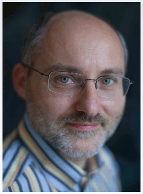
Mancoosi tools for the analysis and quality assurance of FOSS distributions

Ralf Treinen

UFR Informatique
Université Paris Diderot
`treinen@pps.jussieu.fr`

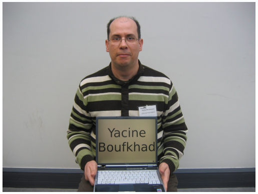
pkgsrcCon Berlin, March 23, 2013

Joint work with the Mancoosi team at Paris-Diderot


Roberto Di Cosmo

Pietro Abate

Jaap Boender



Yacine Boufkhad

Jérôme Vouillon

Zack

Our research direction

Our long-term goal

Apply tools and method from computer science to advance the quality of Free and Open Source Software.

Why are we doing this?

- We are scientists working on formal methods
- We are users and/or contributors to FOSS projects

Where we can help

Package-based software distributions:

1. Better tools to install packages
2. Better tools to assess the quality of distributions

Our research direction

Our long-term goal

Apply tools and method from computer science to advance the quality of Free and Open Source Software.

Why are we doing this?

- We are scientists working on formal methods
- We are users and/or contributors to FOSS projects

Where we can help

Package-based software distributions:

1. Better tools to install packages
2. Better tools to assess the quality of distributions

Our research direction

Our long-term goal

Apply tools and method from computer science to advance the quality of Free and Open Source Software.

Why are we doing this?

- We are scientists working on formal methods
- We are users and/or contributors to FOSS projects

Where we can help

Package-based software distributions:

1. Better tools to install packages
2. Better tools to assess the quality of distributions

(Binary) packages in Debian

$$\text{Package} = \begin{cases}\text{some files}\\ \text{some scripts}\\ \text{metadata}\end{cases}$$

- Identification
- Inter-package rel.
  - Dependencies
  - Conflicts
- Feature declarations
- Other
  - Package maintainer
  - Textual descriptions
  - ...

Example (package metadata)

```
Package: aterm
Version: 0.4.2-11
Section: x11
Installed-Size: 280
Maintainer: Göran Weinholt ...
Architecture: i386
Depends: libc6 (>= 2.3.2.ds1-4),
  libice6 | xlibs (>> 4.1.0), ...
Conflicts: suidmanager (<< 0.50)
Provides: x-terminal-emulator
...
```

Installation process in Debian

Phase	Trace
User request	`# apt-get install aterm`

Constraint resolution	`Reading package lists... Done`
	`Building dependency tree... Done`
	`The following extra packages will be installed:`
	`  libafterimage0`
	`The following NEW packages will be installed`
	`  aterm libafterimage0`
	`0 upgraded, 2 newly installed, 0 to remove and 1786 not upgraded.`
	`Need to get 386kB of archives.`
	`After unpacking 807kB of additional disk space will be used.`
	`Do you want to continue [Y/n]? Y`

Package retrieval	`Get: 1 http://debian.ens-cachan.fr testing/main libafterimage0 2.2.8-2 [301kB]`
	`Get: 2 http://debian.ens-cachan.fr testing/main aterm 1.0.1-4 [84.4kB]`
	`Fetched 386kB in 0s (410kB/s)`

Pre-Inst Script	

Unpacking	`Selecting previously deselected package libafterimage0.`
	`(Reading database ... 294774 files and directories currently installed.)`
	`Unpacking libafterimage0 (from .../libafterimage0_2.2.8-2_i386.deb) ...`
	`Selecting previously deselected package aterm.`
	`Unpacking aterm (from .../aterm_1.0.1-4_i386.deb) ...`

Post-Inst Script	`Setting up libafterimage0 (2.2.8-2) ...`
	`Setting up aterm (1.0.1-4) ...`

- *each* phase can fail
- efforts should be made to identify errors as early as possible

Installation process in Debian

Phase	Trace
User request	`# apt-get install aterm`

Constraint resolution	`Reading package lists... Done`
	`Building dependency tree... Done`
	`The following extra packages will be installed:`
	`  libafterimage0`
	`The following NEW packages will be installed`
	`  aterm libafterimage0`
	`0 upgraded, 2 newly installed, 0 to remove and 1786 not upgraded.`
	`Need to get 386kB of archives.`
	`After unpacking 807kB of additional disk space will be used.`
	`Do you want to continue [Y/n]? Y`

