Solving the bootstrap problem for Debian based operating systems

Johannes 'josch' Schauer

Jacobs University Bremen

DebConf 2013, Vaumarcus

Overview

- botch = Bootstrap/Build Ordering ToolCHain
- Started as Debian Google Summer of Code project 2012
- Continued as my master thesis at Jacobs University Bremen
- Mentors:
  - **Wookey** practical side
  - **Pietro Abate** theoretical/academic side

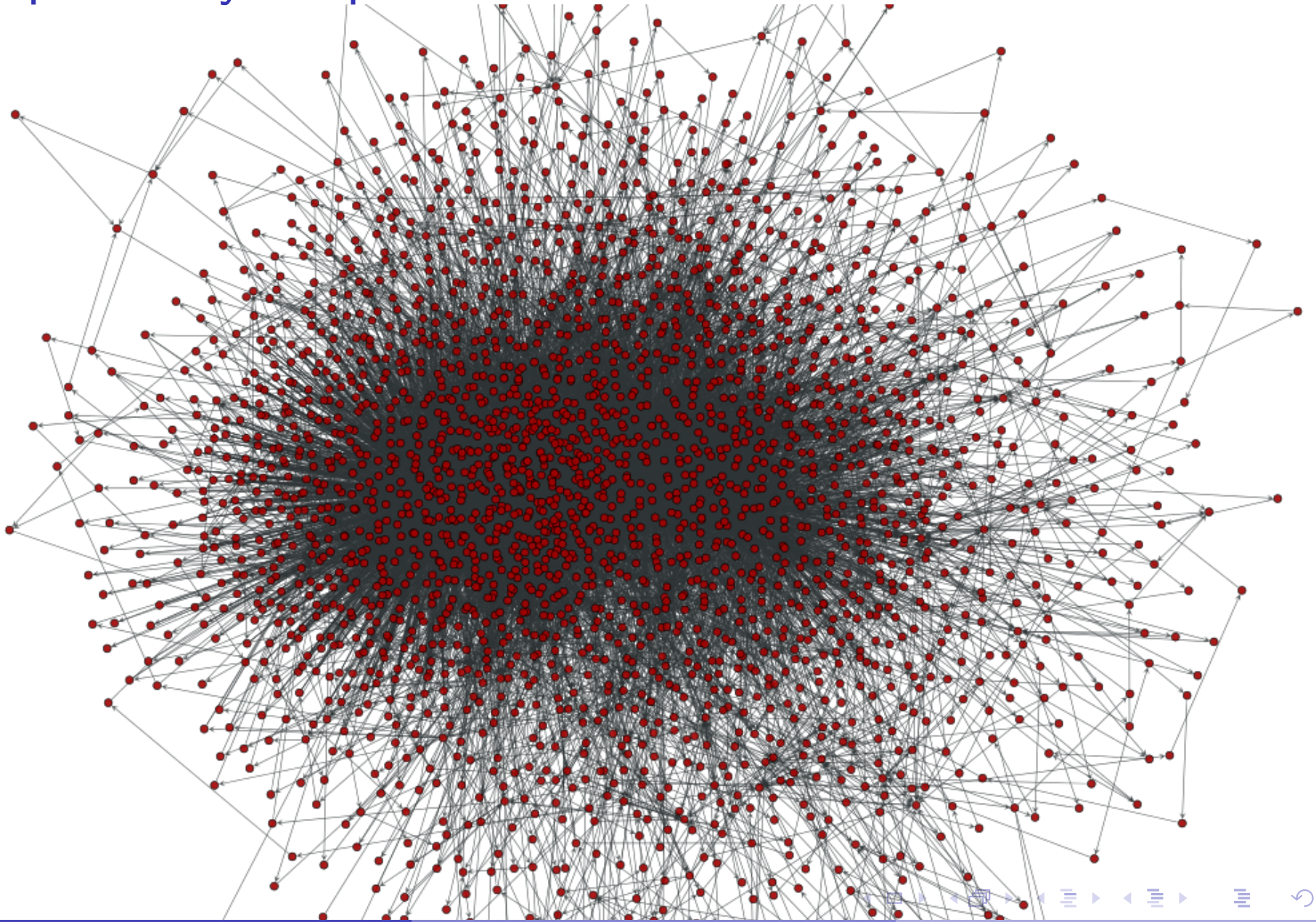
Common Case

- Source packages are always natively compiled
- Source packages are compiled with the full archive of binary packages available

