Software Dependency Management

From p2 to p2cudf

Daniel Le Berre joint work with Emmanuel Lonca Pierre Marquis Anne Parrain Pascal Rapicault

CRIL-CNRS UMR 8188, Lens, France

Lococo 2011 - September 12, 2011, Perugia, Italy.

Agenda

Software dependency management

p2

p2cudf

Ongoing work based on p2cudf

Agenda

Software dependency management

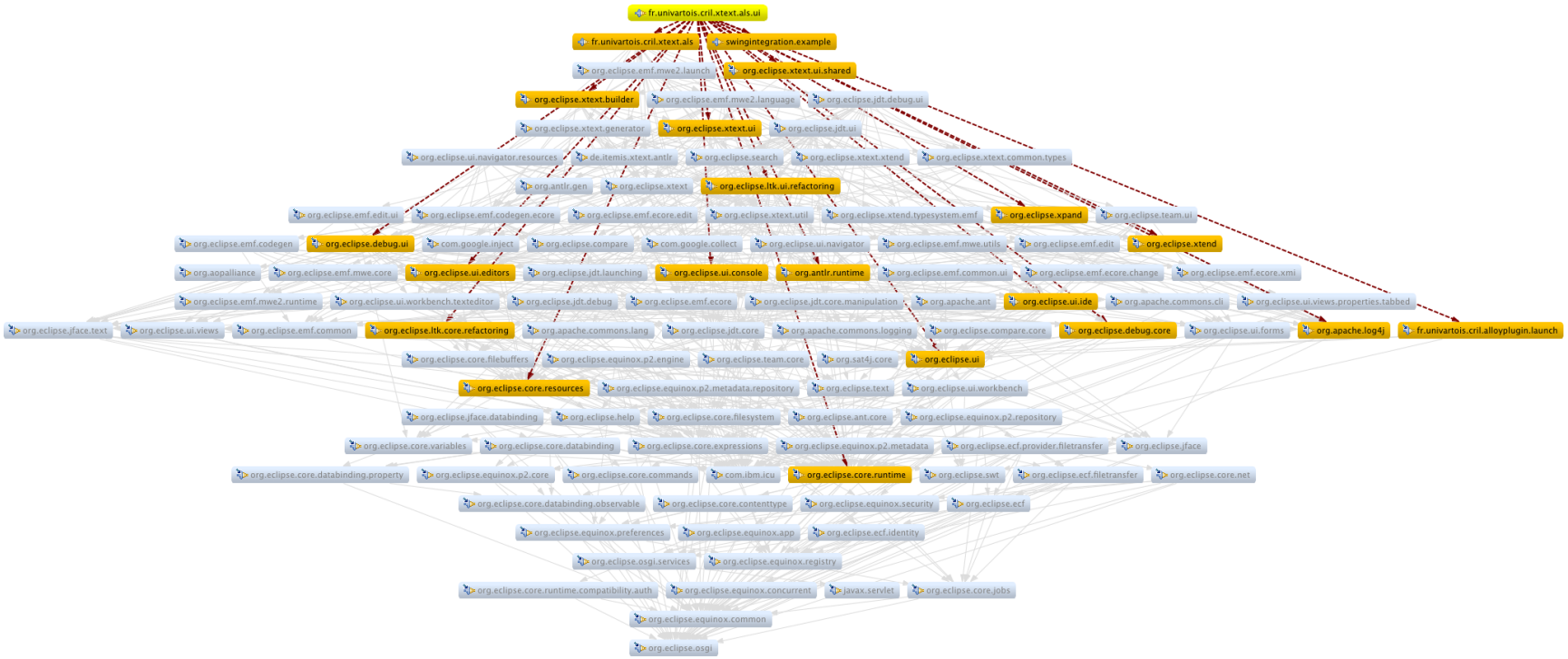
p2

p2cudf

Ongoing work based on p2cudf

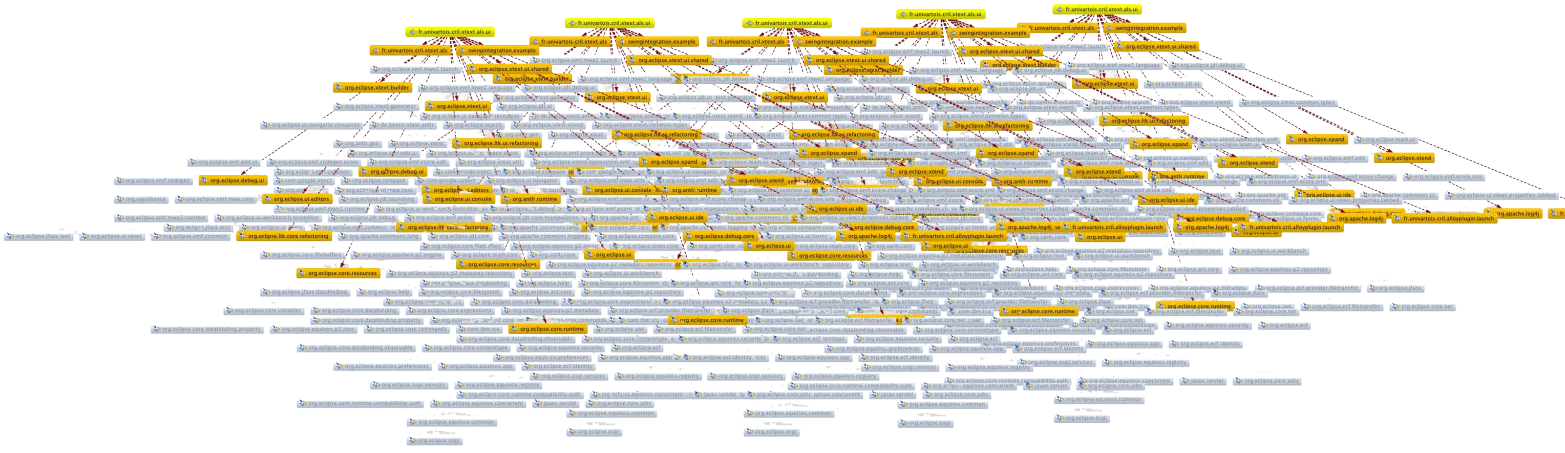
Current softwares are composite !

Alloy 4 Eclipse dependencies : 20 direct / 108 total



Current softwares are composite !

Problems occur when installing several plugins !



Current softwares are composite !

- Linux distributions : made of packages (Debian >50K packages)
- Component based software/platform (Eclipse ecosystem >3K bundles)
- Any complex software : made of libraries (Maven universe >200K libraries)
- There are requirements between the diverse components
  - capabilities can be provided by several components (disjunction)
  - some components cannot be installed together (conflicts)

Dependency Management Problem : formal definition

P a set of packages

$$P = \{mpm_1, p2cudf_1, p2cudf_2, aspcud_1, aspcud_2, rpm_1, debian_1\}$$

depends $P \rightarrow 2^{2^P}$ requirement constraints

$$P = \{mpm_1 \rightarrow \{\{p2cudf_1, p2cudf_2, aspcud_1, aspcud_2\}, \{rpm_1, debian_1\}\}\}$$

conflicts $P \rightarrow 2^P$ impossible configurations

$P = \{p2cudf_1 \rightarrow \{p2cudf_2, aspcud_1, aspcud_2\},$
$p2cudf_2 \rightarrow \{p2cudf_1, aspcud_1, aspcud_2\}\}$

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, depends, conflicts)$ iff
$\forall q \in Q, (\forall dep \in depends(q), dep \cap Q \neq \emptyset) \wedge (conflicts(q) \cap Q = \emptyset)$.