Package retrieval	`Get: 1 http://debian.ens-cachan.fr testing/main libafterimage0 2.2.8-2 [301kB]`
	`Get: 2 http://debian.ens-cachan.fr testing/main aterm 1.0.1-4 [84.4kB]`
	`Fetched 386kB in 0s (410kB/s)`

Pre-Inst Script	

Unpacking	`Selecting previously deselected package libafterimage0.`
	`(Reading database ... 294774 files and directories currently installed.)`
	`Unpacking libafterimage0 (from .../libafterimage0_2.2.8-2_i386.deb) ...`
	`Selecting previously deselected package aterm.`
	`Unpacking aterm (from .../aterm_1.0.1-4_i386.deb) ...`

Post-Inst Script	`Setting up libafterimage0 (2.2.8-2) ...`
	`Setting up aterm (1.0.1-4) ...`

- *each* phase can fail
- efforts should be made to identify errors as early as possible

Our Setting

Meta-data of packages

- Core inter-package relationships :
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

Global analysis

- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to local-only checks (e.g. checking that all packages mentioned in metadata exist)

Our Setting

Meta-data of packages

- Core inter-package relationships :
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

Global analysis

- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to local-only checks (e.g. checking that all packages mentioned in metadata exist)

Our Setting

Meta-data of packages

- Core inter-package relationships :
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

Global analysis

- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to local-only checks (e.g. checking that all packages mentioned in metadata exist)

Our Setting

Meta-data of packages

- Core inter-package relationships :
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

Global analysis

- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to local-only checks (e.g. checking that all packages mentioned in metadata exist)

Our Setting

Meta-data of packages

- Core inter-package relationships :
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

Global analysis

- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to local-only checks (e.g. checking that all packages mentioned in metadata exist)

Our Setting

Meta-data of packages

- Core inter-package relationships :
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

Global analysis

- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to local-only checks (e.g. checking that all packages mentioned in metadata exist)

Our Setting

Meta-data of packages

- Core inter-package relationships :
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

Global analysis

- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to local-only checks (e.g. checking that all packages mentioned in metadata exist)

At the beginning: a quite basic problem

- Given a repository R of packages and a package $p \in R$, is p installable w.r.t. R?
- That is: Does there exist $I \subseteq R$ such that
  - is *in peace*: no conflicts inside R;
  - is *abundant*: all dependencies in R satisfied.
- That means: installable in a completely empty environment.

At the beginning: a quite basic problem

- Given a repository R of packages and a package $p \in R$, is p installable w.r.t. R?
- That is: Does there exist $I \subseteq R$ such that
  - does the job: $p \in I$;
  - is *in peace*: no conflicts inside R;
  - is *abundant*: all dependencies in R satisfied.
- That means: installable in a completely empty environment.

At the beginning: a quite basic problem

- Given a repository R of packages and a package $p \in R$, is p installable w.r.t. R?
- That is: Does there exist $I \subseteq R$ such that
  - does the job: $p \in I$;
  - is *in peace*: no conflicts inside R;
  - is *abundant*: all dependencies in R satisfied.
- That means: installable in a completely empty environment.

At the beginning: a quite basic problem

- Given a repository R of packages and a package $p \in R$, is p installable w.r.t. R?
- That is: Does there exist $I \subseteq R$ such that
  - does the job: $p \in I$;
  - is *in peace*: no conflicts inside R;
  - is *abundant*: all dependencies in R satisfied.
- That means: installable in a completely empty environment.

At the beginning: a quite basic problem

- Given a repository R of packages and a package $p \in R$, is p installable w.r.t. R?
- That is: Does there exist $I \subseteq R$ such that
  - does the job: $p \in I$;
  - is *in peace*: no conflicts inside R;
  - is *abundant*: all dependencies in R satisfied.
- That means: installable in a completely empty environment.

At the beginning: a quite basic problem

- Given a repository R of packages and a package $p \in R$, is p installable w.r.t. R?
- That is: Does there exist $I \subseteq R$ such that
  - does the job: $p \in I$;
  - is *in peace*: no conflicts inside R;
  - is *abundant*: all dependencies in R satisfied.
- That means: installable in a completely empty environment.

