Bootstrapping Software Distributions

Pietro Abate [1] Johannes Schauer [2]

[1] Univ Paris Diderot, PPS,
UMR 7126, Paris, France

[2] Jacobs University Bremen,
College Ring 3, MB670,
28759 Bremen

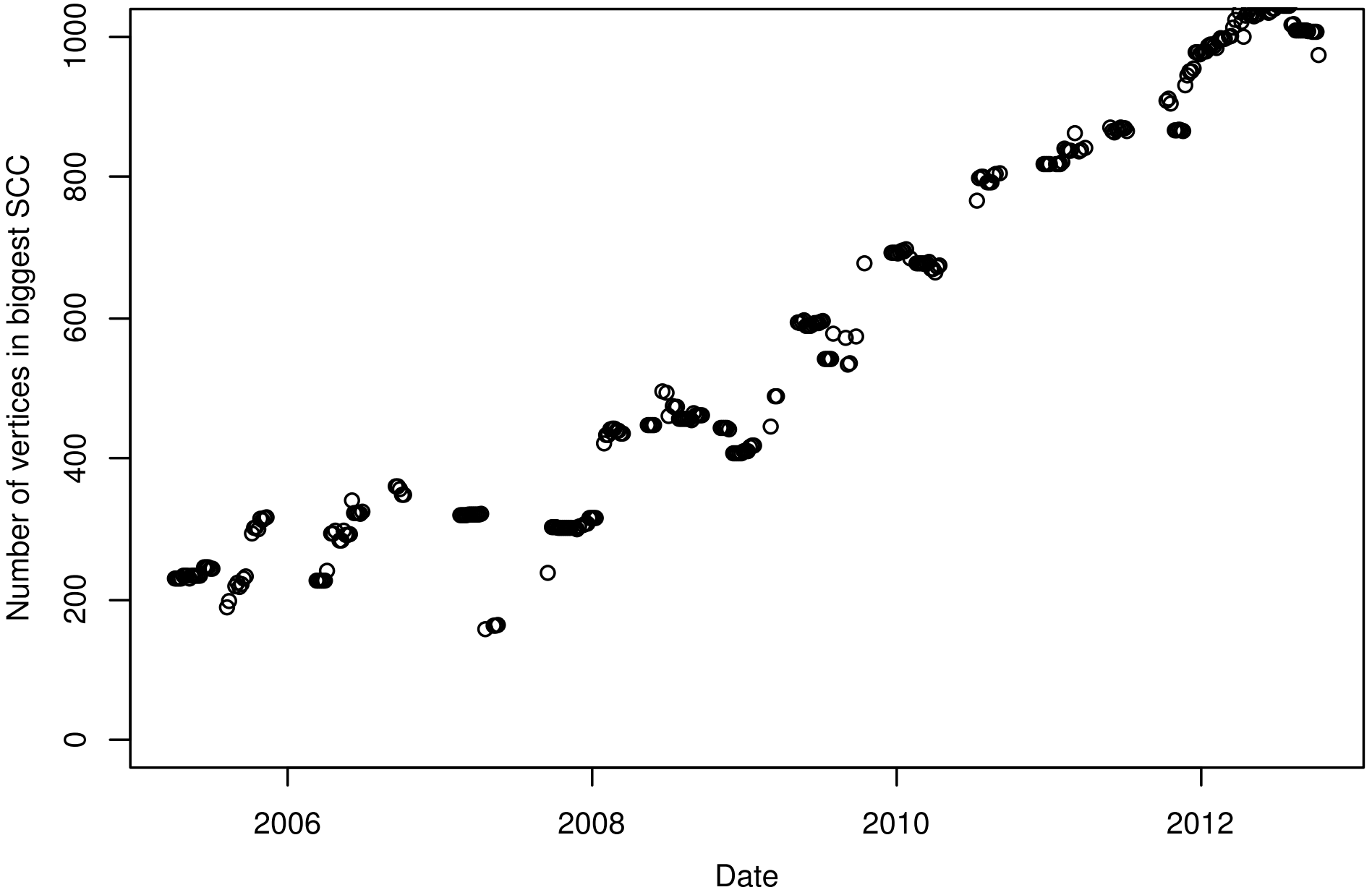
CBSE 2013

The Bootstrap Problem

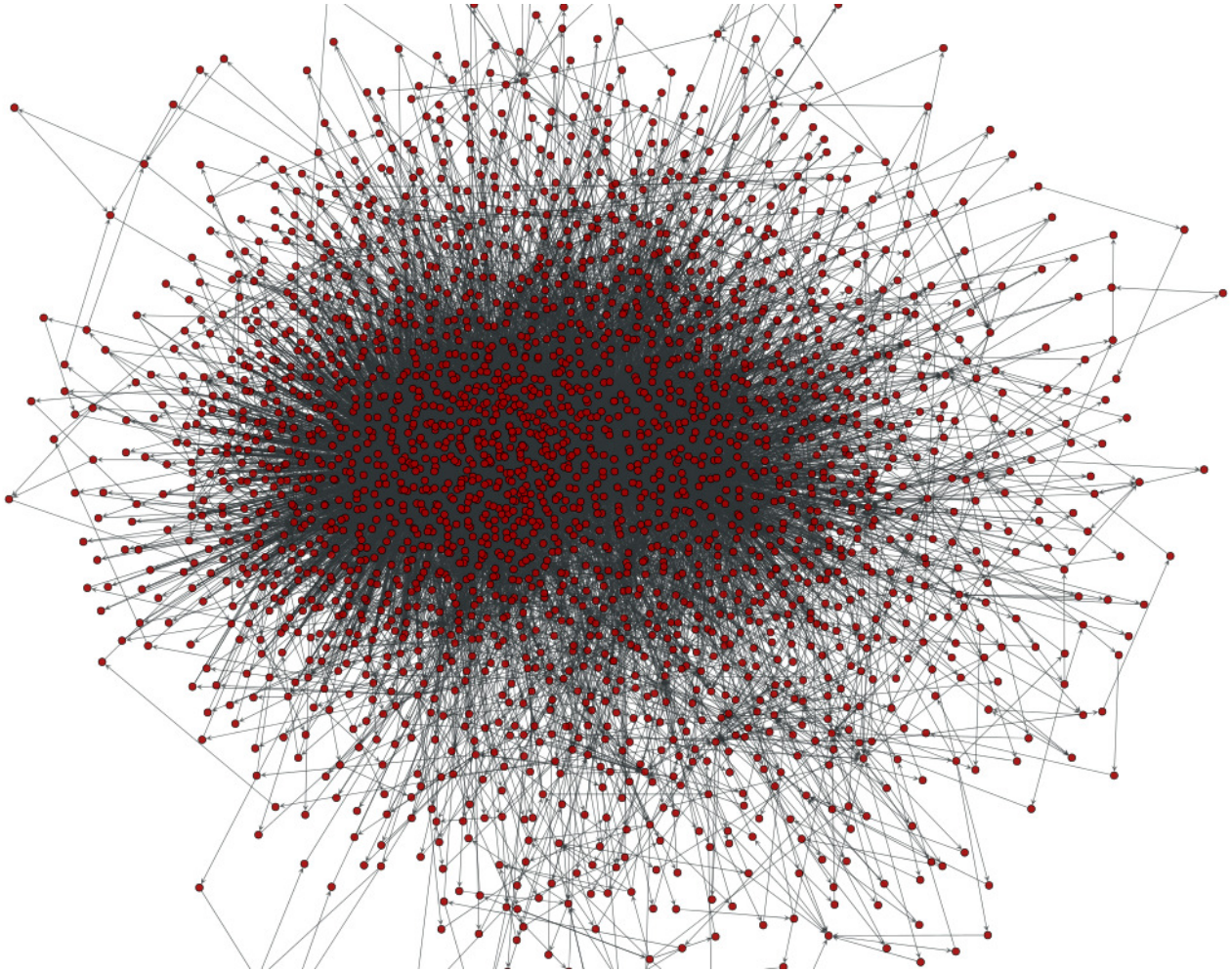
- **Software distributions** are composed of thousands of components, interconnected by a web of dependency and conflict relations among **binary packages**.
- FOSS distributions are available for a number of different **hardware architectures** with new architectures added every year.
- Interdependencies between **source packages** give rise to millions of circular dependencies in form of **Strongly Connected Components**
- **Porting** a distribution to a new architecture means to recompile all source packages for that new architecture.
- The goal of this work is to provide tools and heuristics to **remove** all build dependency cycles with a **minimal** amount of changes and to deduce a **build order**

Status Quo

- Porting a distribution to a new hardware platform was not an automatic process.
- Build cyclic dependencies had to be found manually without a clear method (mostly handicraft work).
- Removing cyclic build dependencies likely entailed more changes than necessary.
- Poor documentation: the process had to be repeated for every port.
- Process took up to a year.


