Ultra-Large-Scale Repository Analysis via Graph Compression

Stefano Zacchiroli
zack@irif.fr
@zacchiro
joint work with Paolo Boldi, Antoine Pietri, and Sebastiano Vigna

Université de Paris & Inria, France

19 February 2020
SANER 2020: 27th Intl. Conf. on Software Analysis, Evolution and Reengineering
London, ON, Canada

Motivations

- Free/Open Source Software (FOSS) + social coding (GitHub, GitLab, ...) = massive amount of data for empirical software engineering (ESE)
- software evolution and clone detection have vastly benefited from it

Motivations

- Free/Open Source Software (FOSS) + social coding (GitHub, GitLab, ...) = massive amount of data for empirical software engineering (ESE)
- software evolution and clone detection have vastly benefited from it

An ESE growth crisis?

- GitHub alone: ~100 M repositories
- exponential growth rate, doubling every ~2 years (Rousseau et al., 2009)
- possibly the tip of the iceberg w.r.t. the rise of distributed forges and non-public collaborative development (cf. *inner source*)

Motivations

- Free/Open Source Software (FOSS) + social coding (GitHub, GitLab, …)
  = massive amount of data for empirical software engineering (ESE)
- software evolution and clone detection have vastly benefited from it

An ESE growth crisis?

- GitHub alone: ~100 M repositories
- exponential growth rate, doubling every ~2 years (Rousseau et al., 2009)
- possibly the tip of the iceberg w.r.t. the rise of distributed forges and non-public collaborative development (cf. *inner source*)

Current mitigation approaches

- scale-out analysis: not always applicable, expensive
- sampling: (e.g., top-starred repos) prone to selection bias and external validity issues

Research question

Is it possible to efficiently perform software development history analyses at ultra large scale, on a single, relatively cheap machine?

Research question

Is it possible to efficiently perform software development history analyses at ultra large scale, on a single, relatively cheap machine?

- development history: all information captured by state-of-the-art Version Control Systems (VCS)

Research question

Is it possible to efficiently perform software development history analyses at ultra large scale, on a single, relatively cheap machine?

- development history: all information captured by state-of-the-art Version Control Systems (VCS)
- cheap machine: commodity hardware, desktop- or server-gread, few kUSD of investment

Research question

Is it possible to efficiently perform software development history analyses at ultra large scale, on a single, relatively cheap machine?

- development history: all information captured by state-of-the-art Version Control Systems (VCS)
- cheap machine: commodity hardware, desktop- or server-gread, few kUSD of investment
- ultra large scale: in the ballpark of (the known extent of) all publicly available software source code

- our proxy for publicly available software:

Software Heritage
THE GREAT LIBRARY OF SOURCE CODE

- both source code and its development history as captured by VCS
- coverage:
  - all public repositories from GitHub and GitLab.com
  - historical forges: Google Code, Gitorious
  - package manager repositories: NPM, PyPI, Debian
- 90 M repositories, 5.5 B unique files, 1.1 B unique files (data dump: 2018-09-25)
- available as offline dataset

Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli
The Software Heritage Graph Dataset: Public software development under one roof
MSR 2019: 16th Intl. Conf. on Mining Software Repositories. IEEE

(Web) graph compression

Definition (The graph of the Web)

Directed graph that has Web pages as nodes and hyperlinks between them as edges.

Properties (1)

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

$\longrightarrow$ use D-gap compression

Adjacency lists

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
...	...	...

D-gapped 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
...	...	...

(Web) graph compression (cont.)

Definition (The graph of the Web)

Directed graph that has Web pages as nodes and hyperlinks between them as edges.