Example

Repository R

Package:	a	Package:	b	Package:	d
Version:	1	Version:	2	Version:	3
Depends:	b (≥ 2) | d	Conflicts:	d		
Package:	a	Package:	c	Package:	d
Version:	2	Version:	3	Version:	5
Depends:	c (> 1)	Depends:	d (> 3)		
		Conflicts:	d ($= 5$)		

Is a installable?

- $(a, 1)$ is installable. Why?
- $(a, 2)$ is *not* installable. Why?

edos/dose-distcheck

- 2005: Tools `edos-debcheck` and `edos-rpmcheck`
- Very efficient, using SAT-solver technology, and caching of results obtained for various packages in the distribution.
- Today: `dose-distcheck`, part of the dose3 tool suite.
- Time for a demonstration ...

edos/dose-distcheck

- 2005: Tools `edos-debcheck` and `edos-rpmcheck`
- Very efficient, using SAT-solver technology, and caching of results obtained for various packages in the distribution.
- Today: `dose-distcheck`, part of the dose3 tool suite.
- Time for a demonstration . . .

edos/dose-distcheck

- 2005: Tools `edos-debcheck` and `edos-rpmcheck`
- Very efficient, using SAT-solver technology, and caching of results obtained for various packages in the distribution.
- Today: `dose-distcheck`, part of the `dose3` tool suite.
- Time for a demonstration . . .

edos/dose-distcheck

- 2005: Tools `edos-debcheck` and `edos-rpmcheck`
- Very efficient, using SAT-solver technology, and caching of results obtained for various packages in the distribution.
- Today: `dose-distcheck`, part of the `dose3` tool suite.
- Time for a demonstration ...

Debian weather

- Running on `edos.debian.net` (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between `arch=all` and arch-specific
- Date since when package uninstallable
- Explanation of failed installability
- Demo . . .

Debian weather

- Running on `edos.debian.net` (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between `arch=all` and arch-specific
- Date since when package uninstallable
- Explanation of failed installability
- Demo . . .

Debian weather

- Running on `edos.debian.net` (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between `arch=all` and arch-specific
- Date since when package uninstallable
- Explanation of failed installability
- Demo ...

Debian weather

- Running on `edos.debian.net` (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between `arch=all` and arch-specific
- Date since when package uninstallable
- Explanation of failed installability
- Demo . . .

Debian weather

- Running on `edos.debian.net` (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between `arch=all` and arch-specific
- Date since when package uninstallable
- Explanation of failed installability
- Demo . . .

Debian weather

- Running on `edos.debian.net` (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between `arch=all` and arch-specific
- Date since when package uninstallable
- Explanation of failed installability
- Demo . . .

Debian weather

- Running on `edos.debian.net` (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between `arch=all` and arch-specific
- Date since when package uninstallable
- Explanation of failed installability
- Demo ...

More uses of distcheck in Debian

- `emdebian`: check installability of package before uploading new (versions of) packages to the archive
- Build-dependencies:
  - turn a build-dependency (conflict) into a normal dependency (conflict) of a dummy package
  - `edos-builddepcheck`: (currently) a wrapper that generates a new repository, then runs `edos-debcheck` on it
  - Used by Debian auto-builders to avoid useless attempts to create build environments.

More uses of distcheck in Debian

- `emdebian`: check installability of package before uploading new (versions of) packages to the archive
- Build-dependencies:
    - turn a build-dependency (conflict) into a normal dependency (conflict) of a dummy package
    - `edos-builddepcheck`: (currently) a wrapper that generates a new repository, then runs `edos-debcheck` on it
    - Used by Debian auto-builders to avoid useless attempts to create build environments.

More uses of distcheck in Debian

- `emdebian`: check installability of package before uploading new (versions of) packages to the archive
- Build-dependencies:
  - turn a build-dependency (conflict) into a normal dependency (conflict) of a dummy package
  - `edos-builddepcheck`: (currently) a wrapper that generates a new repository, then runs `edos-debcheck` on it
  - Used by Debian auto-builders to avoid useless attempts to create build environments.

More uses of distcheck in Debian

- `emdebian`: check installability of package before uploading new (versions of) packages to the archive
- Build-dependencies:
  - turn a build-dependency (conflict) into a normal dependency (conflict) of a dummy package
  - `edos-builddepcheck`: (currently) a wrapper that generates a new repository, then runs `edos-debcheck` on it
  - Used by Debian auto-builders to avoid useless attempts to create build environments.

