Leveraging solver preferences
to tame your package managers

Roberto Di Cosmo (I+i+D), Pietro Abate (D), Stefano Zacchiroli (i+D), Fabrice Le Fessant (i+I), Louis Gesbert (O) and also Jérôme Vouillon (i+D), Ralf Treinen (i+D)

Université Paris (D)iderot, (I)NRIA, (i)rill, and (O)CamlPro

September 5th, 2014
OCaml 2014
Today: *an aspect* of our work on OPAM at Irill

Ten years of research on package management ...
- two european projects: EDOS and MANCOOSI ([www.mancoosi.org](http://www.mancoosi.org))
- bridging research communities, dependency solver competition
- beautiful results, stable and efficient OCaml tools and libraries

... used as foundation of the `op-get` OPAM package manager

- `libcudf` CUDF manipulation library: `opam show cudf`
- `dose` Mancoosi toolbox: `opam show dose`

Relevant literature for this talk

- Di Cosmo, Leroy, Treinen, Vouillon et al *Managing the complexity of large FOSS package-based software distributions*. ASE 2006
Package managers are all around us

**Definition of Package Manager (Wikipedia)**

... tools to automate ... *installing, upgrading, configuring, and removing* software packages ... *in a consistent manner*. It typically maintains a database of software *dependencies* and *version* information ...

**Some package managers**

**Binary distributions** apt, aptitude, yum,...

**Source distributions** portage, *BSD ports, homebrew, opam*...

**Language specific** PyPI, Eclipse P2, (opam), ...

**Application specific** steam, ...

**Decentralised** 0install, ...

**Functional approach** nixOS, disnixOS,...

... you name it
Architecture of a package management system

Many distinct tasks are performed in a package management system:

**Server side**
Maintain a *coherent* set of packages when we add, build, remove, update packages. This includes spotting packages that are no longer installable (or co-installable), due to *dependency issues*.

**Client side**
- *fetch and authenticate* metadata and packages
- *dependency solver* to find a solution to dependency constraints
- *user preferences* to pick the right solution
- *deploy* the chosen solution

Focus on two aspects that are really *common to all*
Dependency solving: how hard is it?

Theorem

The following problems are NP-complete:
- installability of a single package
- co-installability of a set of packages

Proof idea

Equivalence of dependency resolution and boolean satisfiability.


Alternative proof of NP-hardness (Daniel Burrows, 2008)

Encode Sudokus as a package installation problem, left as an exercise
### Application: find uninstallable packages in a repository

#### Basic idea
- Package installations can be encoded as Boolean Satisfiability problems.
- Just call a SAT solver on each package in the repository (may be tens of thousands! is this a bad idea?)

#### Good news: it's feasible in practice
- Modern SAT solvers perform well on practical instances.
- Jérôme Vouillon’s specialised solver, now part of the dose library, is engineered to check installability of tens thousands of packages in a few seconds.
Debian: this technology is in use since 2006

qa.debian.org/dose (wrapper around dose-distcheck)

Packages not installable in scenario unstable_main

In a pair ra,n, n is the number of packages that are build for that architecture, m is the number of packages with Architecture=all.

Summary for the last 7 days

<table>
<thead>
<tr>
<th>Date</th>
<th>amd64</th>
<th>arm64</th>
<th>armel</th>
<th>armhf</th>
<th>hurd-i386</th>
<th>i386</th>
<th>kfreebsd-amd64</th>
<th>kfreebsd-i386</th>
</tr>
</thead>
<tbody>
<tr>
<td>Today's Weather:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014-09-01 05:00:03</td>
<td>123/76</td>
<td>239/4516</td>
<td>133/230</td>
<td>123/146</td>
<td>891/2270</td>
<td>125/78</td>
<td>130/562</td>
<td>132/576</td>
</tr>
<tr>
<td>Diff</td>
<td>+4/1</td>
<td>+45/1</td>
<td>+4/1</td>
<td>+4/1</td>
<td>+9/1 -40</td>
<td>+4/1</td>
<td>+6/20 -6/0</td>
<td>+6/20 -6/0</td>
</tr>
<tr>
<td>2014-08-31 05:00:05</td>
<td>125/75</td>
<td>203/4523</td>
<td>133/229</td>
<td>123/145</td>
<td>886/2269</td>
<td>125/77</td>
<td>130/562</td>
<td>130/556</td>
</tr>
<tr>
<td>Diff</td>
<td>+7/0</td>
<td>+0/1</td>
<td>+7/0</td>
<td>+7/0</td>
<td>+5/-6/2</td>
<td>+7/0</td>
<td>+7/0 -1/0</td>
<td>+7/0 -1/0</td>
</tr>
<tr>
<td>2014-08-30 05:00:03</td>
<td>119/75</td>
<td>216/4713</td>
<td>127/229</td>
<td>117/145</td>
<td>887/2270</td>
<td>119/77</td>
<td>124/562</td>
<td>124/556</td>
</tr>
</tbody>
</table>
A word of warning