Properties (2)

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

$\longrightarrow$ use reference compression

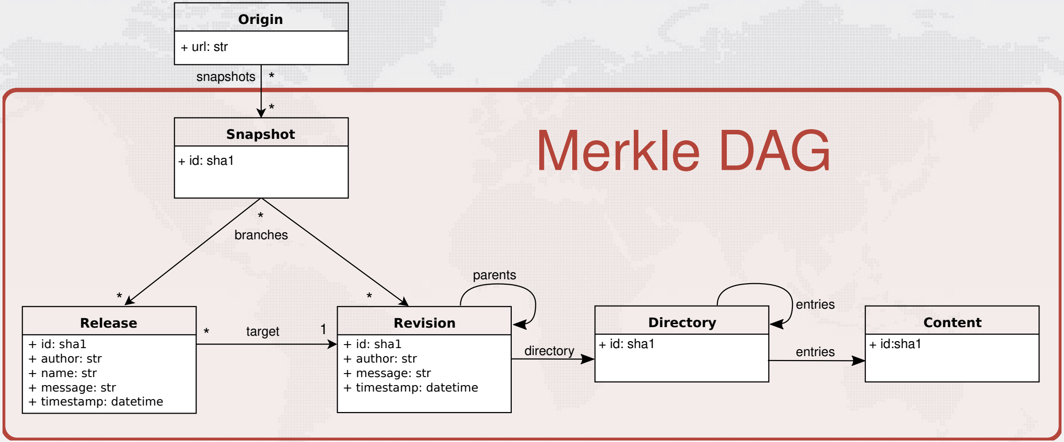
Adjacency lists

Node	Outd.	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
...	...	...

Copy lists

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

Data model



Origin
+ url: str
snapshots
*
*
Snapshot
+ id: sha1
*
branches
Merkle DAG
*
*
parents
entries
Release
+ id: sha1
+ author: str
+ name: str
+ message: str
+ timestamp: datetime
*
target
1
Revision
+ id: sha1
+ author: str
+ message: str
+ timestamp: datetime
directory
Directory
+ id: sha1
entries
Content
+ id:sha1

Corpus — as a graph

Nodes

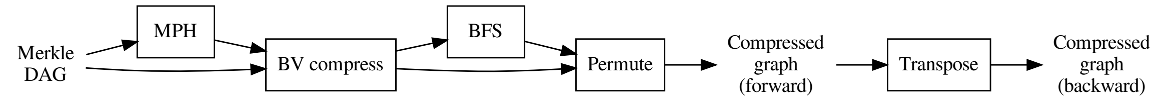
Node type	N. of nodes
origins	88 M
snapshots	57 M
releases	9.9 M
revisions	1.1 B
directories	4.9 B
contents	5.5 B
Total nodes	12 B

Edges

Edge type	N. of edges
origin ⟶ snapshot	195 M
snapshot ⟶ revision	616 M
snapshot ⟶ release	215 M
release ⟶ revision	9.9 M
revision ⟶ revision	1.2 B
revision ⟶ directory	1.1 B
directory ⟶ directory	48 B
directory ⟶ revisiony	482 M
directory ⟶ content	112 B
Total edges	165 B

Archive snapshot 2018-09-25, from the Software Heritage graph dataset.
Growth rate: exponential, doubling every 22-30 months.

Compression pipeline



- MPH: minimal perfect hash, mapping Merkle IDs to 0..N-1 integers
- BV compress: Boldi-Vigna compression (based on MPH order)
- BFS: breadth-first visit to renumber
- Permute: update BV compression according to BFS order

(Re)establishing locality

- key for good compression is a node ordering that ensures locality and similarity
- which is very much *not* the case with Merkle IDs…
- …but is the case *again* after BFS

Compression time

We ran the compression pipeline on the input corpus using the WebGraph framework

Paolo Boldi and Sebastiano Vigna.
The WebGraph framework I: Compression techniques
WWW 2004: 13th Intl. World Wide Web Conference. ACM

Step	Wall time (hours)
MPH	2
BV Compress	84
BFS	19
Permute	18
Transpose	15
Total	138 (6 days)

- server equipped with 24 CPUs and 750 GB of RAM
- RAM mostly used as I/O cache for the BFS step
- *minimum* memory requirements are close to the RAM needed to load the final compressed graph in memory

Compression efficiency

Forward graph

total size	91 GiB
bits per edge	4.91

Backward graph

total size	83 GiB
bits per edge	4.49

Operation cost

The structure of a full bidirectional archive graph fits in less than 200 GiB of RAM, for a hardware cost of ~300 USD.