More uses of distcheck in Debian

- `emdebian`: check installability of package before uploading new (versions of) packages to the archive
- Build-dependencies:
  - turn a build-dependency (conflict) into a normal dependency (conflict) of a dummy package
  - `edos-builddepcheck`: (currently) a wrapper that generates a new repository, then runs `edos-debcheck` on it
  - Used by Debian auto-builders to avoid useless attempts to create build environments.

Detecting file conflicts

- Goal: detect cases where two packages can be installed at the same time, but doing so causes an error since one package tries to highjack a file owned by another package.
- Algorithm:
  - Look at the Debian `Contents` file, compute all pairs of packages that contain a common file (Debian sid: $\sim$ 1000 pairs)
  - Use `dose-debcheck` to select pairs that are installable together (Debian sid: $\sim$ 170 pairs)
  - Test installation in a `chroot`
- See the list of bugs on `edos.debian.net`

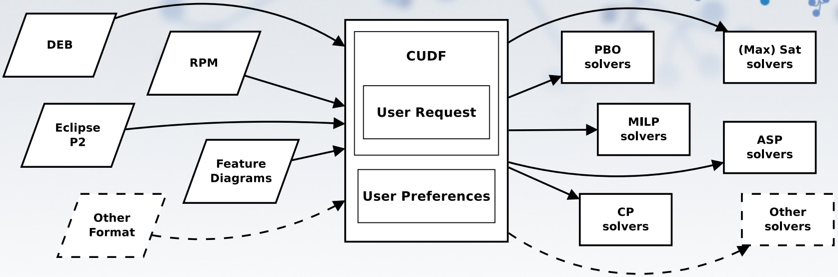
Detecting file conflicts

- Goal: detect cases where two packages can be installed at the same time, but doing so causes an error since one package tries to highjack a file owned by another package.
- Algorithm:
  - Look at the Debian `Contents` file, compute all pairs of packages that contain a common file (Debian sid: ~ 1000 pairs)
  - Use `dose-debcheck` to select pairs that are installable together (Debian sid: ~ 170 pairs)
  - Test installation in a `chroot`
- See the list of bugs on `edos.debian.net`

Detecting file conflicts

- Goal: detect cases where two packages can be installed at the same time, but doing so causes an error since one package tries to highjack a file owned by another package.
- Algorithm:
  - Look at the Debian `Contents` file, compute all pairs of packages that contain a common file (Debian sid: ~ 1000 pairs)
  - Use `dose-debcheck` to select pairs that are installable together (Debian sid: ~ 170 pairs)
  - Test installation in a `chroot`
- See the list of bugs on `edos.debian.net`

A Universal Format for Package Metadata



Translators to CUDF know about . . .

- specific format and semantics of version numbers
  (*Is* `0:7.00008.a~-1` $>$ `7.8.a-0.1` *?*)
- distribution-specific quirks
  (*What does it mean for a package to conflict with itself?*)
- the installation model
  (*Is it possible to install two packages of same name and different version?*)

Installability is a hard problem

What makes the problem hard

Two features that together make the problem NP-complete:

- Disjunctions in dependencies (may be implicit: Provides, or multiple available versions of packages)
- Conflicts (may be implicit: two packages of the same name and different version may be in implicit conflict)

The good news

Modern solving techniques (SAT solvers, or others) cope very well with analyzing distribution files.

Easy cases

The problem becomes computationally trivial when there are

- no disjunctions (explicit or implicit)
- or no conflicts (explicit or implicit)

Installability is a hard problem

What makes the problem hard

Two features that together make the problem NP-complete:

- Disjunctions in dependencies (may be implicit: Provides, or multiple available versions of packages)
- Conflicts (may be implicit: two packages of the same name and different version may be in implicit conflict)

The good news

Modern solving techniques (SAT solvers, or others) cope very well with analyzing distribution files.

Easy cases

The problem becomes computationally trivial when there are

- no disjunctions (explicit or implicit)
- or no conflicts (explicit or implicit)

Installability is a hard problem

What makes the problem hard

Two features that together make the problem NP-complete:

- Disjunctions in dependencies (may be implicit: Provides, or multiple available versions of packages)
- Conflicts (may be implicit: two packages of the same name and different version may be in implicit conflict)

The good news

Modern solving techniques (SAT solvers, or others) cope very well with analyzing distribution files.