Priceless if we have a proper QA process in place, which we have not.
## Finding a needle in a haystack

Finding a solution is NP-complete, but installing and upgrading is more demanding... *how many* ways are there to install a package?

### Too many *upgrade* candidates

Suppose we have components $q_i$, for $1 \leq i \leq n$, available in versions 1 and 2, all of which are installed in version 1 on the system. We want to install a component $p$ in version 1 that depends on all of $q_1, \ldots, q_n$, any version.

Any of the $2^n$ configurations $\{(p, 1)\} \cup \{(q_i, i) | i \in 1\ldots n, 1 \leq i \leq 2\}$ is a solution.

### Which one do we choose?

- **paranoid** only install $p$
- **trendy** install $p$ and step up all $q_i$’s to version 2

... and exponentially many in between!
An exponential number of solutions???

The *sidestep* approach

- **centralize and group**: *patch tuesday, service pack*, ...
- **isolate on a monolith**: AppStore(s), ...
- **coexist**: NixOS, *backward compatibility policy, no conflicts policy*, ...

**Various ad-hoc algorithms**

Implement a *fixed* strategy for selecting a solution, e.g.

- Identify “suites” of components (e.g. *stable, testing, unstable*), let the user order them, and then try to stick to this order

  **Advantage** tries to align the system with a chosen suite

  **Disadvantage**

  - Depends on the quality of the most recent state: *not assured* for development versions, or when mixing repositories
  - Difficult and cumbersome to obtain a different behaviour
  - When things go wrong, we may *really* get lost
Installation woes: debatable solution

# sudo apt-get install debhelper
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
armagetron armagetron-common autoconf bonobo-activation codebreaker debconf
debconf-i18n debconf-utils dialog esound-common fb-music-high fontconfig
frozen-bubble-data grepmail gv intltool-debian libaiksaurus-data
libaiksaurus0c102 libatk1.0-0 libatk1.0-dev libbonobo-activation4 libbonobo2-0
libbonobo2-common libdb3 libdbd-mysql-perl libdbi-perl libeel2-data libesd0
...
The following packages will be REMOVED:
autoconf2.13 frozen-bubble frozen-bubble-lib gconf2 gnomemeeting itk3.1-dev
libbonobou2-0 libbonobou2-common libdigest-md5-perl libforms0.89 libgconf2-4
libgnome2-0 libgnome2-common libgnomeui-0 libgnomevfs2-0 libgnomevfs2-common
libgtk1.2-dev libgtk2.0-0png3 libgtk2.0-dev libmime-base64-perl
libpango1.0-dev libSDL-mixer1.2-dev libSDL-perl libSDL-ttf1.2-dev
libX11-1.2-dev libsmpeg-dev libstorable-perl nautilus tk8.3-dev tktable-dev
x-window-system x-window-system-core xaw3dg-dev xlib6g xlib6g-dev xlibmesa-dev
xlibmesa3 xlibosmesa3 xlibs-dev xpdf xpdf-reader
The following NEW packages will be installed:
armagetron-common debconf-i18n fb-music-high fontconfig intltool-debian
libaiksaurus-data libaiksaurus0c102 libeel2-data libfilehandle-unget-perl
libfontconfig1 libforms1 libgd3 libgnutls7 libgsf-1 libice-dev libice6
libid10 liblz4 libmagick5.5.7 libmail-mbox-messageparser-perl
libmysqlclient12 libncursesw5 libnet-daemon-perl libnewt0.51 libpaper1
libplrpc-perl libSDL-console ...

75 packages upgraded, 80 newly installed, 42 to remove and 858 not upgraded.
Need to get 67.1MB of archives. After unpacking 26.9MB will be used.
Do you want to continue? [Y/n] Abort.
Towards modular package managers
Dependency solving is NP-hard: stop coding a petty solver for every new component based system, and adopt a modular approach!