$Q_1 = \{mpm_1, p2cudf_2, debian_1\}$
$Q_2 = \{mpm_1, aspcud_1, aspcud_2, rpm_1\}$

Dependency Management Problem : formal definition

P a set of packages

$$P = \{mpm_1, p2cudf_1, p2cudf_2, aspcud_1, aspcud_2, rpm_1, debian_1\}$$

depends $P \rightarrow 2^{2^P}$ requirement constraints

$$P = \{mpm_1 \rightarrow \{\{p2cudf_1, p2cudf_2, aspcud_1, aspcud_2\}, \{rpm_1, debian_1\}\}\}$$

conflicts $P \rightarrow 2^P$ impossible configurations

$P = \{p2cudf_1 \rightarrow \{p2cudf_2, aspcud_1, aspcud_2\},$
$p2cudf_2 \rightarrow \{p2cudf_1, aspcud_1, aspcud_2\}\}$

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, depends, conflicts)$ iff
$\forall q \in Q, (\forall dep \in depends(q), dep \cap Q \neq \emptyset) \wedge (conflicts(q) \cap Q = \emptyset)$.

What is the complexity of finding if a Q containing a specific package exists ?

Just as hard as SAT : NP-complete !

See how to decide satisfiability of $(\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge a \wedge \neg c$

```
package: a
version: 1
conflicts: a = 2

package: a
version: 2
conflicts: a = 1

package: b
version: 1
conflicts: b = 2

package: b
version: 2
conflicts: b = 1

package: c
version: 1
conflicts: c = 2

package: c
version: 2
conflicts: c = 1
```

```
package: clause
version: 1
depends: a = 2 | b = 1 | c = 1

package: clause
version: 2
depends: a = 2 | b = 2 | c = 1

package: clause
version: 3
depends: a = 1

package: clause
version: 4
depends: c = 2

package: formula
version: 1
depends: clause = 1, clause = 2,
         clause = 3, clause = 4

request: satisfiability
install: formula
```

From dependencies to clauses

- Dependencies can easily be translated into clauses :

```
package: a
version: 1
depends: b = 2 | b = 1, c = 1
```

$$a_1 \rightarrow (b_2 \vee b_1) \wedge c_1$$

$$\neg a_1 \vee b_2 \vee b_1, \neg a_1 \vee c_1$$

- Conflict can easily be translated into binary clauses :

```
package: a
version: 1
conflicts: b = 2, d = 1
```

$$\neg a_1 \vee \neg b_2, \neg a_1 \vee \neg d_1$$

From decision to optimization

The issue is not to find one solution (easy for current SAT solvers), but to find a good solution

- Minimizing the number of installed packages
- Minimizing the size of installed packages
- Ensuring capacity constraints
- Keeping up to date versions of packages
- Preferring most recent packages to older ones
- ...

We need to solve an optimization problem.

How did we start working on this ?

A benefit of open sourcing research software

June 12, 2006 Email from Chris Tucker to `help@sat4j.org`, asking for help while he is working on Opium $\Rightarrow$ he mentions the EDOS project.

How did we start working on this ?

A benefit of open sourcing research software

June 12, 2006 Email from Chris Tucker to `help@sat4j.org`, asking for help while he is working on Opium $\Rightarrow$ he mentions the EDOS project.

He finally moved to another solver

We were shipping only cutting planes based solvers at that time, too slow for that particular application.

How did we start working on this ?

A benefit of open sourcing research software

June 12, 2006 Email from Chris Tucker to `help@sat4j.org`, asking for help while he is working on Opium $\Rightarrow$ he mentions the EDOS project.

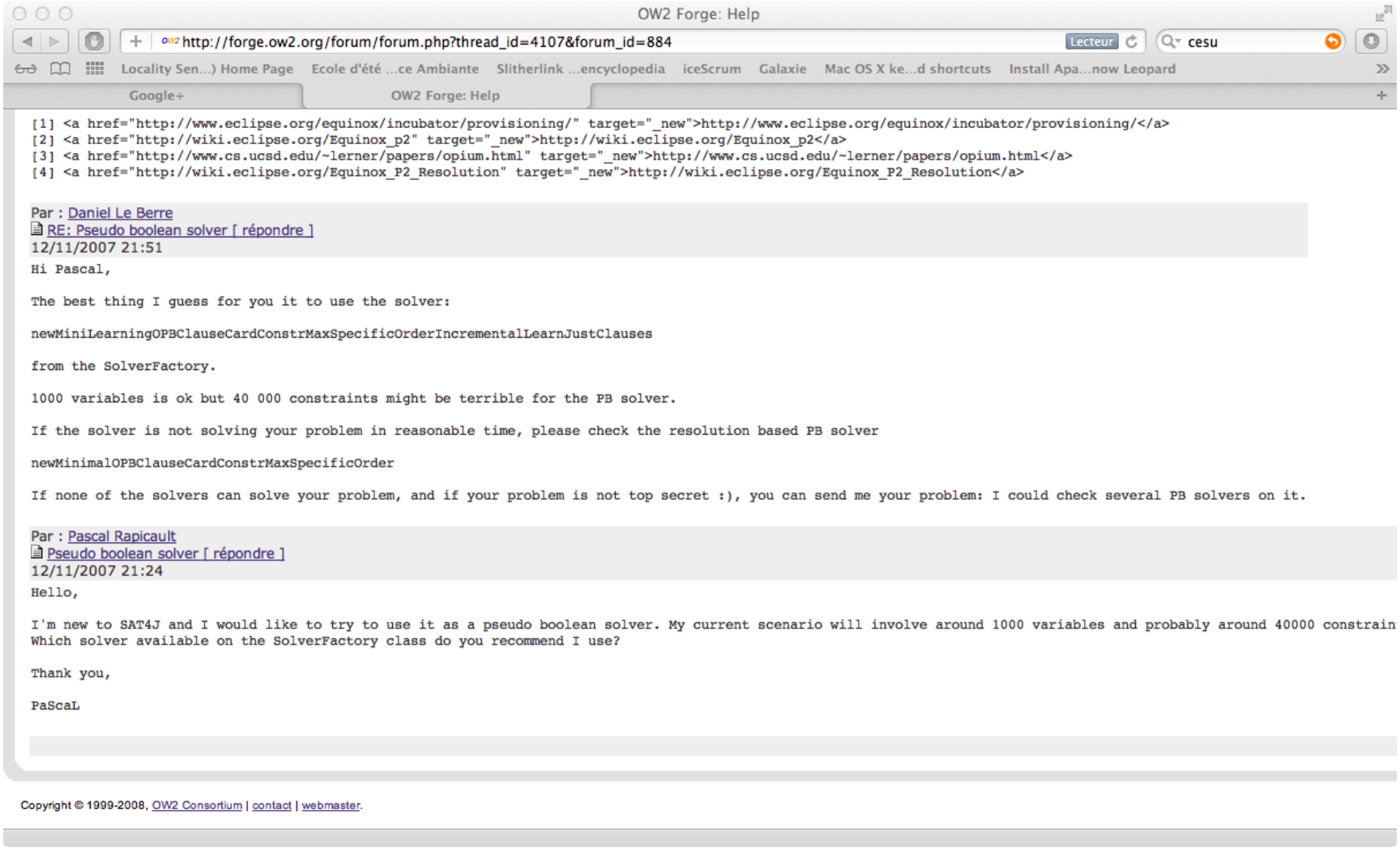
He finally moved to another solver

We were shipping only cutting planes based solvers at that time, too slow for that particular application.