Number of vertices in biggest SCC
0
200
400
600
800
1000
2006
2008
2010
2012
Date

Biggest SCC January 2013



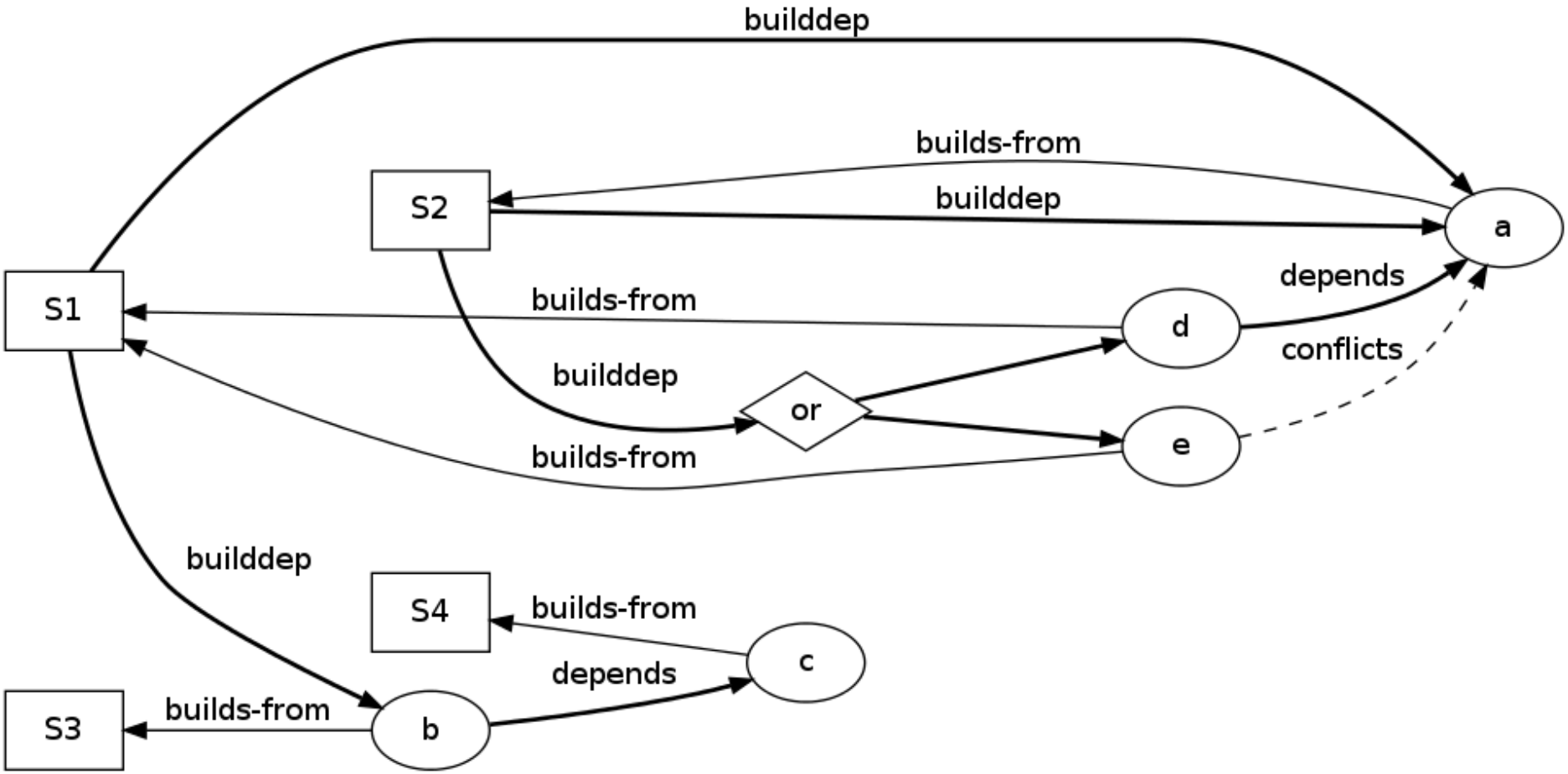
Our Contribution

- Provide a formal framework to reason about build dependencies.
- Develop algorithms to untangle the dependency graph and heuristic to remove build cycles.
- Calculate a build order with minimal changes to packages
- Test our tools on the Debian software distribution and create a liaison with the Debian community.

Binary and Source Packages

- **Source Packages**
  - Project source code and metadata
  - Build-Depends on binary packages
  - Build-Conflicts with binary packages
  - Builds binary packages
- **Binary Packages**
  - Executables, data files, metadata
  - Depends on other binary packages
  - Conflicts with other binary packages
  - Builds from source packages

