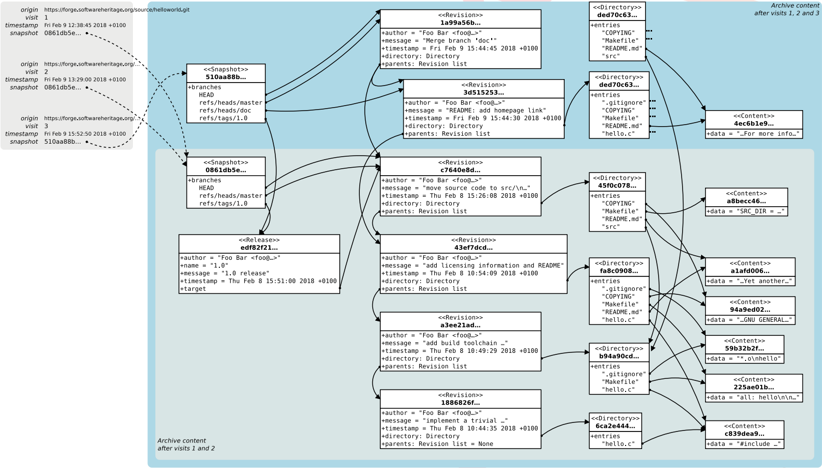
`.uk`
18.5 million pages
300 million links

R	Bits/link		
	$W = 1$	$W = 3$	$W = 7$
∞	2.75	2.38	2.22
3	3.87	3.25	3.00
1	5.05	3.91	3.46

`WebBase`
118 million pages
1 billion links

R	Bits/link		
	$W = 1$	$W = 3$	$W = 7$
∞	3.59	3.22	3.08
3	4.46	3.92	3.74
1	5.40	4.49	4.17

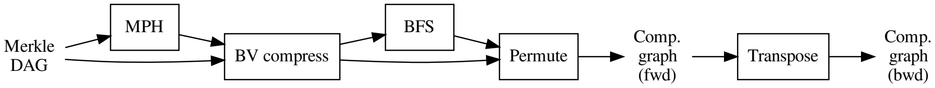
Software Heritage graph



- 12 B nodes, 165 B edges (MSR 2019 dataset snapshot)
  - in-memory size (naive): 165 B × 2 nodes × 64 bits = 2.5 TiB
- will it ~~blend~~ compress?

Software Heritage graph compression

Pipeline



Resources needed

	Timings	Max mem usage
MPH	3h30	10GB
BV Compress	103h	15GB
BFS	10h	1057GB
Permute	24h40	115GB
Stats	4h	102GB
Transpose	21h30	19GB
Total	**167h** (7 days)	**1TB**

Software Heritage graph compression — results

And the winner is. . .

- bit/edge: 4.913
- compressed size
  - forward: 101 GiB
  - backward: 94 GiB
- compression ratio: 3.95% w.r.t. naive in-memory graph (!)

In less than 200 GiB of RAM we can have the full structure of the Software Heritage Merkle DAG, browsable in both directions, with in-memory read performances.

Software Heritage graph compression — results

And the winner is. . .

- bit/edge: 4.913
- compressed size
  - forward: 101 GiB
  - backward: 94 GiB
- compression ratio: 3.95% w.r.t. naive in-memory graph (!)

In less than 200 GiB of RAM we can have the full structure of the Software Heritage Merkle DAG, browsable in both directions, with in-memory read performances.

Software Heritage graph compression — results

And the winner is. . .

- bit/edge: 4.913
- compressed size
  - forward: 101 GiB
  - backward: 94 GiB
- compression ratio: 3.95% w.r.t. naive in-memory graph (!)

In less than 200 GiB of RAM we can have the full structure of the Software Heritage Merkle DAG, browsable in both directions, with in-memory read performances.

swh-graph

New SWH software component: swh-graph

- https://forge.softwareheritage.org/source/swh-graph/repository/master/
- Phabricator tag “Graph service”

Contains:

- Docker-ized compression pipeline
  - input: .nodes/.edges text files (SWH PIDs)
  - output: WebGraph comp. output + mappings SWH PID ↔ long
- Java server: bridge Webgraph ↔ REST API
- Python client for the REST API

swh-graph — use cases and API

Browsing

- **`ls`**

  `/graph/neighbors/:DIR_ID?edges=dir:cnt,dir:dir`

- **`git log`**

  `/graph/visit/nodes/:REV_ID?edges=rev:rev`

Vault

- **tarball** (i.e., `ls -R`)

  `/graph/visit/paths/:DIR_ID?edges=dir:cnt,dir:dir`

- **`git bundle`**

  `/graph/visit/nodes/:NODE_ID?edges=*`

swh-graph — use cases and API

Browsing

- **ls**

  `/graph/neighbors/:DIR_ID?edges=dir:cnt,dir:dir`

- **git log**

  `/graph/visit/nodes/:REV_ID?edges=rev:rev`

Vault

- **tarball** (i.e., `ls -R`)

  `/graph/visit/paths/:DIR_ID?edges=dir:cnt,dir:dir`

- **git bundle**

  `/graph/visit/nodes/:NODE_ID?edges=*`

swh-graph — use cases and API (cont.)

Provenance

- **commit provenance** (one commit)

  ```
  /graph/walk/:NODE_ID/rev?direction=backward\
      &edges=dir:dir,cnt:dir,dir:rev
  ```

- **commit provenance** (all commits)

  ```
  /graph/leaves/:NODE_ID?direction=backward\
      &edges=dir:dir,cnt:dir,dir:rev
  ```

- **origin provenance** (one origin)

  ```
  /graph/walk/:NODE_ID/ori?direction=backward&edges=*
  ```

- **origin provenance** (all origins)

  ```
  /graph/leaves/:NODE_ID?direction=backward&edges=*
  ```

swh-graph — use cases and API (cont.)

Provenance

- **commit provenance** (one commit)
  ```
  /graph/walk/:NODE_ID/rev?direction=backward\
      &edges=dir:dir,cnt:dir,dir:rev
  ```
- **commit provenance** (all commits)
  ```
  /graph/leaves/:NODE_ID?direction=backward\
      &edges=dir:dir,cnt:dir,dir:rev
  ```

- **origin provenance** (one origin)
  ```
  /graph/walk/:NODE_ID/ori?direction=backward&edges=*
  ```
- **origin provenance** (all origins)
  ```
  /graph/leaves/:NODE_ID?direction=backward&edges=*
  ```

Next steps

- T1867: compress Merkle DAG + origin nodes (in progress)
- T1885: benchmark use cases; goal 2–3μs per /neighbors
- T1884: (high-level) Python bindings
- T1851: integrate swh-graph into swh-environment
- fully automated pipeline, including graph export (T1847)
- where/how to host swh-graph in production?

- storing the entire SWH Merkle DAG in memory is doable
- tooling to do so and exploit the result is now in swh-graph
- huge potential to speed up existing needs and enable new ones

Thanks!

Questions?

Thibault Allançon, Stefano Zacchiroli
haltode@gmail.com, zack@upsilon.cc

about the slides:
available at https://upsilon.cc/~zack/talks/2019/2019-07-17-5min-graph-compression.pdf
© 2019 Stefano Zacchiroli
license CC BY-SA Creative Commons Attribution-ShareAlike 4.0 International License