November 12, 2007 Ask for help from Pascal Rapicault on Sat4j forum.

How did we start working on this ?

A benefit of open sourcing research software



UNIVERSITÉ D'ARTOIS

How did we start working on this ?

A benefit of open sourcing research software

June 12, 2006 Email from Chris Tucker to `help@sat4j.org`, asking for help while he is working on Opium $\Rightarrow$ he mentions the EDOS project.

He finally moved to another solver

We were shipping only cutting planes based solvers at that time, too slow for that particular application.

November 12, 2007 Ask for help from Pascal Rapicault on Sat4j forum.

That time it worked !

Several issues had to be solved first (IP, license, etc.)

Agenda

Software dependency management

p2

p2cudf

Ongoing work based on p2cudf


P2
A NEW LEVEL OF FEAR

The specific case of the Eclipse platform

- Open platform developed by the Eclipse foundation
- Designed for extensibility : the basic platform is enriched with plugins
- Widely adopted (more than 13M downloads for Eclipse 3.4 and 3.5)
- Many vendors ship products on top of Eclipse
- Plugins are usually coming from various, uncontrolled sources (main difference compared to Linux case).

Example of metadata

```
id=org.eclipse.swt, version=3.5.0, singleton=true
Capabilities:
    {namespace=org.eclipse.equinox.p2.iu,
     name=org.eclipse.swt, version=3.5.0}
    {namespace=org.eclipse.equinox.p2.eclipse.type
     name=bundle version=1.0.0}
    {namespace=java.package,
     name=org.eclipse.swt.graphics, version=1.0.0}
    {namespace=java.package,
     name=org.eclipse.swt.layout, version=1.2.0}
Requirements:
    {namespace=java.package,
     name=org.eclipse.swt.accessibility2,
     range=[1.0.0,2.0.0), optional=true, filter=(&(os=linux))}
    {namespace=java.package, name=org.mozilla.xpcom,
     range=[1.0.0, 1.1.0), optional=true, greed=false}
Updates:
    {namespace=org.eclipse.equinox.p2.iu, name=org.eclipse.swt,
                    range=[0.0.0, 3.5.0)}
```

Some specific notions in Eclipse

optional dependencies must be satisfied as much as possible (*recommends* in MISC 2011). Used for the drop in folder.

greedy dependencies Non greedy dependencies do not force the installation of plugins. Used for platform specific dependencies for instance.

patches allow to change the dependencies during provisioning. Specifically used when shipping a product based on Eclipse.

Example of installable unit patch for the Platform group

```
id=org.eclipse.ant.critical.fix,version=1.0.0
Capabilities:
    {namespace=org.eclipse.equinox.p2.iu,
     name=org.eclipse.ant.core.critical.fix, version=3.5.0.v2009}
Requirement Changes:
    { from={namespace=org.eclipse.equinox.p2.iu,
            name=org.eclipse.ant.core, range=[3.1.0, 3.4.0)},
      to={namespace=org.eclipse.equinox.p2.iu,
          name=org.eclipse.ant.core, range=[3.4.3]}}
    { from={namespace=org.eclipse.equinox.p2.iu,
            name=org.apache.ant, range=[1.7.1.v2009]},
      to={namespace=org.eclipse.equinox.p2.iu,
          name=org.apache.ant, range=[1.7.2.v2009]}}
Applicability Scope:
    {namespace=org.eclipse.equinox.p2.iu,
     name=org.eclipse.platform.feature.group, range=[3.5.0.v2009]}
Lifecycle:
    {namespace=org.eclipse.equinox.p2.iu, name=my.product,
     range=[1.0.0], greed=false}
Updates:
    {namespace=org.eclipse.equinox.p2.iu,
     name=org.eclipse.platform.feature.group, range=[0.0.0, 3.5.0.v2009)}
```

Encoding optional dependencies

An optional requirement is just a soft clause in MAXSAT terminology !

Using selector variables and a linear optimization function :

$$f(IU_i^k) = \bigwedge_{cap_j \in optReq(IU_i^k),} (IU_i^k \rightarrow Noop_z \vee \bigvee_{IU_x^v \in alt(cap_j)} IU_x^v) \quad (1)$$

$$minimize \sum Noop_z \quad (2)$$

Encoding greedy dependencies

Each requirement of the form "IU_i requires non greedily capability cap_j" is encoded the following way :

$$f(IU_i) = \bigwedge_{cap_j \in reqNonGreedy(IU_i),} (IU_i \rightarrow \bigvee_{IU_x^v \in alt(cap_j)} NG\_U_x^v) \quad (3)$$

Then, the non greedy IUs are associated to the IUs that require them greedily :

$$NG\_U_x^v \rightarrow \bigvee_{IU_x^v \in alt(cap_j), cap_j \in req(IU_i^j)} U_i^j \quad (4)$$

Encoding of patches

The patch encoding applies only to the requirements affected by a patch :

$$\bigwedge_{<old,new>\in patchedReqs(IU,p)} (p \rightarrow encode(new)) \wedge (\neg p \rightarrow encode(old))$$

where $encode(x)$ denote the previous encoding of dependencies.

The patch shown for the IU Platform would be encoded :

$$IU_{Platform} \rightarrow IU^{3.1.0}_{ant-core} \vee patch \quad (a)$$
$$patch \wedge IU_{Platform} \rightarrow IU^{3.4.3}_{ant-core} \quad (b)$$
$$IU_{Platform} \rightarrow IU^{1.7.1}_{ant} \vee patch \quad (a)$$
$$patch \wedge IU_{Platform} \rightarrow IU^{1.7.2}_{ant} \quad (b)$$
$$IU_{Platform} \rightarrow IU^{4.0}_{help} \quad (c)$$
$$IU_{Platform} \rightarrow IU^{4.0}_{jetty} \quad (c)$$
$$IU_{Platform} \rightarrow IU^{3.1}_{rcp} \quad (c)$$

From decision to optimization

Eclipse dependency problem is under-constrained : it admits lots of solutions. But they are not of equal quality :

1. An IU should not be installed if there is no dependency to it.
2. If several versions of the same bundle exist, the latest one should preferably be used.
3. When optional requirements exist, the optional requirements should be satisfied as much as possible.
4. User installed patches should be applied independently of the consequences of its application (i.e., the version of the IUs forced, the number of installable optional dependencies, etc.).
5. Updating an existing installation should not change packages unrelated from the request being made.

Objective function

- Each version of an IU gets a penalty as a power of K proportional to its age, the older it is the more penalized it is :

$$\sum_{IU_v^i \in versions(IU_v)} K^j \times IU_v^i \tag{5}$$

Having a penalty of K for the most recent version prevents the solver to install a plugin that is not required !