A domain-agnostic benchmark — full corpus traversal

Benchmark — Full BFS visit

Forward graph	
wall time	1h48m
throughput	1.81 M nodes/s
	(553 ns/node)

Backward graph	
wall time	3h17m
throughput	988 M nodes/s
	(1.01 µs/node)

Benchmark — Edge lookup

random sample: 1 B nodes (8.3% of entire graph)

Forward graph	
visited edges	13.6 B
throughput	12.0 M edges/s
	(83 ns/edge)

Backward graph	
visited edges	13.6 B
throughput	9.45 M edges/s
	(106 ns/edge)

Note how edge lookup time is close to DRAM random access time (50-60 ns).

Domain-specific benchmarks — source code artifact multiplication

Simple clone detection style experiments realized exploiting the compressed corpus:

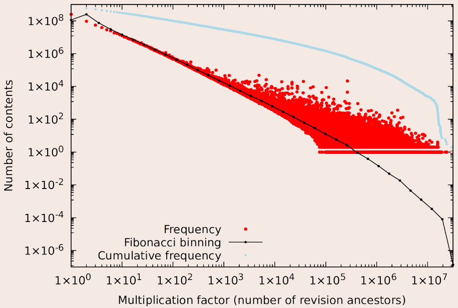
1. file→commit multiplication: how much identical source code files re-occur in different comments
2. commit→origin multiplication: how much identical commits re-occur in different repositories

Implementation

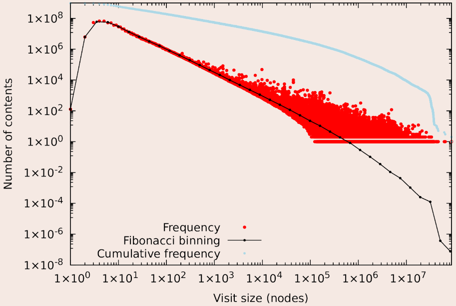
- for each node—content for (1), commit for (2)—visit the backward graph and count all reachable nodes of the desired type—commit for (1), origin for (2)
- naive approach, O(|V|x|E|) complexity

File⟶commit multiplication — results

Multiplication factor



Visit size



- random sample of 953 M contents (17% of the full corpus)
- processing time: ~2.5 days (single machine with 20 x 2.4 GHz cores)
  - *in spite of* the naive O(|V|x|E|) approach, generally considered intractable at this scale

Limitations

Incrementality

- compression is inherently not incremental
- not an issue for most research use cases, because we analyze immutable data dumps
- common workaround (e.g., for the Web and social networks) is to keep an uncompressed in-memory overlay for graph updates, and periodically recompress

Limitations

Incrementality

- compression is inherently not incremental
- not an issue for most research use cases, because we analyze immutable data dumps
- common workaround (e.g., for the Web and social networks) is to keep an uncompressed in-memory overlay for graph updates, and periodically recompress

In-memory v. on-disk

- the compressed in-memory graph structure has no attributes
- usual data design is to exploit the 0..N-1 integer ranges to memory map *node* attributes to secondary storage
    - we have done this with a node type map; it weights 4 GB (3 bit per node)
- works well for queries that do graph traversal first and "join" node attributes last; ping-pong between the two is expensive
- *edge* attributes are more problematic

Wrapping up

- Graph compression is a viable technique to analyze the history of all public source code, as captured by modern version control systems (VCS), on a budget.
- It is a novel tool for VCS analyses that might allow to expand the scope of our experiments, reducing selection biases and improving external validity.
- More work is needed to provide compression incrementality and allow to efficiently query VCS properties during traversal.

See full paper for more details

Paolo Boldi, Antoine Pietri, Sebastiano Vigna, Stefano Zacchiroli
Ultra-Large-Scale Repository Analysis via Graph Compression
SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering. IEEE

preprint: `http://bit.ly/swh-graph-saner20`

Contacts

Stefano Zacchiroli / zack@irif.fr / @zacchiro / talk to me at SANER 2020!