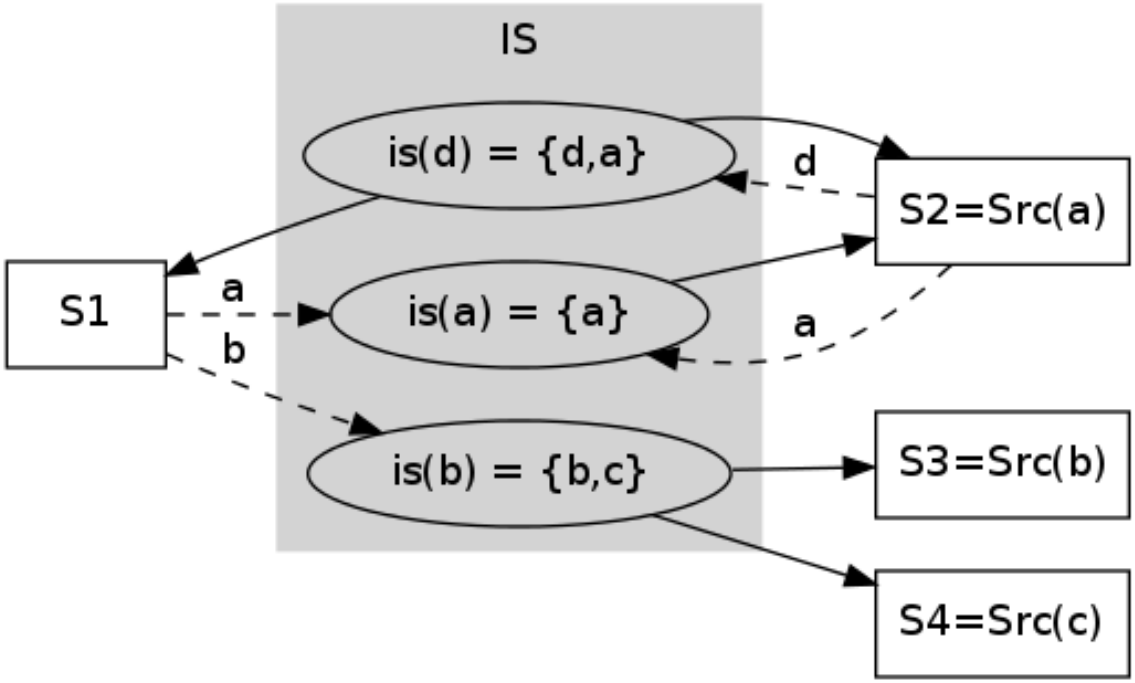
Dependency Graph

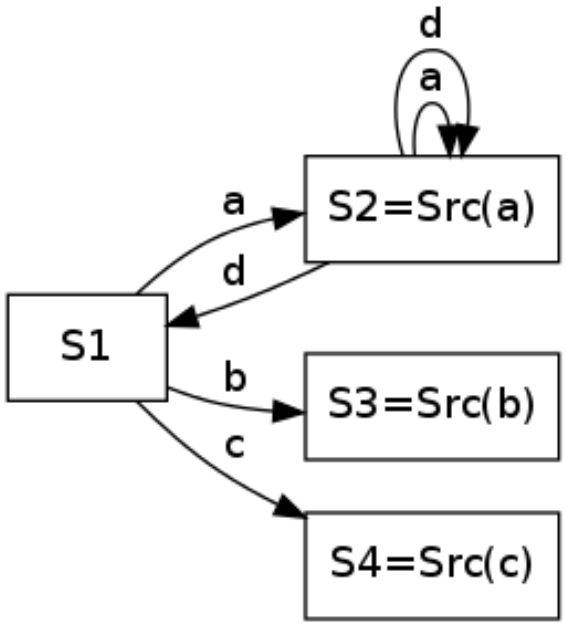


Installation Sets

- A set of packages for which for each package
  - All dependencies are satisfied
  - No conflicting package is part of the set
- Due to disjunctive dependencies binary packages can have multiple installation sets
- We do not treat binary packages individually but always together with an installation set

Build Graph and Source Graph





Repositories

A repository is a tuple (P, Dep, Con, Bin) where

- P is a set of binary or source packages
- Dep is the dependency function (builddep or depends)
- Con is the conflict relation toward binary packages
- Bin is the binary function mapping source packages to the binary packages they build

Build Profiles

- A **Build profiles** is a different configuration for compiling a source package
- It is used to remove dependency cycles by building source packages with a reduced feature set and therefore less build dependencies
- Examples:
  - no documentation
  - no language bindings
  - optional features
  - disable unit tests
- Record changes in source package metadata
- The function *Pmap* transforms repositories into repositories where some of the source packages were transformed using Build profiles.

The Bootstrap Problem

- R is the initial repository.
- B_0 is the minimal build system.
- B is the final binary repository.
- $Src(R)$ retrieves all source packages from a repository R.

$$
\begin{aligned}
R_1 &= Pmap_1(R) \\
S_1 &= Src(R_1) \\
B_1 &= Bin(Compilable(S_1, B_0)) \cup B_0 \\
\\
R_2 &= Pmap_2(R_1) \\
S_2 &= Src(R_2) \\
B_2 &= Bin(Compilable(S_2, B_1)) \cup B_0 \\
&\dots \\
R_n &= Pmap_n(R_{n-1}) \\
S_n &= Src(R_n) \\
B_n &= Bin(Compilable(S_n, B_{n-1})) = B
\end{aligned}
$$

Bootstrap Workflow

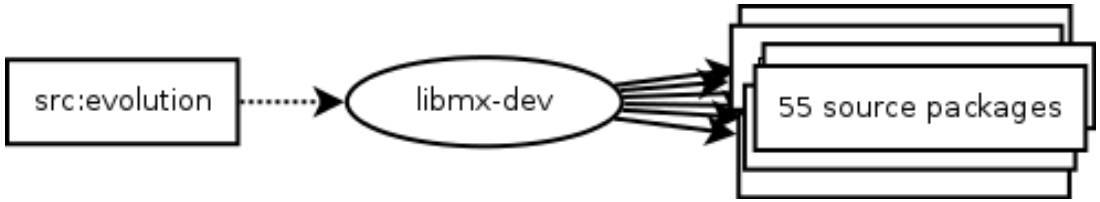
1. Select binary packages for minimal build system and cross compile them for the new platform
2. Build all source packages that can be built without breaking cycles (fixpoint algorithm in the paper)
3. Create Build Graph and extract SCC
4. If the amount of existing build profiles is not enough to make build graph acyclic:
   - use heuristics to find source packages to add build profiles to
   - modify the respective source packages
   - go back to 3
5. Feedback Vertex Set algorithm selects source packages to profile build and a build order is deduced from the now acyclic graph