- Each selector variable $Noop_z$ variable gets a penalty to favor the installation of optional dependencies

$$\sum Noop_z \tag{6}$$

Objective function (continued)

- Each *patch* variable gets a reward R_p if it is applicable else a penalty P_p

$$\sum_{p_i \in applicablePatches()} -R_p \times p_i + \sum_{p_i \notin applicablePatches()} P_p \times p_i \quad (7)$$

- Already installed packages and Root packages should be kept installed whenever possible. However, it should be possible to update the packages found in the transitive closure of the requirements of the Root IUs :

$$\sum_{IU_v^i \in (Installed \setminus transitiveClosure(Root)) \cup Root} 1 \times IU_v^i \quad (8)$$

Objective function : global view

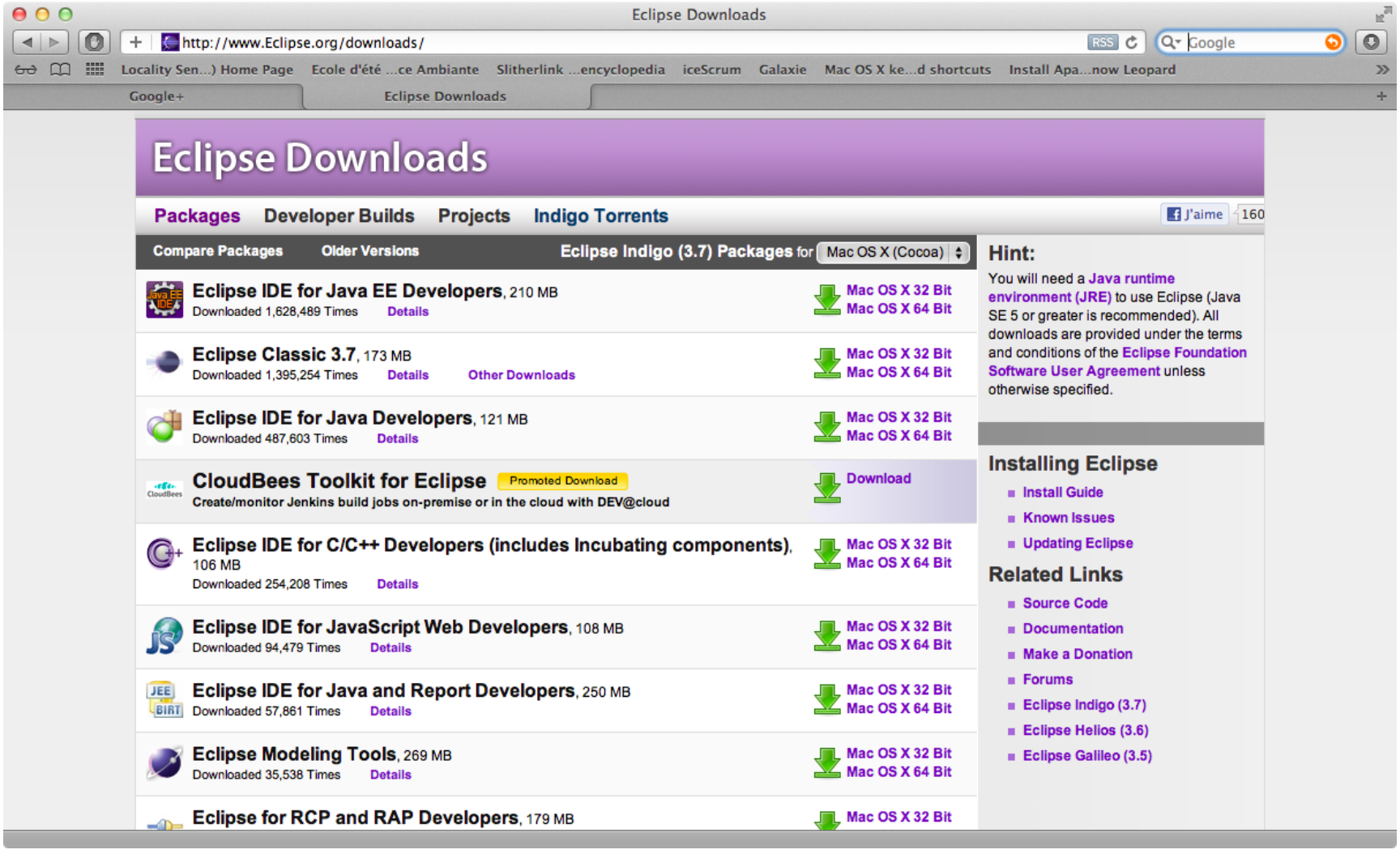
The objective function of our optimization problem is thus to minimize (5) + (6) + (7) + (8).

$$
\begin{aligned}
\textit{minimize} \quad & \sum_{IU_v^i \in versions(IU_v)} K^i \times IU_v^i \\
& + \sum Noop_x \\
& - \sum_{p_i \in applicablePatches()} R_p \times p_i \\
& + \sum_{p_i \notin applicablePatches()} P_p \times p_i \\
& + \sum_{IU_v^i \in (Installed \setminus transitiveClosure(Root)) \cup Root} 1 \times IU_v^i
\end{aligned}
$$