1 - Use a Common Upgrade Description Format

2 - Provide means for expressing our choice

A full fledged user preferences language to guide the solver towards our preferred solution.
1 - An excerpt from a CUDF file

```plaintext
preamble:
property: opam-version: string, opam-name: string

package: herelib
version: 3
depends: ocamlfind
conflicts: herelib
opam-name: herelib
opam-version: 109.12.00

...

package: lwt
version: 6
depends: base-threads, base-unix, camlp4, ocamlfind
conflicts: react >= 3, lwt
opam-name: lwt
opam-version: 2.4.4

...

request: opam
install: tyxml
```
2 - User preferences

A *preference expression* is built from four basic ingredients:

- **package selectors** denote in a proposed solution certain classes of packages (the ones that changed, the ones that got removed, etc.)
- **measurements** can be applied to a package selector to obtain an integer value (the number of package selected, the number of packages selected that are up-to-date, etc.)
- **maximisation/minimisation** directives to ask the solver to find a solution that maximises or minimises the value of a measurement
- **aggregation** combinations can be used to ask the solver to combine criteria in lexicographical order

For full details, see

- [www.dicosmo.org/Articles/usercriteria.pdf](http://www.dicosmo.org/Articles/usercriteria.pdf)
- [www.mancoosi.org/misc-2012/criteria/](http://www.mancoosi.org/misc-2012/criteria/)
- [opam.ocaml.org/doc/Specifying_Solver_Preferences.html](http://opam.ocaml.org/doc/Specifying_Solver_Preferences.html)
User preferences: package selectors

- **solution** packages installed in the solution proposed by the solver
- **down, up** packages downgraded or upgraded
- **removed, new** packages no longer there (removed) or not present before (new)
- **changed** packages changed (an aggregation of the above two lines)
- **request** packages *explicitly mentioned* in the *user request* ...
- **installrequest, upgraderequest** ... for installation or upgrade
User preferences: measurements

\begin{align*}
\text{count}(X) & : \text{number of packages in } X \\
\text{sum}(X,f) & : \text{sum of values of key } f \text{ over packages in } X \\
\text{notuptodate}(X) & : \text{number of packages in } X \text{ not current} \\
\text{unsat\_recommends}(X) & : \text{number of unsatisfied clauses in the recommends field of packages from } X \\
\text{aligned}(X,g1,g2) & : \text{number of packages aligned according to given criteria (see the full documentation for examples and explanations)}
\end{align*}
Optimising and combining preferences

Optimisation

We can ask for a solution that maximizes (+) or minimizes (−) each of these criteria, e.g.:

\[-\text{count(removed)}\]

specifies that we want a solution where the number of removed packages is minimised.

Aggregation

We can combine criteria in lexicographic order, e.g.

\[-\text{count(removed)}, -\text{count(changed)}\]

specifies that among all solutions where the number of removed packages is minimised, we look for one that has the smallest number of changes.
Examples preferences when installing your packages

Global focus

<table>
<thead>
<tr>
<th>paranoid</th>
<th>-count(removed),-count(changed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>trendy</td>
<td>-count(removed),-notuptodate(solution),-count(new)</td>
</tr>
</tbody>
</table>

Drawback

May upgrade *all* your packages, when you only wanted to change *a few*

Local focus

<table>
<thead>
<tr>
<th>paranoid</th>
<th>-count(removed),-notuptodate(request),-count(down),-count(changed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>trendy</td>
<td>-notuptodate(request),-count(removed),-count(down),-count(changed)</td>
</tr>
</tbody>
</table>

Drawback

may *use* not uptodate versions of dependecies of the request
Examples preferences when installing your packages, cont’d

<table>
<thead>
<tr>
<th>Local focus, alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>paranoid</td>
</tr>
<tr>
<td>-count(removed),-notuptodate(request),-notuptodate(changed),...</td>
</tr>
<tr>
<td>trendy</td>
</tr>
<tr>
<td>-notuptodate(request),-count(removed),-notuptodate(changed),...</td>
</tr>
</tbody>
</table>

**Drawback**

may leave untouched *existing* not up-to-date versions of dependencies of the request

**Bottomline**

there is no “one size fits all” solution

help us design a set of profiles with an intuitive meaning and a well defined rationale
More exotic examples

Performing upgrades

upgrade  \( -\text{count(down)}, -\text{count(removed)}, -\text{notuptodate(solution)}, -\text{count(new)} \)

priority  \( -\text{count(down)}, -\text{count(removed)}, -\text{notuptodate(solution)}, +\text{sum(solution,priority)}, -\text{count(new)} \)