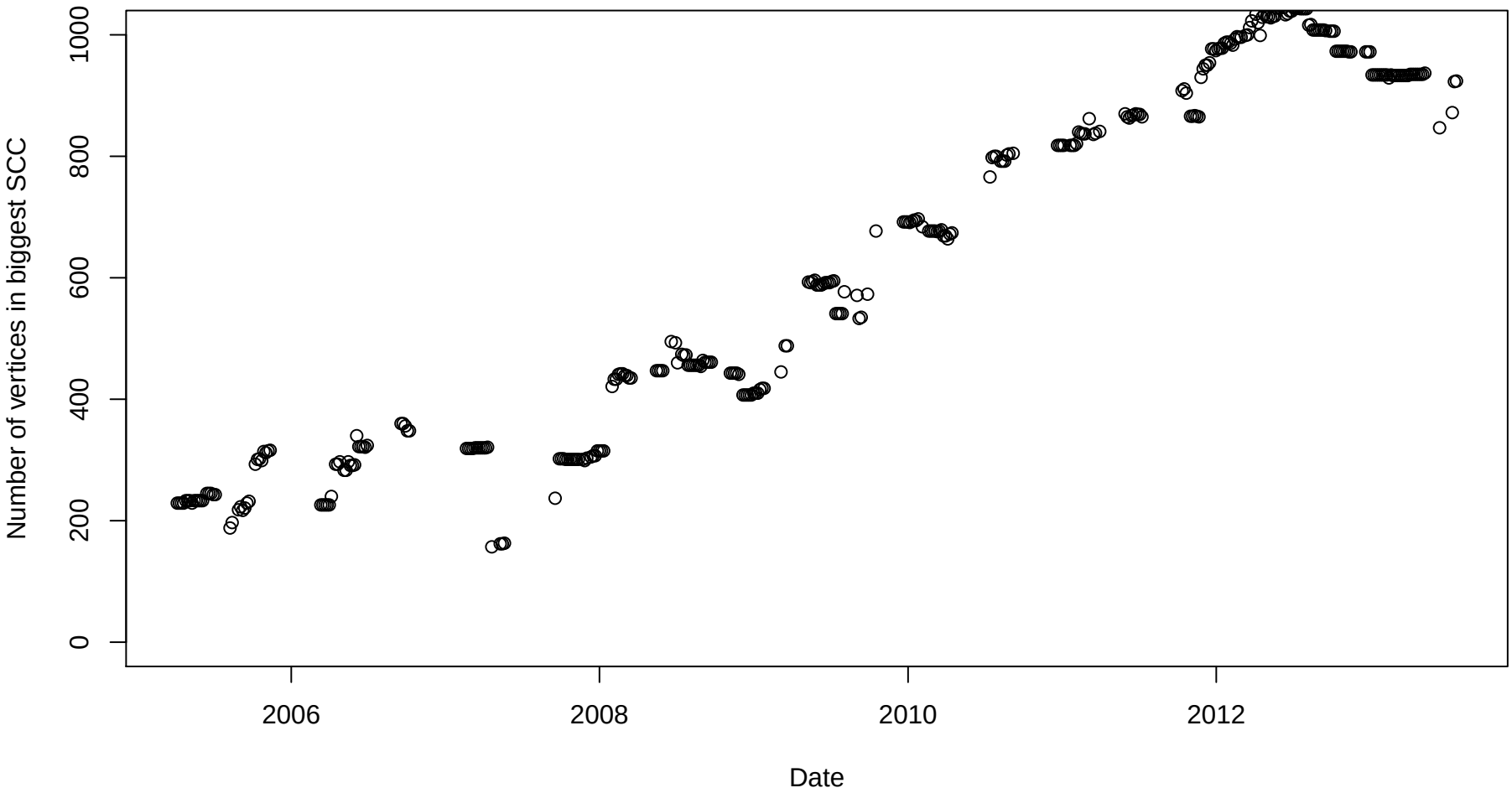
During Bootstrapping

- Some source packages must be cross compiled
- Only a few binary packages are available $\rightarrow$ dependency cycles

Dependency Graph in Debian Sid



Development of problem size



Current Bootstrapping Practice

- Using Gentoo or OpenEmbedded to avoid cross compilation of the base system
- Manual dependency cycle analysis
- Manual hacking of source packages to build with fewer build dependencies
- Takes several months up to a year to complete

What if bootstrapping was easier?

- Easier porting for upcoming architectures
- More custom ports, optimized for a specific CPU
- Remove the need of Gentoo or OpenEmbedded (make Debian more universal)

Wait, there is more!

- Update lagging architectures
- Build for targets that can't build themselves (once cross building gained better support)
- QA tool which allows to check the archive for bootstrappability
- Order rebuilds for library transitions (Haskell, OCaml)

The essence of this talk

- The core algorithms for graph analysis exist and they are fast and seem to be correct
- We need decisions about new dependency syntaxes, multiarch and cross building to do the practical plumbing

The tools

- Written in OCaml, Python, Shell
- LGPL3+
- Using dose3 as helper library (parser, solver, ...)
- UNIX Philosophy: Multiple application, each executing one algorithm connected by pipes
- Exchange format is a plain text Debian package description format
- Graphs are output in GraphML to be consumed and analyzed by 3rd party tools
- Git: `https://gitorious.org/debian-bootstrap/botch`

More specifically we can now...

- ... create & analyze a dependency graph
- ... find source packages to modify
- ... create a build order

It's only theory

- Tools only work on package meta data
- No source packages are compiled, no binary packages installed

What is needed to test in practice

- Reduced build dependencies (build profiles)
- Cross compilation support in base packages

Bootstrap Workflow

1. Select binary packages for minimal build system and cross compile them for the new platform
2. Create Build Graph and extract strongly connected components
3. If the amount of existing build profiles is not enough to make build graph acyclic:
   - use heuristics to find source packages to add build profiles to
   - modify the respective source packages
   - go back to 2
4. Feedback Vertex Set algorithm selects source packages to profile build and a build order is deduced from the now acyclic graph

Breaking dependency cycles

- Remove build dependencies (build profiles)
- Move dependencies from `Build-Depends` to `Build-Depends-Indep`
- Choose different installation sets for not-strong dependencies
- Make binary packages available through cross compilation
- Use existing `Multi-Arch:foreign` packages
- Split a source package