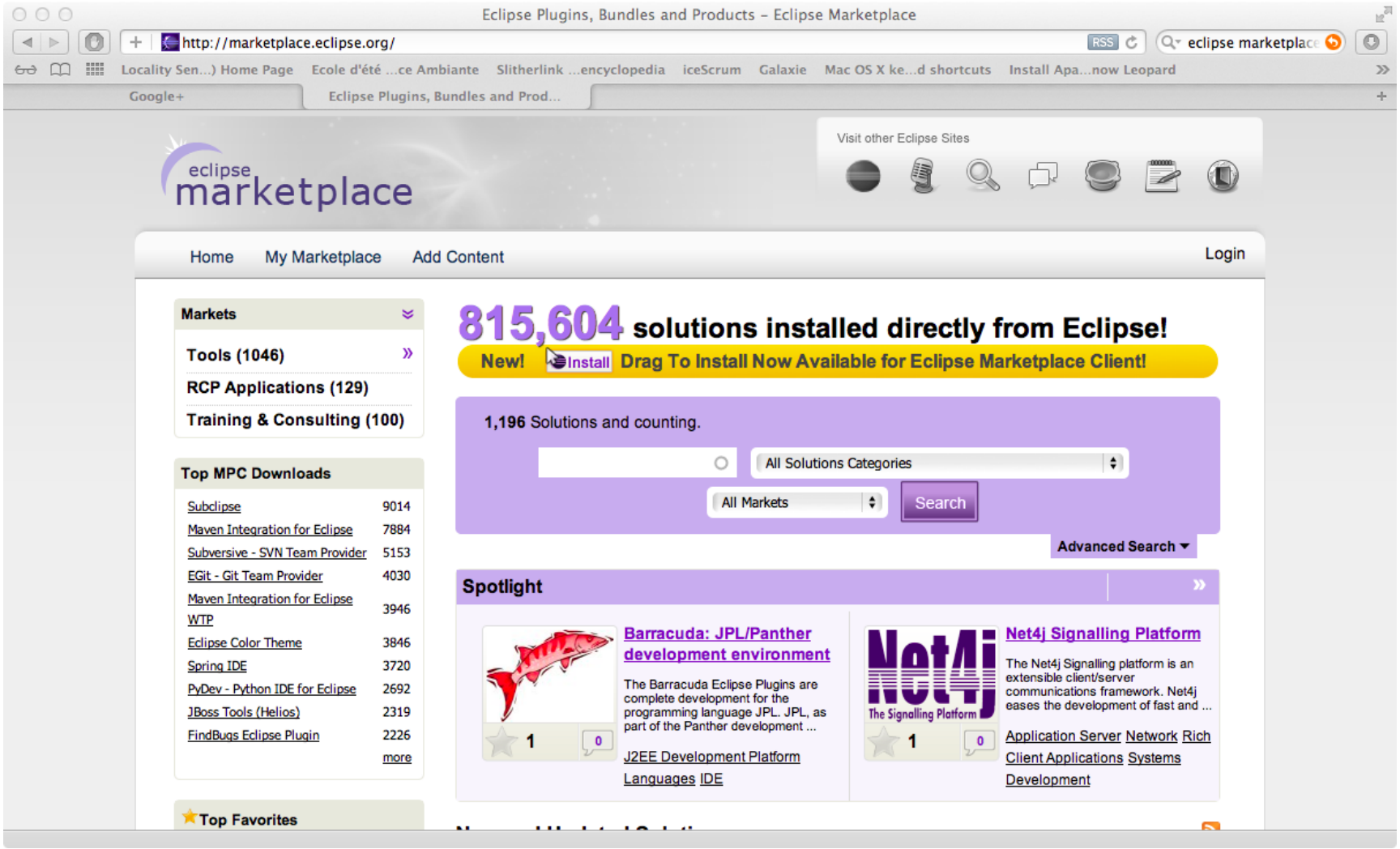
Some thoughts about that experience

- Got lucky to have the ability to work on that problem, because Eclipse needed :
  - An open source solver
  - In Java
  - With the right license
- No issues in modeling constraints (but optionality)
- Main issues met have been :
  - Real integration with Sat4j needed (explanation support, 3.5)
  - Building a better objective function (patches, stability of installations 3.5-3.6)
  - Understanding subtleties like patches and non greedy use cases.
- Main reward is that is works routinely every day since June 2008 !

Sat4j/p2 is used by the foundation for building Eclipse



Sat4j/p2 is used by Eclipse users



Agenda

Software dependency management

p2

p2cudf

Ongoing work based on p2cudf

Motivation for p2cudf

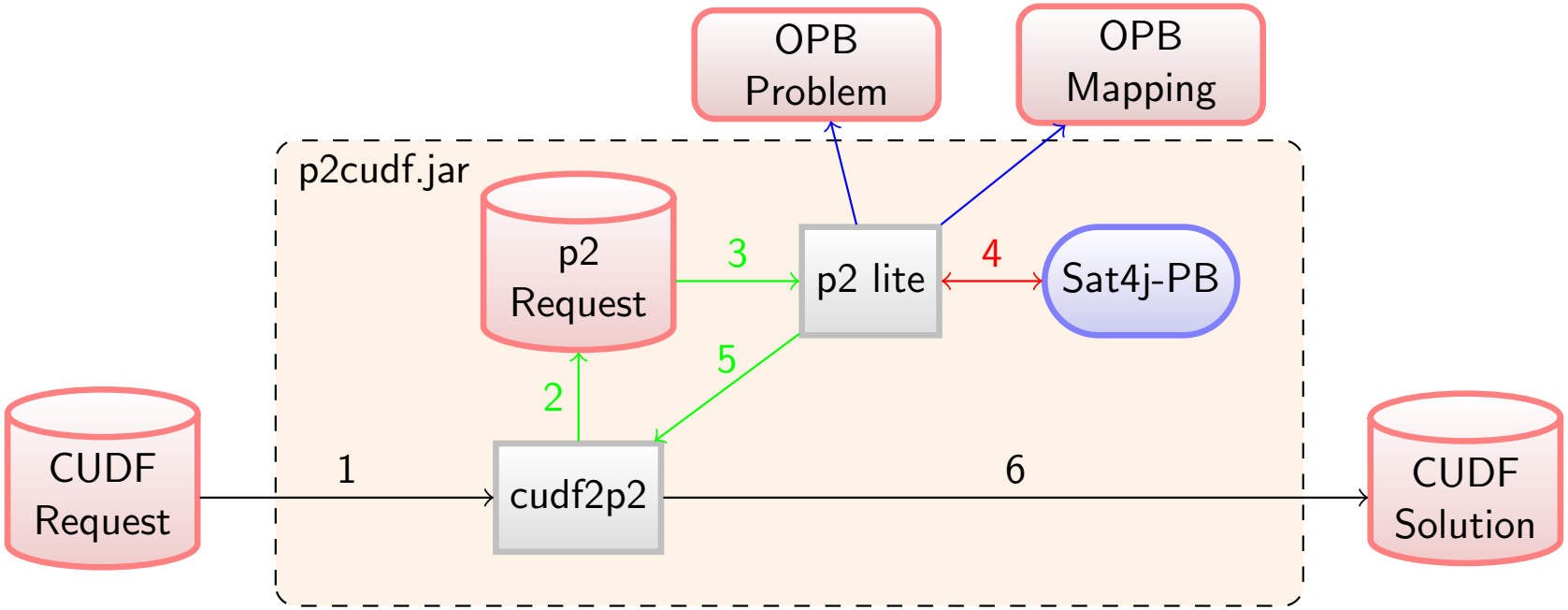
- Reuse existing knowledge about dependency management gathered from Eclipse problems to solve Linux problems
- Use Linux problems to see if p2 approach scales well
- Participate to Mancoosi internal/International Solver Competitions

Main differences with Eclipse :

- Simpler metadata (no patches, groups, etc) $\Rightarrow$ simplified Eclipse 3.5 p2 code base to avoid unnecessary abstractions
- Specific optimization function : lexicographic order of basic, counting-based, criterion to be either maximized or minimized

Initial p2cudf architecture (MISC 2010)

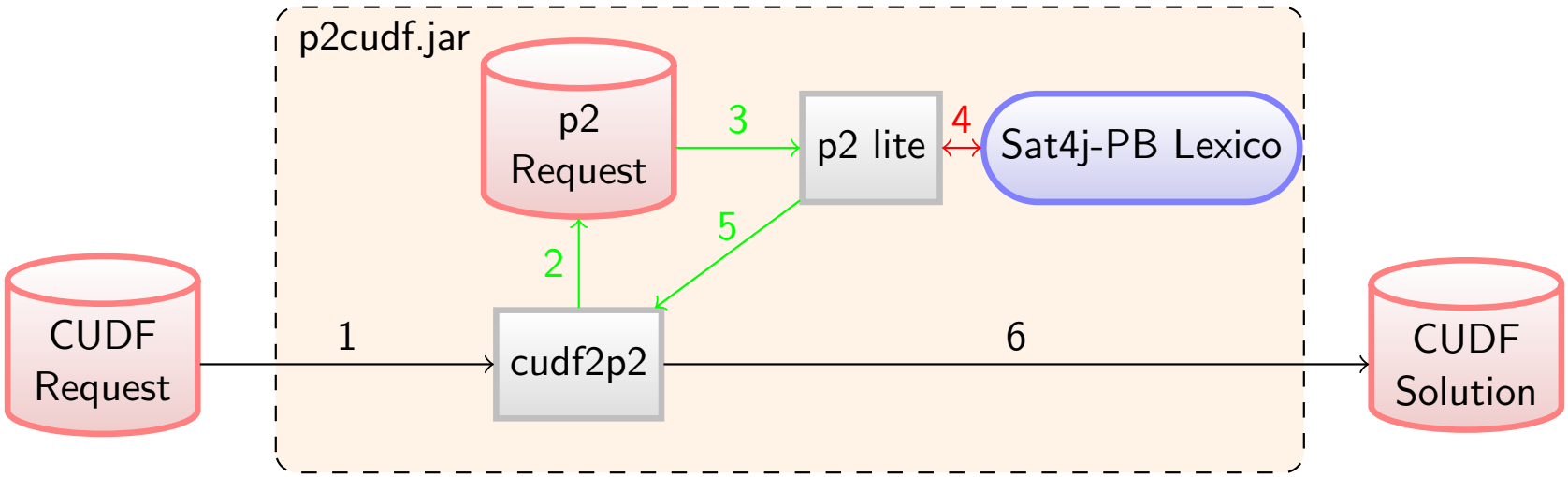
Translate the initial problem into an OPB problem (lexico-optimization translated into a single optimization function) through p2 software.