Easy cases

The problem becomes computationally trivial when there are

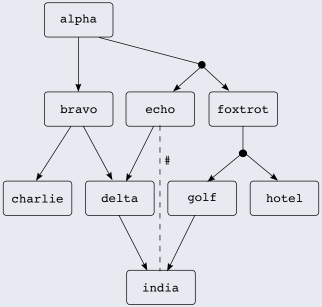
- no disjunctions (explicit or implicit)
- or no conflicts (explicit or implicit)

Finding strong dependencies

Definition

Strong dependency: A dependency that is a logical consequence of all the package relations.

Example



alpha strongly depends on foxtrot

Learning from the future of a distribution

Two different questions that we have worked on:

- If we upgrade a particular package p, what are the other packages that (in their current version) become uninstallable? These are the packages that will have to be upgraded together with p
- If the current version of a package p is found uninstallable w.r.t. the current repository: can this be solved by upgrading *other* packages in the distribution? If not, that means that p has to upgraded!

And this is done with *distcheck* too!

Learning from the future of a distribution

Two different questions that we have worked on:

- If we upgrade a particular package *p*, what are the other packages that (in their current version) become uninstallable? These are the packages that will have to be upgraded together with *p*
- If the current version of a package *p* is found uninstallable w.r.t. the current repository: can this be solved by upgrading *other* packages in the distribution? If not, that means that *p* has to upgraded!

And this is done with *distcheck* too!

Learning from the future of a distribution

Two different questions that we have worked on:

- If we upgrade a particular package p, what are the other packages that (in their current version) become uninstallable? These are the packages that will have to be upgraded together with p
- If the current version of a package p is found uninstallable w.r.t. the current repository: can this be solved by upgrading *other* packages in the distribution? If not, that means that p has to upgraded!

And this is done with *distcheck* too!

What's the future of a distribution?

- New packages may be created
- Packages may be removed
- Infinitely many possible future versions of packages
- Future versions of packages may change their dependencies/conflicts in an arbitrary way

What's the future of a distribution?

- New packages may be created
- Packages may be removed
- Infinitely many possible future versions of packages
- Future versions of packages may change their dependencies/conflicts in an arbitrary way

What's the future of a distribution?

- New packages may be created
- Packages may be removed
- Infinitely many possible future versions of packages
- Future versions of packages may change their dependencies/conflicts in an arbitrary way

What's the future of a distribution?

- New packages may be created
- Packages may be removed
- Infinitely many possible future versions of packages
- Future versions of packages may change their dependencies/conflicts in an arbitrary way

Example 1: Is (*foo*,1) installable?

```
Package: foo
Version: 1
Depends: baz (= 2.5) | bar (= 2.3),
  bar (> 2.6) | baz (< 2.3)

Package: bar
Version: 2

Package: baz
Version: 2
Conflicts: bar (< 3)
```

Example 1: Is (*foo*,1) *outdated*?

```
Package: foo
Version: 1
Depends: baz (= 2.5) | bar (= 2.3),
  bar (> 2.6) | baz (< 2.3)

Package: bar
Version: 2

Package: baz
Version: 2
Conflicts: bar (< 3)
```

Example 2: Is (*foo*,1) outdated?

```
Package: foo
Version: 1
Depends: baz (= 2.5) | bar (= 2.3),
  bar (> 2.6) | baz (< 2.3)

Package: bar
Version: 2.3

Package: baz
Version: 2.5
Conflicts: bar (> 2.6)
```

Results: challenging packages in Debian

Source	Version	Target Version	#(BP)
python-defaults	2.5.2-3	≥ 3	1079
python-defaults	2.5.2-3	$2.6 \leq . < 3$	1075
e2fsprogs	1.41.3-1	any	139
ghc6	6.8.2dfsg1-1	$\geq$ 6.8.2+	136
libio-compress-base-perl	2.012-1	$\geq$ 2.012.	80
libcompress-raw-zlib-perl	2.012-1	$\geq$ 2.012.	80
libio-compress-zlib-perl	2.012-1	$\geq$ 2.012.	79
icedove	2.0.0.19-1	$>$ 2.1-0	78
iceweasel	3.0.6-1	$>$ 3.1	70
haskell-mtl	1.1.0.0-2	$\geq$ 1.1.0.0+	48
sip4-qt3	4.7.6-1	$>$ 4.8	47
ghc6	6.8.2dfsg1-1	6.8.2dfsg1+ $\leq . <$ 6.8.2+	36

Understanding co-installability issues

Identify co-installability issues

Find quickly and concisely all pairs of components that are incompatible.