Heuristics

- Goal: finding source packages to modify
- Heuristics are needed because the investigation can only be done by a human developer
- Different heuristic kinds based on dependency graph syntax (mostly ignoring semantics)
  - Simple (degree ratios, degree count)
  - Component based (strong brides and articulation points)
  - Cycle based
  - Feedback Arc Set

HTML Display of Heuristics

- `http://mister-muffin.de/bootstrap/stats/`

SCC #1

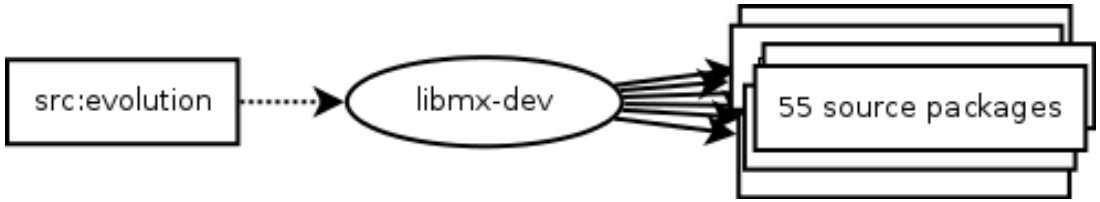
first prev 1/45 next last 10 per page (446 rows in total)

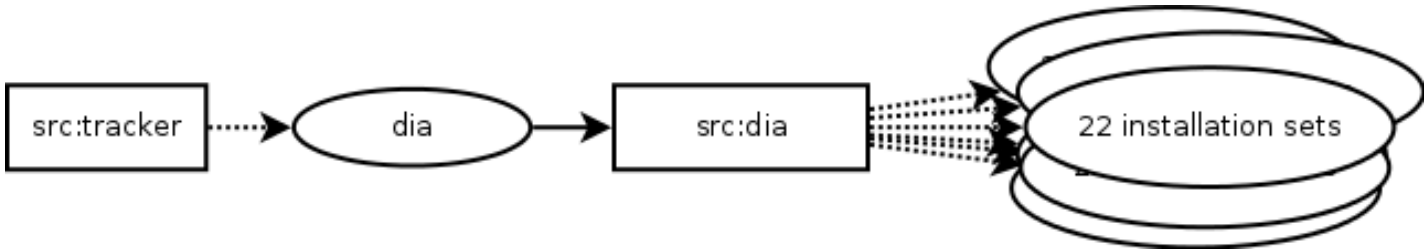
Amount	Edge
595	src:libgcrypt11, texlive-generic-recommended
594	src:libgcrypt11, texlive-latex-base
279	src:libtasn1-3, texlive-latex-base
233	src:avahi, python-gtk2
221	src:avahi, libgtk-3-dev
211	src:gnutls26, gtk-doc-tools
196	src:libtasn1-3, gtk-doc-tools