Can easily swap the underlying boolean optimization solver (e.g. wbo or msuncore).

Current p2cudf architecture (MISC 2011)

Since MISC Live 3, Sat4j contains a specific lexicographic optimization scheme



Would require a lexico optimization OPB format to allow testing others OPB engine.

Why changing the architecture ?

- Using a single objective function allows to easily try any OPB solver
- Such objective function can contain a huge number of literals and huge coefficients : p2cudf had sometimes problems to converge to the optimal solution
- Using a specific lexicographic procedure limit the size of the objective function
- a specific procedure must be implemented
- Solvers using a specific lexicographic procedure performed generally well in MISC 2010
- Sat4j PB has sometimes hard time to prove unsatisfiability, requiring proving each criterion optimal might be an issue

Lessons learned from p2cudf

- CUDF input format is great to present software dependency problems use cases
- CUDF semantic has been very hard to implement in p2cudf
- Work done in making p2 work in Eclipse has been successfully reused (e.g. aggressive slicing stage)
- All improvements on the optimization function in p2 could not be reused in p2cudf
- MISC provides many use cases, and allowed us to spot limitations in p2
- p2cudf allows us to try ideas because it is a research tool

Agenda

Software dependency management

p2

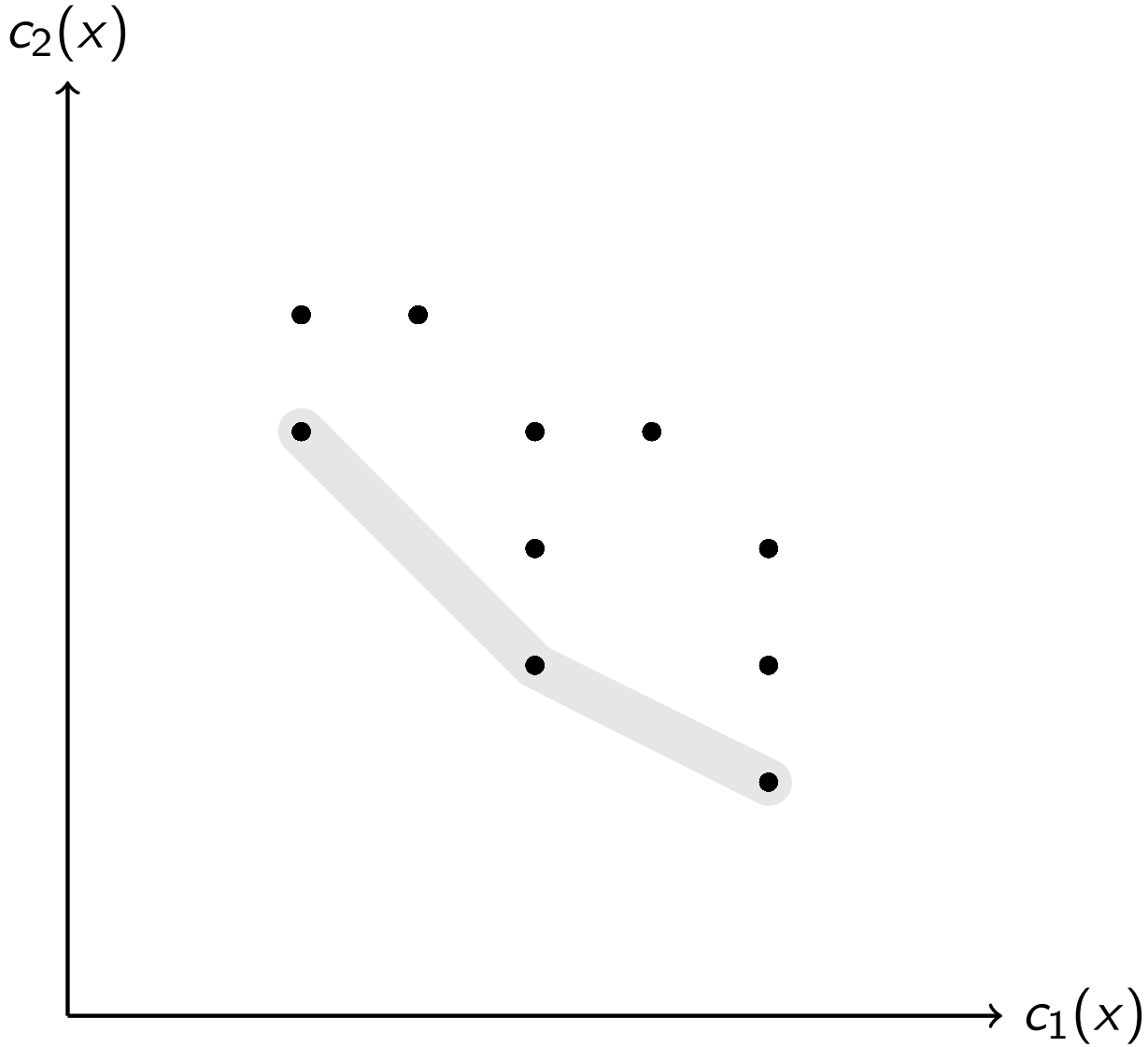
p2cudf

Ongoing work based on p2cudf

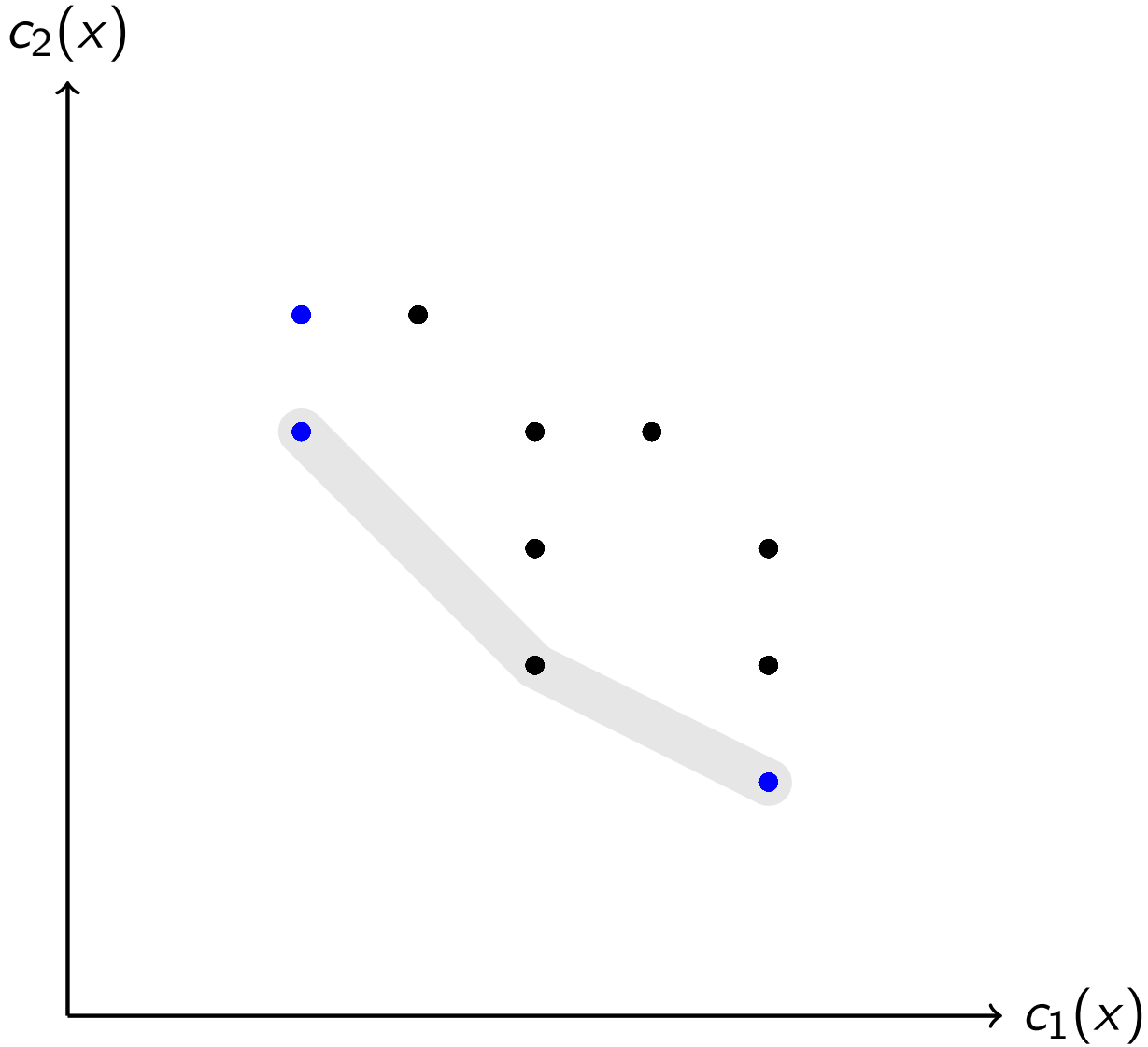
Regarding current optimization functions in p2/p2cudf

- p2 optimization function tailored thanks to user feedback : there are still some issues (esp. compensation effects), because there is no order between criterion.
- Lexico optimization has also some drawbacks : is $(-45, -128, -23, -5)$ really better than $(-46, -34, -23, -4)$?
- The idea : study possibilities to compute "balanced" solutions
- What are the consequences in terms of computation time ?