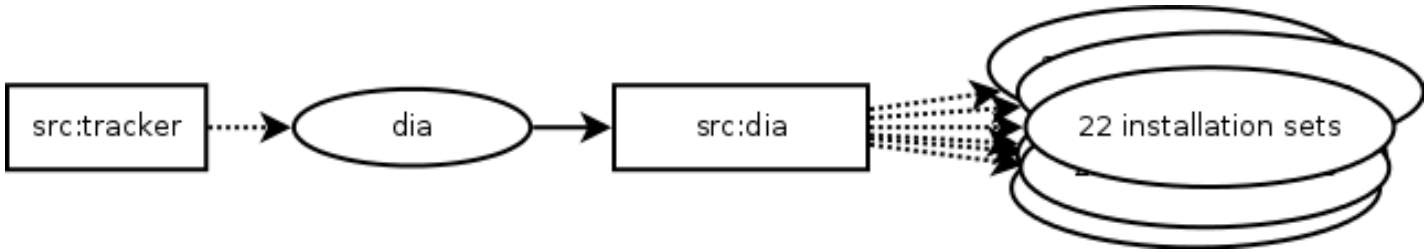
Heuristics

- Goal: finding build dependencies to add to a build profile
- Heuristics are needed because the investigation can only be done by a human developer
  - Investigation of source code and build system
  - Decision about best software engineering practices (cross building vs. build profile vs. package splitting . . .)
  - Decision about trade-off of different kinds of build profiles
  - Decision about long-term viability
  - Communication with upstream developers
- 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

