Compressing the graph of package and repository dependencies

Val Lorentz (@vlorentz)

Software Heritage

2026-10-13 - SWHSec Plenary



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Outline

1. Data source
2. Goals
3. Data structures
4. Conclusion

ecosyste.ms

- https://ecosyste.ms/
- By Andrew Nesbitt, author of libraries.io
- 15M source or binary packages from 110 ecosystems (PyPI, NPM, Docker, Debian, F-Droid, …)
- 350M repositories
- Mines dependencies, security advisories, SBOMs, Dependabot PRs, licenses, funding, paper citations, …

Data dump

- Latest public dump is from 2023
- 200GB gzipped postgresql dump (excludes indices)
- 153M nodes: 8M packages, 145M repos (half of today's)
- 11.4B arcs: 0.9B package deps, 10.5B repo deps
- Easy! That's 100 times smaller than swh-graph

Data model

- Package -> versions (average: 10)
- Version -> dependencies (average: 11)
- Repository -> dependencies (average: 72)
- Dependency = one target package + requirements + …
  - Requirements are arbitrary strings

Outline

1. Data source
2. Goals
3. Data structures
4. Conclusion

Goals

- Recursively list packages that are popular dependencies
- Recursively list packages/repos with a vulnerable dependency
- On commodity hardware, so we need:
  - O(1) access to each package's dependencies and dependants
  - most of it in <16GB of RAM (<32GB today)

Graph representation

- One node for each package or repository
- One arc between two nodes that ever had a dependency relation
- Specify versions and constraints on arc labels

Outline

1. Data source
2. Goals
3. Data structures
4. Conclusion

Nodes

- Minimal Perfect Hash functions:
  - Maps package name to [0, 8M] range using 3MB
  - Maps repo URL to [8M, 153M] range using 40MB
- Permutation + prefix erasure:
  - Store all names/URLs in alphabetical order
  - Deduplicate prefixes
  - Maps [0M, 153M] range to name/URL in 3GB

Arcs

- First, renumber all nodes
  - Cluster close nodes together using Layered Label Propagation (takes 1.5h!)
  - We get a 500MB permutation
- Represent adjacency lists in WebGraph's BVGraph
  - Compresses similar adjacency lists
  - 1.5GB for forward, 1.1GB for backward graph
  - 6.8B arcs (ie. avg 1.8 or 1.3 bits per arc)
  - 2× 150MB for Elias-Fano indices

Elias-Fano sequences

00000 , 00001 , 00011 , 01010 , 01011 , 11111

Elias-Fano sequences

```
00000 , 00001 , 00011 , 01010 , 01011 , 11111
000 00, 000 01, 000 11, 010 10, 010 11, 111 11
```

Elias-Fano sequences

00000 , 00001 , 00011 , 01010 , 01011 , 11111
000 00, 000 01, 000 11, 010 10, 010 11, 111 11

Low bits: 00 01 11 10 11

Elias-Fano sequences

00000 , 00001 , 00011 , 01010 , 01011 , 11111
000 00, 000 01, 000 11, 010 10, 010 11, 111 11

Low bits: 00 01 11 10 11

High bits: 1 1 1 0 0 1 1 0 0 0 0 0 1

Arc labels: problem

- A -> B arc = "there is at least one version of A that depend on at least one range of versions of B"
- We need to label arcs with the exact versions and ranges
- There are 11.4B dependencies, so 3.5 bytes per relation -> 40GB

Arc labels: secondary structures

- Package version "numbers":
  - 84M versions, 7M unique
  - sorted by (package, version) + prefix omission
  - 510MB for version "numbers" (150MB on ZFS + zstd), 200MB for indices
  - I can probably make it smaller by deduplicating
- Version constraints:
  - Only 3.2M unique strings!
  - Sorted + prefix ommission: 46MB
  - Monotone MPH: 7MB

Arc labels: current format 1/2

all numbers use the Gamma instantaneous code:
1, 010, 011, 00100, 00101, 00110, 00111, 0001000, …
on each A -> B arc where A is a repository:

- number of requirement strings
- sorted requirement string ids, stored as deltas

Arc labels: current format 2/2

on each A -> B arc where A is a package:

- gamma-coded 0
- number of unique requirement strings of A
- For each unique requirement string of A, sorted:
  - requirement string id, stored as deltas
  - number of versions with this string
  - relative version id, stored as deltas

Arc labels: not good enough

- Currently averages to 3.5 bytes per relation -> 2× 40GB
- ZFS (with zstd) further compresses them to 19GB (forward) and 7.4GB (backward)
- ideas:
  - look at what exactly takes a lot of space
  - other instantaneous codes, bit fields
  - cluster requirement string ids?
  - deduplicate adjacent labels
  - ?

Outline

1. Data source
2. Goals
3. Data structures
4. Conclusion

Conclusion

- Matrix: `#swh-devel:matrix.org` / `@vlorentz:trix.re`
- Email: `vlorentz@softwareheritage.org`
- Code at `https://gitlab.softwareheritage.org/vlorentz/ecosystems-depgraph`