Figure : Table of edges with most cycles through them

Simple Heuristics

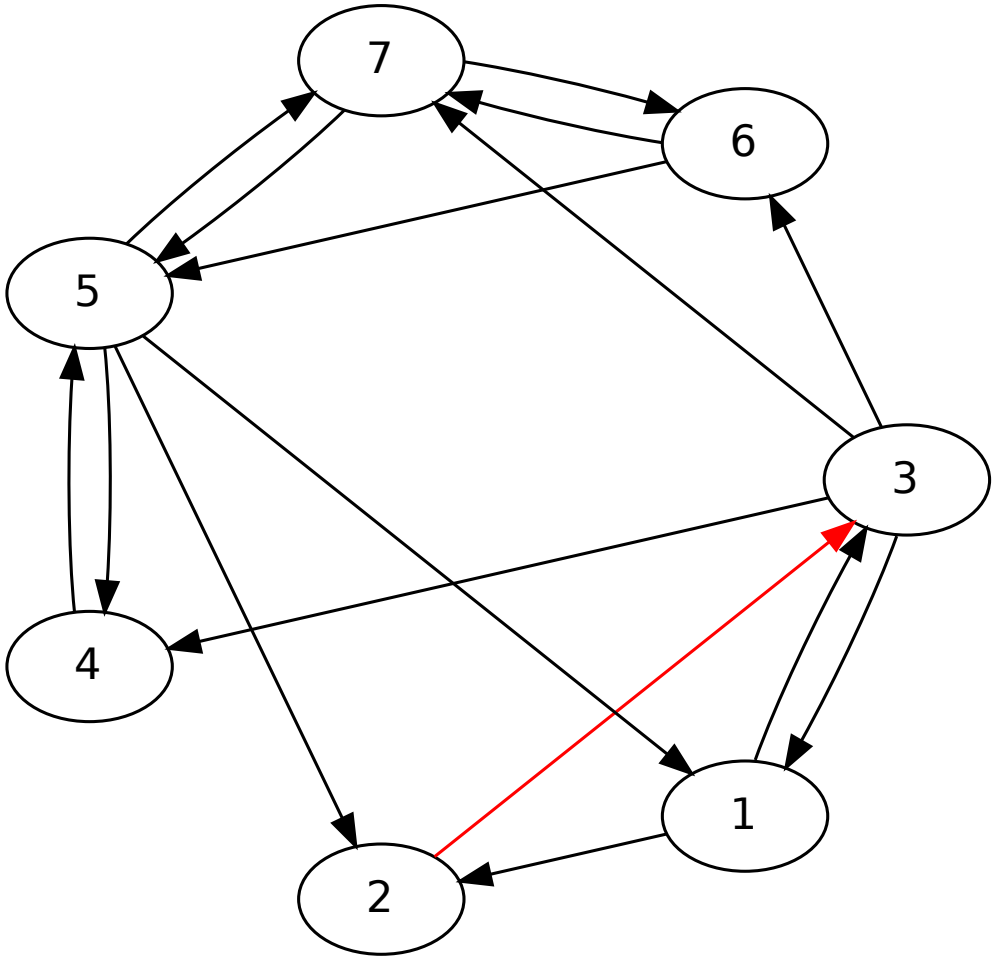
- Ratio based heuristics



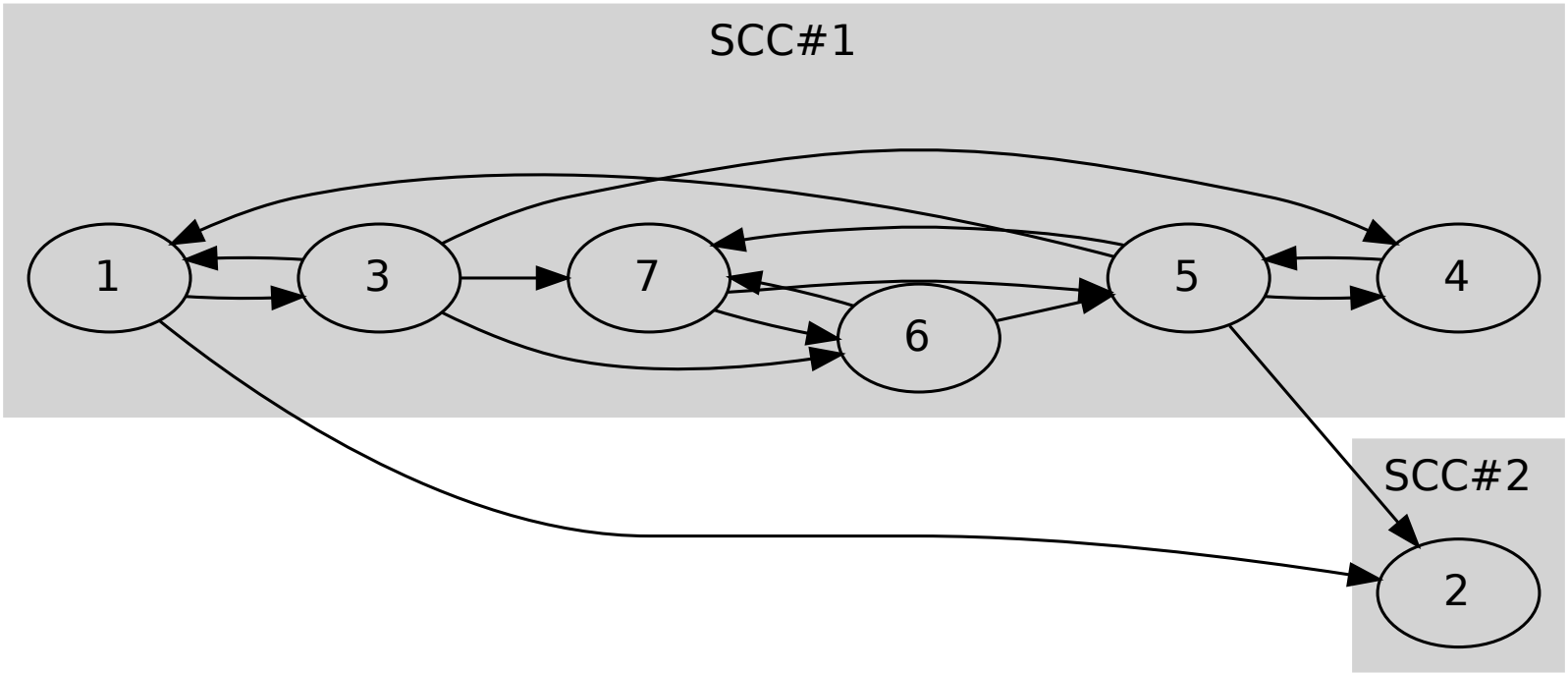


- Amount of missing dependencies
- Amount of missing “weak” dependencies (build dependencies commonly used for documentation generation and unit tests)