Simple Heuristics

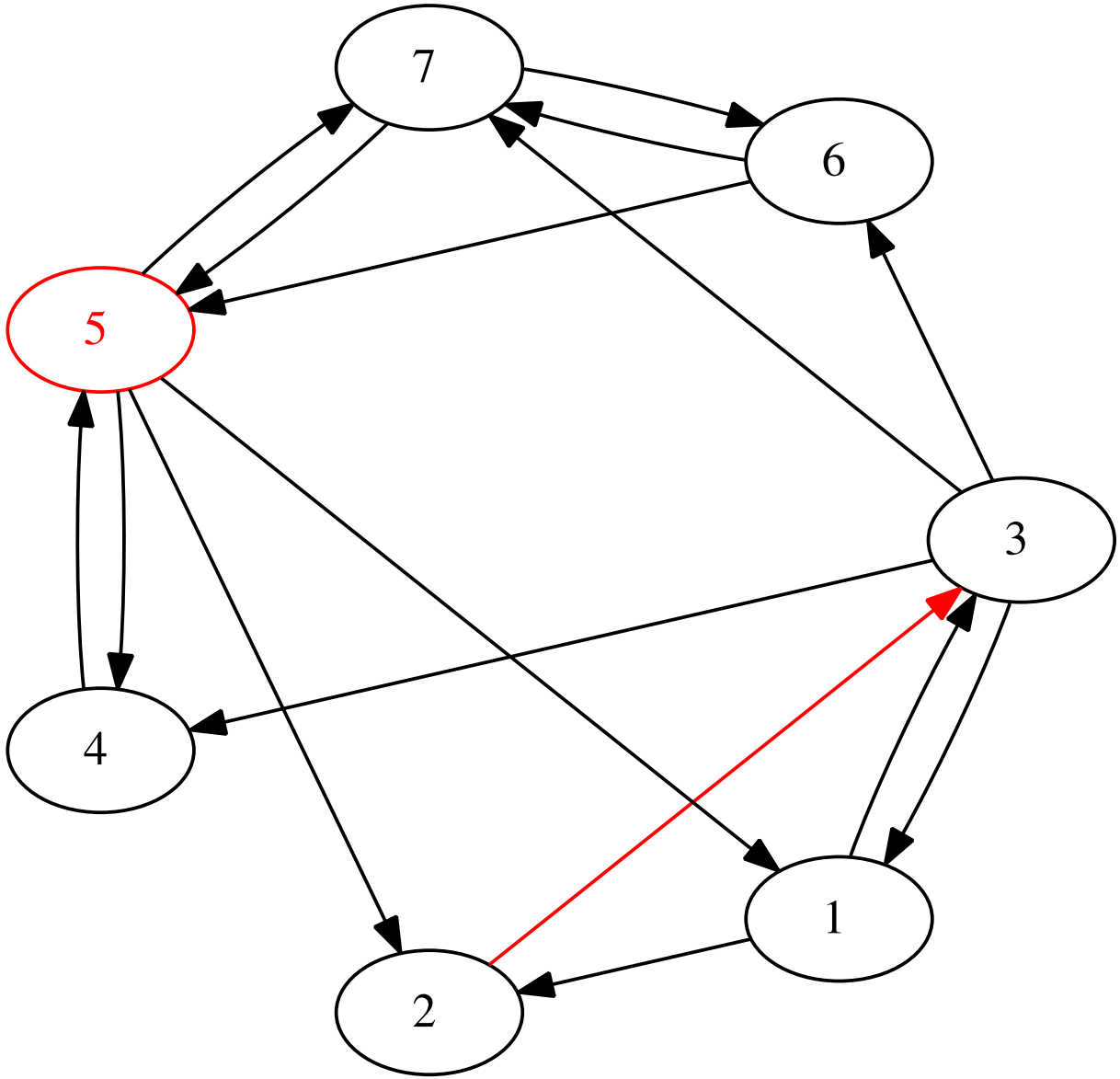
- Ratio based heuristics



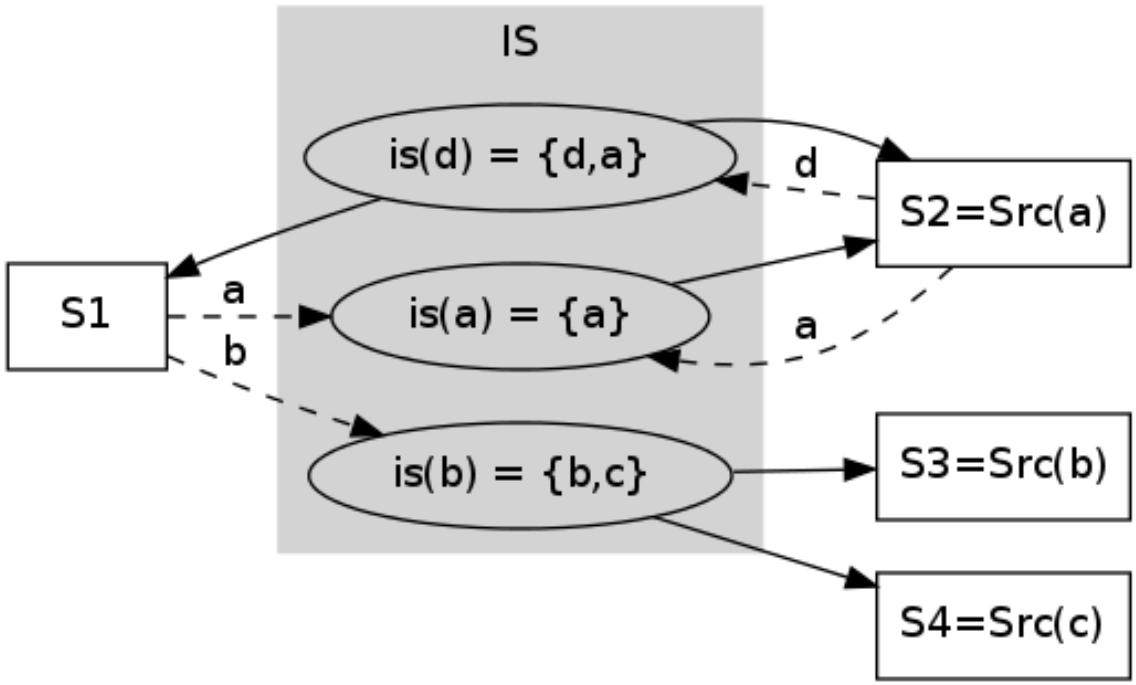


- Amount of missing dependencies
- Amount of missing "weak" dependencies (build dependencies commonly used for documentation generation and unit tests)

Strong bridges and strong articulation points



Small cycles and edges with most cycles through them



Feedback Arc Set Algorithm

1. Remove all self-cycles and add them to the FAS
2. Find cycles up to length N
3. Remove edge with most cycles through them and add it to the FAS
4. If cycles remain, go back to 3, otherwise continue
5. If graph is still cyclic, increment N and go back to 2
6. Return obtained FAS

Toolset

- Freely Available (LGPL3+).
- 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
- Existing tools are used where possible

Test Setup

- Debian Sid, January 2013
  - more than 38000 binary packages
  - more than 18000 source packages