Graphical visualisation and debugging of repositories

Present the co-installaibility issues to the repository mantainer in a compact and usable way, to allow him to focus on the real problem, and non on traversing a huge graph.

Base for further future analyses

Develop tools and theory that allow to manipulate co-installability issues efficiently, to enable more complex analysis, typically for repository evolution.

Understanding co-installability issues

Identify co-installability issues

Find quickly and concisely all pairs of components that are incompatible.

Graphical visualisation and debugging of repositories

Present the co-installaibility issues to the repository mantainer in a compact and usable way, to allow him to focus on the real problem, and non on traversing a huge graph.

Base for further future analyses

Develop tools and theory that allow to manipulate co-installability issues efficiently, to enable more complex analysis, typically for repository evolution.

Understanding co-installability issues

Identify co-installability issues

Find quickly and concisely all pairs of components that are incompatible.

Graphical visualisation and debugging of repositories

Present the co-installaibility issues to the repository mantainer in a compact and usable way, to allow him to focus on the real problem, and non on traversing a huge graph.

Base for further future analyses

Develop tools and theory that allow to manipulate co-installability issues efficiently, to enable more complex analysis, typically for repository evolution.

The tool

Main techniques

- drop package relations that are irrelevant for co-installability
- identify packages that behave the same w.r.t. co-installability

Results on Mainstream GNU/Linux Distributions

	Debian		Ubuntu		Mandriva	
	before	after	before	after	before	after
Packages	28919	1038	7277	100	7601	84
Dependencies	124246	619	31069	29	38599	8
Conflicts	1146	985	82	60	78	62
Median cone size	38	1	38	1	59	1
Avg. cone size	66	1.7	84	1.3	153	1.1
Max. cone size	1134	15	842	4	1016	5
Running time (s)		10.6		1.19		11.6

The tool

Main techniques

- drop package relations that are irrelevant for co-installability
- identify packages that behave the same w.r.t. co-installability

Results on Mainstream GNU/Linux Distributions

	Debian		Ubuntu		Mandriva	
	before	after	before	after	before	after
Packages	28919	1038	7277	100	7601	84
Dependencies	124246	619	31069	29	38599	8
Conflicts	1146	985	82	60	78	62
Median cone size	38	1	38	1	59	1
Avg. cone size	66	1.7	84	1.3	153	1.1
Max. cone size	1134	15	842	4	1016	5
Running time (s)		10.6		1.19		11.6

Funded Research Projects

Past and present projects:

- $1/2004 \longrightarrow 6/2007$: edos www.edos-project.org
- $2/2008 \longrightarrow 5/2011$: mancoosi managing software complexity
- $12/2010 \longrightarrow 3/2014$: Aeolus

Thanks to our sponsors!

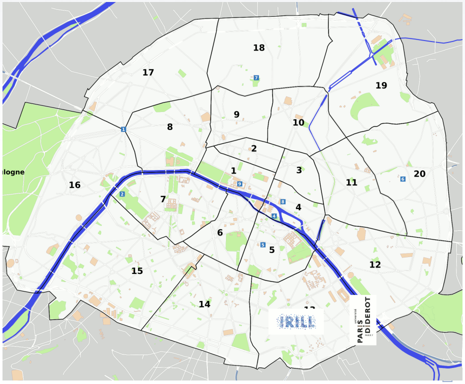
Information Society Technologies

SEVENTH FRAMEWORK PROGRAMME

AGENCE NATIONALE DE LA RECHERCHE ANR

IRILL

- Center for Research and Innovation on Free Software
- Founders: Universities Paris 6 and 7, INRIA
- Recent activities : Mozilla performance week, European LLVM conference, FusionForge developers meeting, LibreOffice conference, GNU hackers meeting, . . .



18
17
19
9
10
8
2
1
3
20
11
logne
16
7
4
6
5
12
15
14
IRILL
PARIS DIDEROT