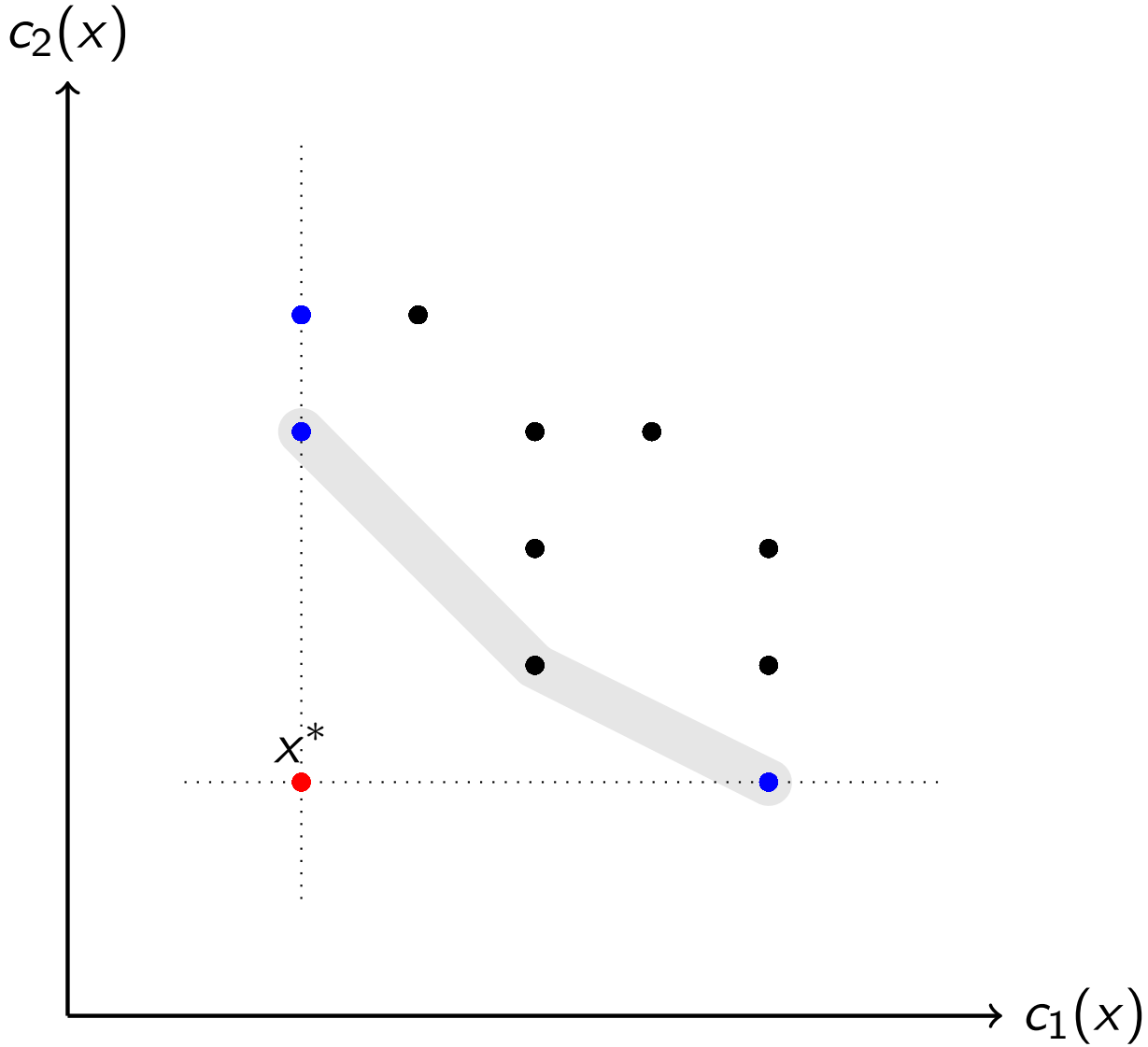
Chebyshev's distance



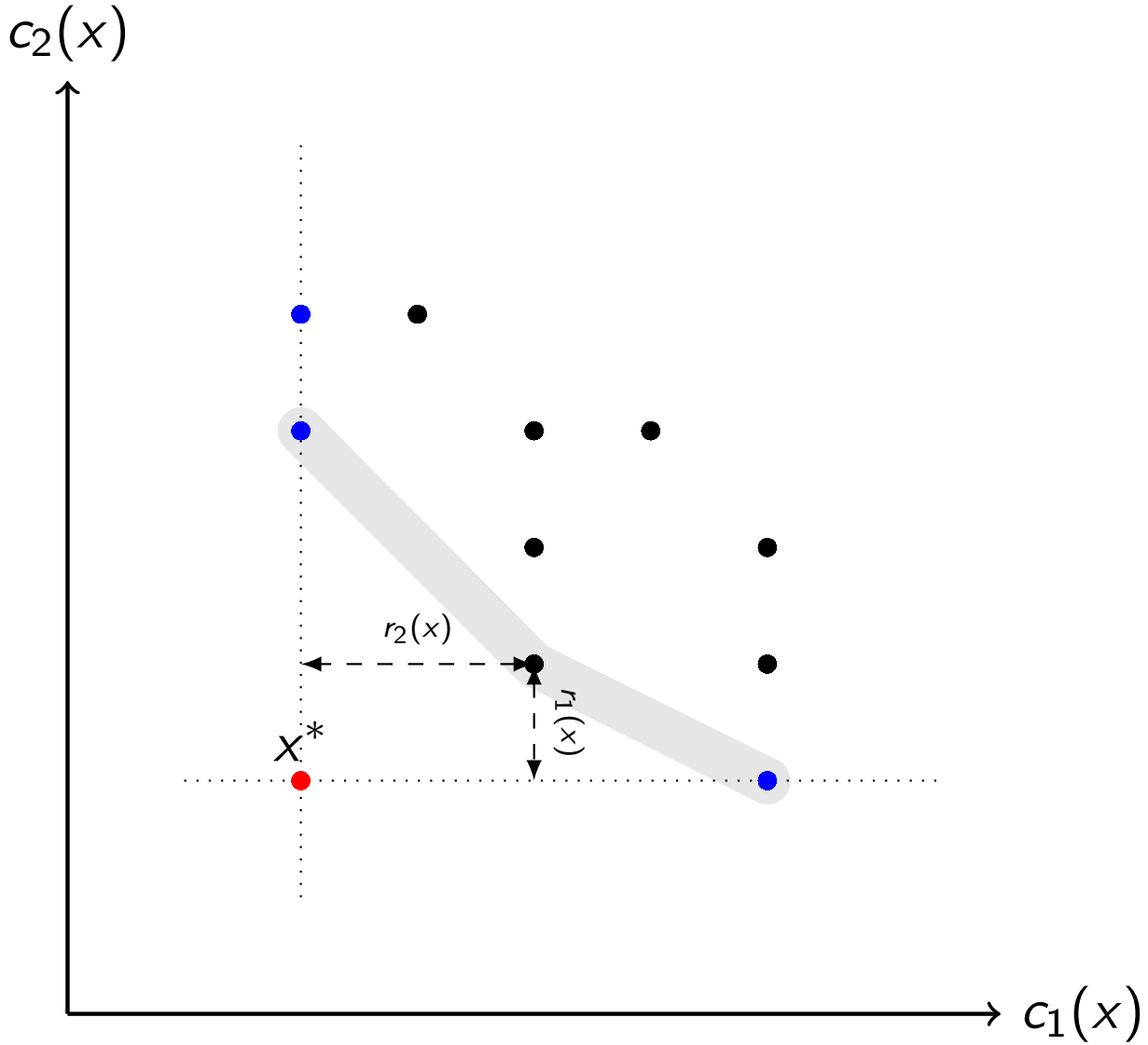
Chebyshev's distance



Chebyshev's distance



Chebyshev's distance



Chebyshev's distance

- for each criteria c_i, compute x_i^*.
- for each $x = (x_1, ..., x_n)$ solution, compute

$$chebyshev(x) = max(\{w_i(x_i^* - x_i) | x_i \in x\})$$

(w_i is a weight given to each criteria)

- find a x minimizing that value

Example

Suppose the optimal values, independently, are
$x^* = (-45, -20, 0, 0)$ with $x_1 = (-45, -128, -23, -5)$ and
$x_2 = (-46, -34, -23, -4)$ then
$chebyshev(x_1) = max(\{0, 108, 23, 5\}) = 108$ and
$chebyshev(x_2) = max(\{1, 14, 23, 4\}) = 23$ for $w_i = 1$.

Preliminary results on some MISC LIVE 3 benchmarks

instance	approach1	approach2	approach3
kwin-style-crystal.cudf	—	**3 :32.50**	3 :58.58
gnash-common-opengl.cudf	7 :50.64	**3 :29.86**	3 :47.15
kbd.cudf	9 :57.78	**5 :03.20**	5 :12.14
kdessh.cudf	—	4 :26.25	**4 :01.17**
kjumpingcube.cudf	9 :48.29	**4 :32.90**	5 :49.06
tasque.cudf	—	—	**5 :02.58**
mono-mcs.cudf	—	6 :05.82	**5 :26.81**
mono-debugger.cudf	—	—	**4 :27.17**
libmono-addins-gui0.2-cil.cudf	—	—	**6 :30.89**
plasma-widgets-workspace.cudf	—	—	**5 :50.05**
r-cran-lme4.cudf	—	6 :25.50	**4 :25.55**
plasma-dataengines-workspace.cudf	—	5 :58.67	**3 :46.45**
r-cran-rodbc.cudf	—	**4 :03.04**	4 :28.20
mono-gac.cudf	—	**4 :05.21**	4 :49.24
kommander.cudf	9 :30.01	**4 :46.28**	5 :33.73