Benchmark Results

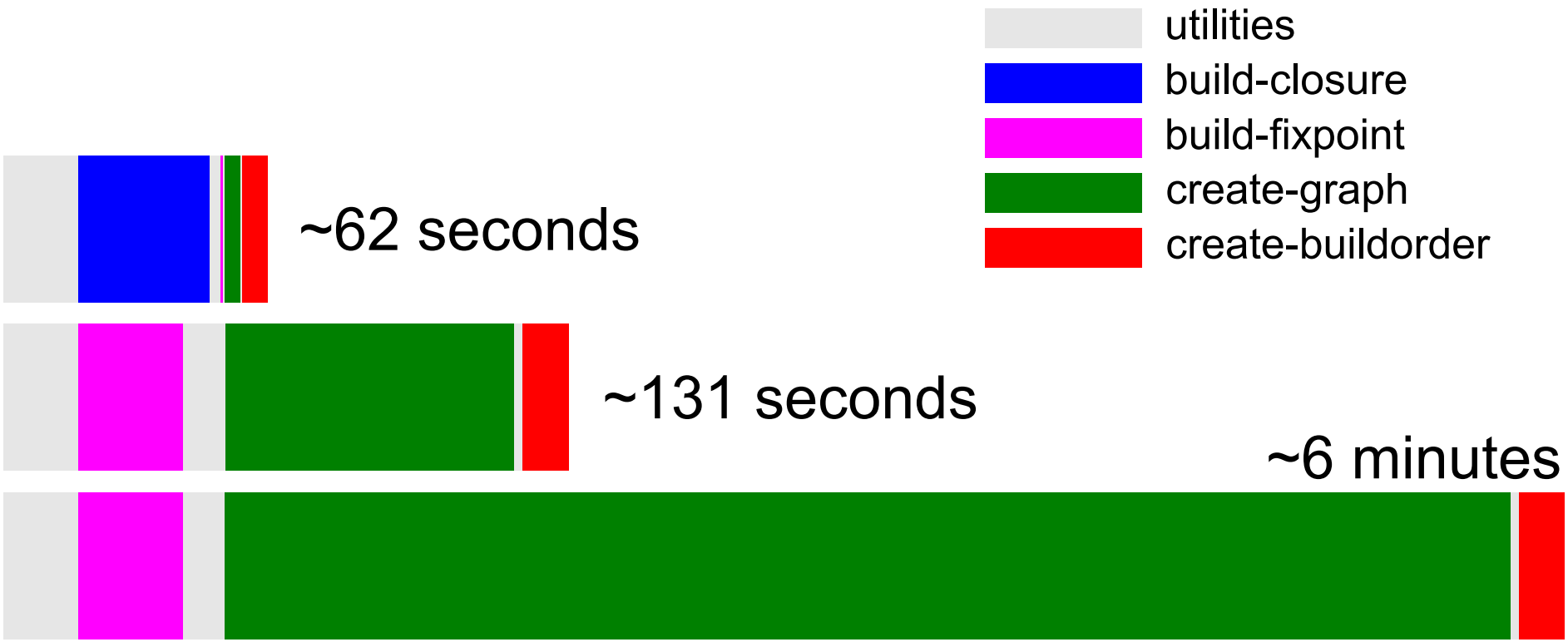


Figure : Execution time using a self-contained repository (top), normal execution (middle) and with computing strong dependencies (bottom)

Quality of Feedback Arc Set Algorithm

- Information of droppability manually gathered and extracted from Gentoo Linux
- Biggest strongly connected component with 993 vertices and 9420 edges

cycle length	modified sources	FAS size	removable
4	53	95	0.91 %
6	54	99	0.93 %
8	57	96	0.91 %
10	57	99	0.92 %
12	53	93	0.91 %

Conclusions and Future Work

- Bootstrapping FOSS distributions used to be a year long manual process and it can now be automatic, deterministic and fast
- This paper lies the theoretical foundations and the analysis is only based on the packages metadata. These tools are going to be used this summer to effectively bootstrap the Debian distribution to a new hardware platform.

- Improve our heuristics
- Add more algorithms
- Extend the support of the tool set to other software distributions.
- Possible use in a different context?

Questions?

Comparison with other FAS algorithms