Building systems

Minimal system size  \( -\text{sum(solution,installedsize)}, -\text{count(solution)} \)

Noah’s ark  \( +\text{count(solution)} \)

Noah’s ark, fresh  \( -\text{notuptodate(solution)}, +\text{count(solution)} \)

Fast bootstrap  \( -\text{sum(solution,compiletime)} \)
More exotic examples, cont’d

Repairing a broken system configuration

Use an empty request with

- **fixup simple**  \(-\text{count}(\text{changed})\)

- **fixup trendy**  \(-\text{count}(\text{changed}),-\text{count}(\text{down}),-\text{notuptodate}(\text{solution})\)

Did you notice?

All of this requires **zero** changes to the package manager code!
Available external solvers

Three external CUDF solvers packaged in Debian

$apt-cache search cudf

aspcud - CUDF solver based on Answer Set Programming
mccs - multi-criteria CUDF solver
packup - CUDF solver based on pseudo-Boolean constraints

There is also a nice solver for Java addicts

p2cudf, based on the Eclipse P2 plugin dependency resolver, available
from http://wiki.eclipse.org/Equinox/p2/CUDFResolver

They do not all support the full language of preferences: aspcud version
1.9 or later is recommended
You can use all this in opam for OCaml packages!
External solvers in opam
The solver aspcud is supported out of the box in opam since 1.0, and in 1.2 typing opam --help shows

... OPTIONS
--criteria=CRITERIA
Specify user preferences for dependency solving for this run. Overrides both $OPAMCRITERIA and $OPAMUPGRADECRITERIA. For details on the supported language, see http://opam.ocaml.org/doc/Specifying_Solver_Preferences.html. The default value is -count(down),-count(removed), -notuptodate(solution),-count(new) for upgrades, and -count(removed),-notuptodate(request),-count(down), -notuptodate(changed),-count(changed),-notuptodate(solution) otherwise.

--cudf=FILENAME
Debug option: Save the CUDF requests sent to the solver to FILENAME-<n>.cudf.

--solver=CMD
Specify the name of the external dependency solver. The default value is aspcud

...
Getting your external solver

Debian/Ubuntu user?

Lucky guy! Just `apt-get install aspcud`, and you are done

No aspcud ≥ 1.9 for you?

- go to http://cudf-solvers.irill.org
- follow the instructions
- access the *Irill CUDF solver farm*
- get your solving done *in the cloud*

Big thanks to OCamlPro folks (Benjamin, Fabrice, Gregoire, Pierre), for help setting this up.

Only real solution for low-power arch (arduino, raspberri-pi)

Absolutely need to work offline?

Try `0install` ... and if it is not enough, get the Dockerised version here: https://github.com/rdicosmo/docked-aspcud
## You can help

### Building profiles

try out different preferences, find those that appear more useful, and propose them as profiles

### Test expressivity of the preferences language

play with the preferences, check whether we need extensions (see [http://www.dicosmo.org/Articles/usercriteria.pdf](http://www.dicosmo.org/Articles/usercriteria.pdf) for existing proposals)

### Help debug opam

if you are unhappy with a proposed install/upgrade, remember to dump the CUDF file (`--cudf` option)
Conclusions

Package managers are complex: a very hard part is dependency solving! Modern package managers must share common components, in particular dependency solvers and user preference language.

You can should use external solvers and preferences in opam today!

You might tell people in other communities they are welcome to adopt the same approach.

Learn more at www.mancoosi.org, cudf-solvers.irill.org and ows.irill.org

User preferences primer at http://www.dicosmo.org/Articles/usercriteria.pdf

Questions?
Old preference combinators

Notice that some solvers (mccs, packup) only support an older preference language (still recognised by aspcud 1.9 and later): here is the correspondence table

<table>
<thead>
<tr>
<th>Old language</th>
<th>New language</th>
</tr>
</thead>
<tbody>
<tr>
<td>removed</td>
<td>count(removed)</td>
</tr>
<tr>
<td>new</td>
<td>count(new)</td>
</tr>
<tr>
<td>changed</td>
<td>count(changed)</td>
</tr>
<tr>
<td>notuptodate</td>
<td>notuptodate(solution)</td>
</tr>
<tr>
<td>unsat_recommends</td>
<td>unsat_recommends(solution)</td>
</tr>
</tbody>
</table>