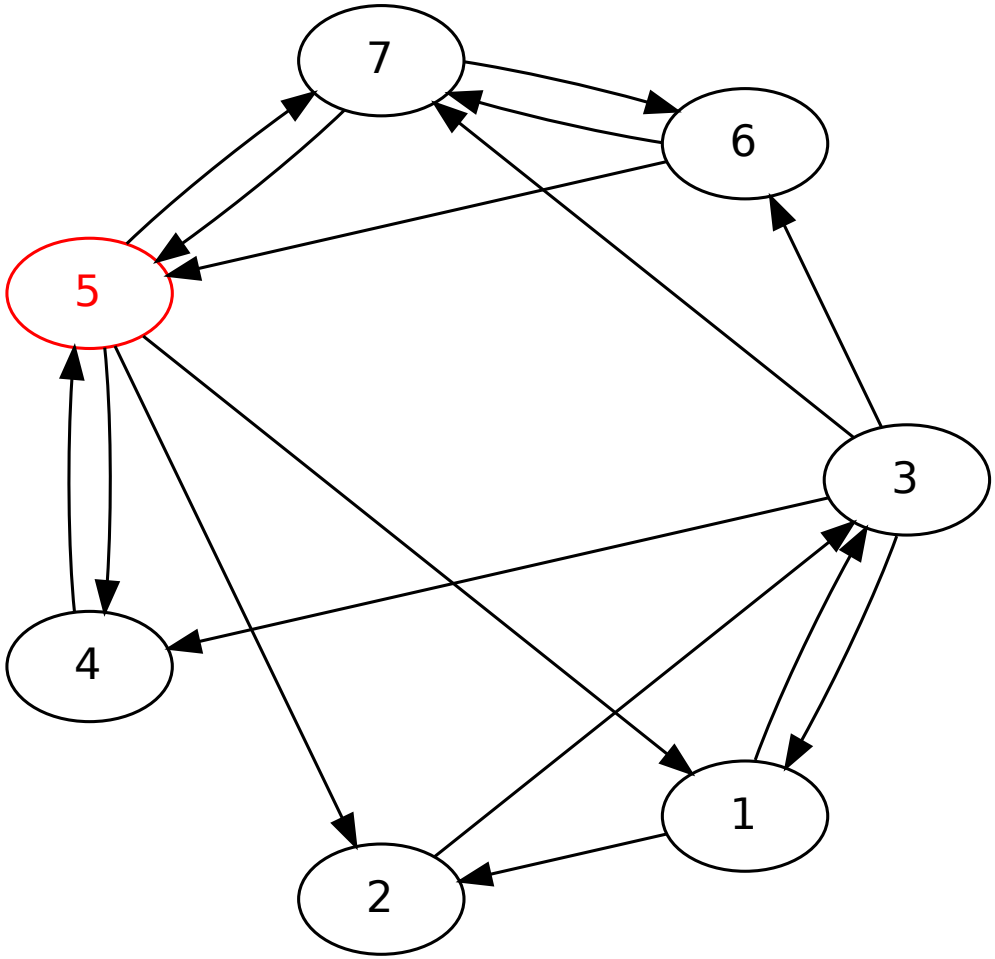
Strong bridges



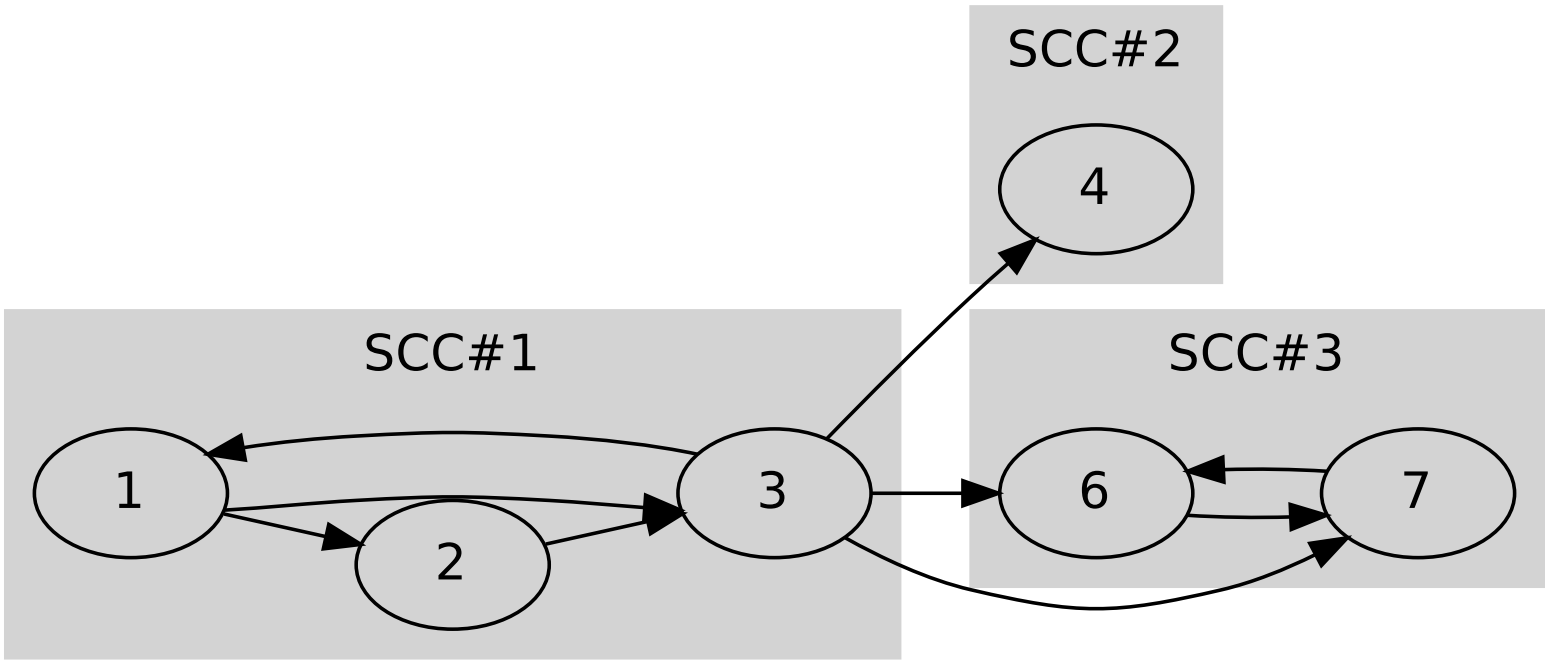
Strong bridges



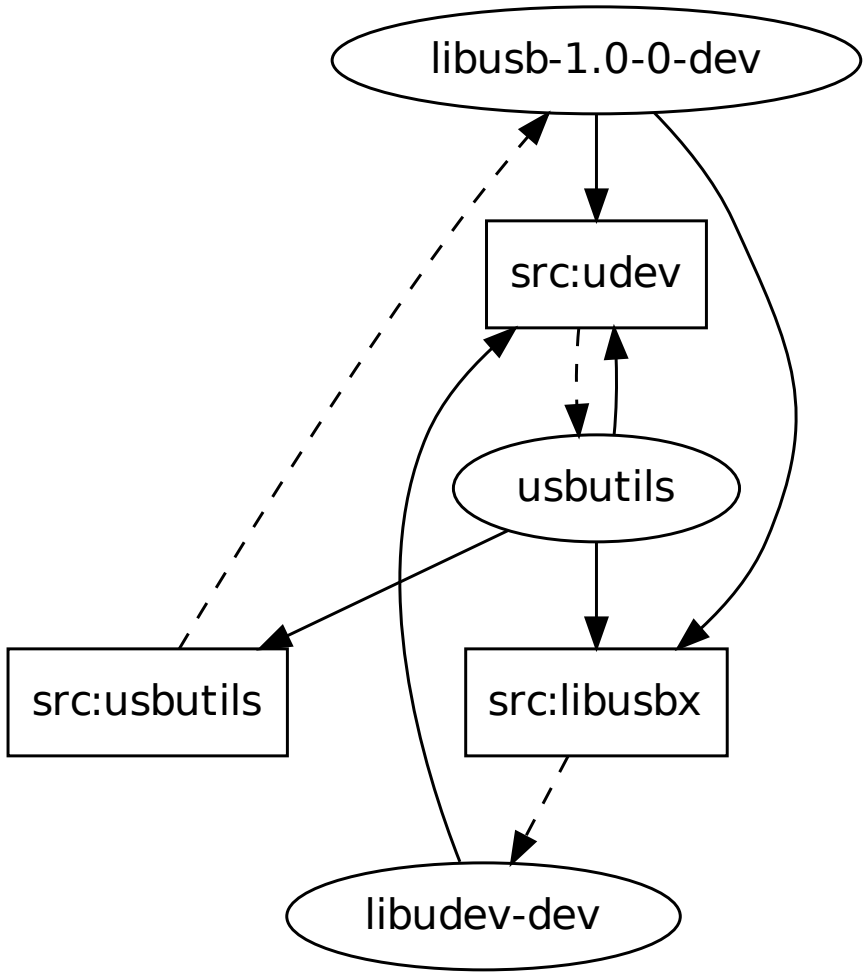
Strong articulation points



Strong articulation points



Small cycles



HTML Display of Self-Cycles

- `http://mister-muffin.de/bootstrap/selfcycles.html`

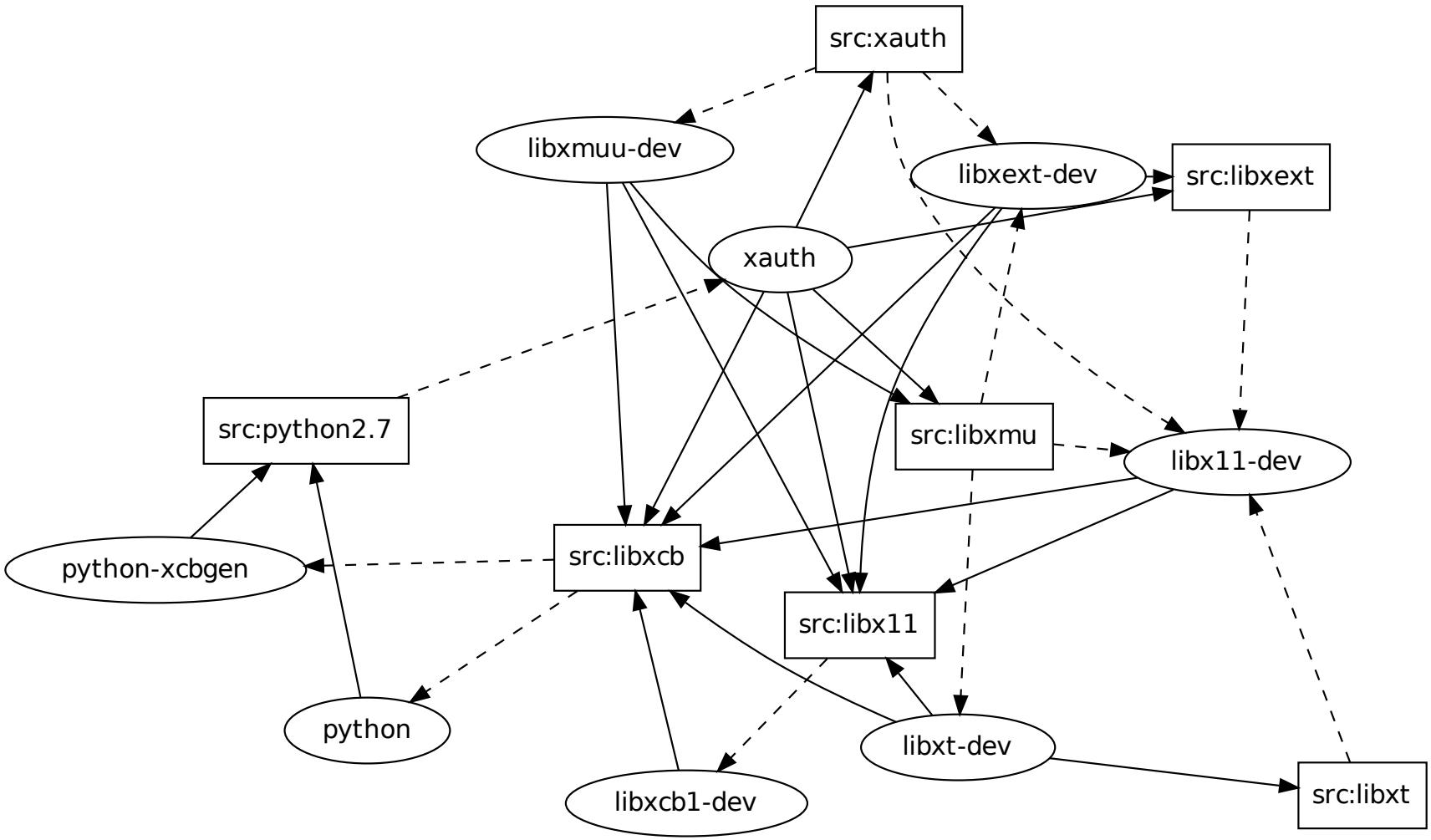
Type 2

Source packages which indirectly strongly depend on binary packages they build.

source package	strongly depends on	because of the source package build depending on	amd64	armel	i386	ia64	kfreebsd-
cyrus-sasl2	libsasl2-2	heimdal-multidev, libldap2-dev, libpq-dev					
hscolour	hscolour	haskell-devscripts					
java-access-bridge	libaccess-bridge-java-jni	openjdk-6-jdk					
libx11	libx11-6	groff					
libxaw	libxaw7	groff					
newlib	newlib-spu	gcc-spu					
openldap	libldap-2.4-2	heimdal-dev					
opensp	libosp5	openjade, openjade1.3					
pkg-config	pkg-config	libglib2.0-dev					
python-numpy	python-numpy	python-matplotlib					
	libpython2.6						