libevolution3.0-cil.cudf	9 :21.93	8 :58.36	**4 :36.97**
openoffice.org-draw.cudf	9 :50.50	5 :12.93	**4 :51.37**
ktouch.cudf	9 :54.45	**4 :32.66**	5 :32.31
kdesudo.cudf	9 :05.84	**3 :38.39**	3 :51.66
gtwitter.cudf	—	8 :26.07	**5 :49.79**
gpc-4.1.cudf	—	6 :13.87	**4 :44.35**
ksystemlog.cudf	—	**4 :13.67**	4 :26.94
upstart-compat-sysv.cudf	—	—	**7 :54.76**
fcron.cudf	8 :06.95	**4 :15.86**	4 :34.11
audacious-plugins-extra.cudf	—	7 :05.73	**5 :20.92**

Remarks

- Running times not compatible with interactive use yet $\rightarrow$ could be an option for people not satisfied by other approaches
- Quality of the "balanced" solutions need to be checked with real users
- Chebyshev approach does not provide only pareto optimal solutions :

$$cheb(-45, -128, -23, -5) = cheb(-55, -128, -53, -65) = 108$$

- Tradeoff between quality and computation time (timeout for optimality ?)
- Implementation done on top of p2cudf not available yet.

Summary

- p2 is an open source PBO-based software dependency manager used in Eclipse since June 2008.
- p2cudf is an open source CUDF compliant solver available since January 2010.
- work on p2 has been partly reused in p2cudf
- work on p2cudf allowed to test ideas to improve p2 a wide variety of benchmarks
- the main difference between the two tools is the objective function
- p2cudf is now a tool that we use to test our ideas on boolean multi-criteria optimization
- each tool uses Sat4j so help us to improve it.