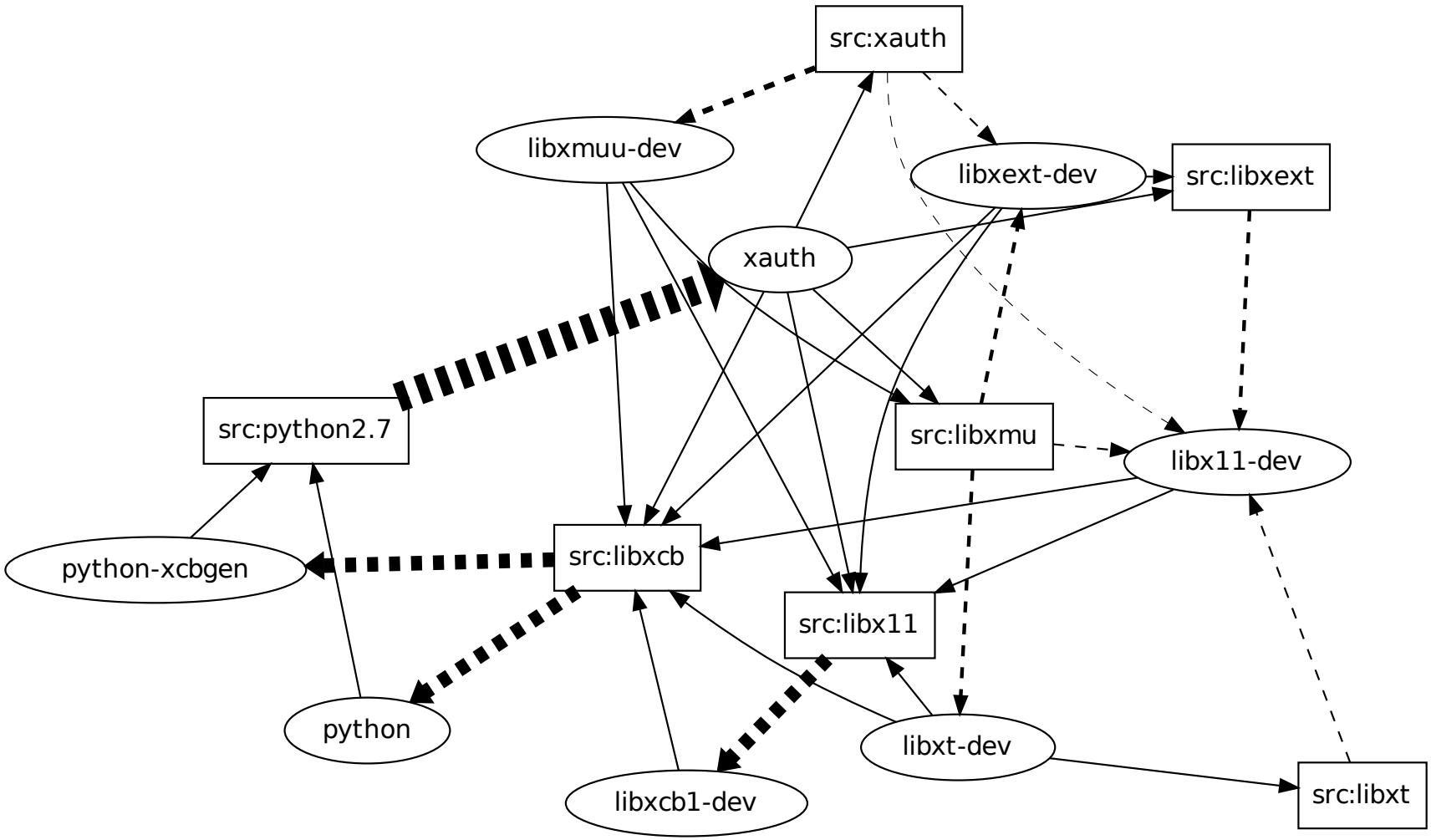
Edges with most cycles through them

- SCC with 15 vertices, 31 edges



Edges with most cycles through them

- SCC with 15 vertices, 31 edges



Calculating a Feedback Arc Set

- biggest SCC: 1080 vertices, 11726 edges
- assuming everything can be broken
  - breakable by modifying 51 source packages
- more realistic using Gentoo and lists by Thorsten Glaser, Patrick McDermott, Daniel Schepler, Wookey
  - breakable by modifying 57 source packages

Benchmark Results

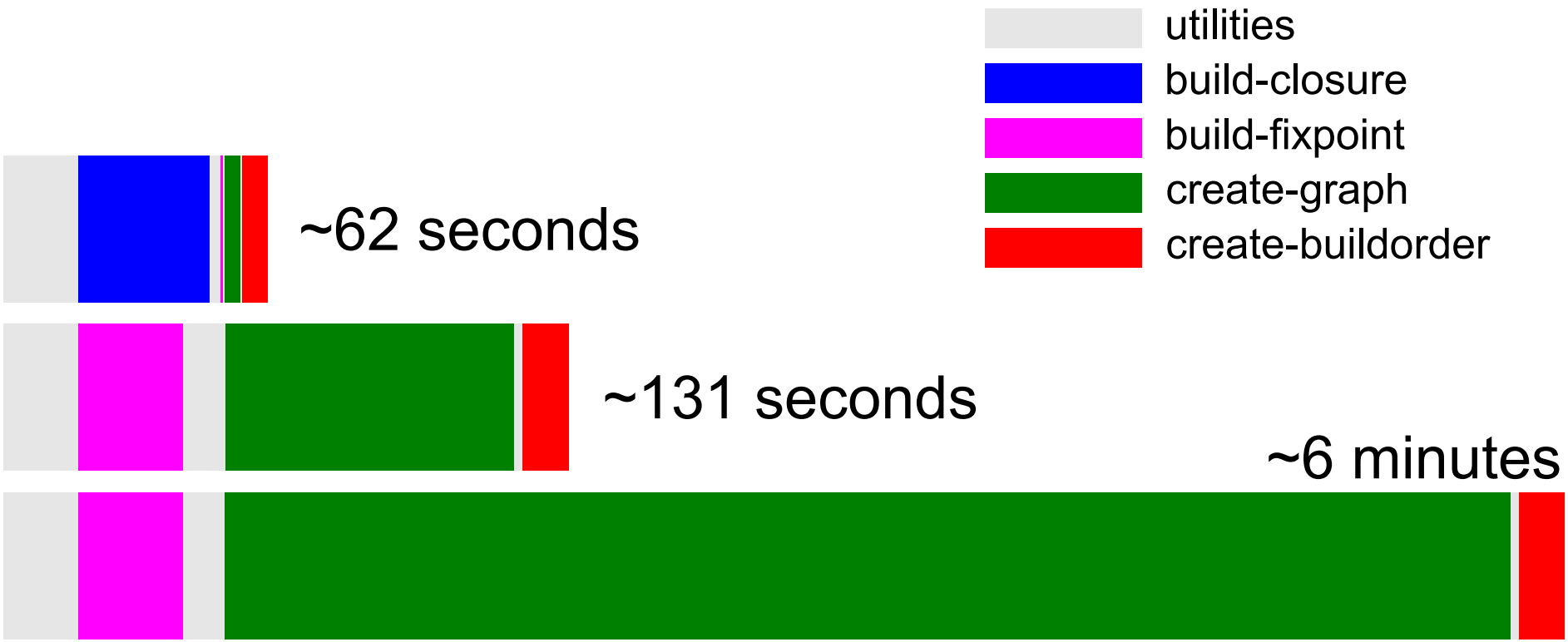


Figure : Execution time using a self-contained repository (top), normal execution (middle) and with computing strong dependencies (bottom)

Resources

- Blog: `http://blog.mister-muffin.de`
- ML: `debian-bootstrap [at] lists.mister-muffin.de`
- IRC: `#debian-bootstrap [at] irc.oftc.net`
- Git1: `https://gitorious.org/debian-bootstrap/botch`
- Git2: `https://gitorious.org/debian-bootstrap/gen2deb`
- Git3: `https://github.com/josch/cycle_test`
- Dose3: `https://gforge.inria.fr/projects/dose/`
- Wiki1: `http://wiki.debian.org/DebianBootstrap`
- Wiki2: `http://wiki.debian.org/DebianBootstrap/TODO`
- Wiki3: `https://gitorious.org/debian-bootstrap/pages/Home`
- Thesis: `http://mister-muffin.de/bootstrap/thesis.pdf`
- Profiles: `https://l.d.o/debian-devel/2013/01/msg00329.html`

Conclusion

- We could have:
  - Easier porting
  - More custom ports
  - Remove the need of Gentoo or OpenEmbedded
  - Update lagging architectures
  - Build for targets that can't build themselves
  - QA tool checking for bootstrappability
- But we are missing:
  - Decision on the build profile format
  - Several fixes to support cross compilation

Questions

Questions?