Apports et Potentiels de la Programmation par Contraintes en Optimisation Globale sous Contraintes

Michel RUEHER

Université de Nice Sophia-Antipolis / CNRS - I3S , France

CPAIOR Workshop on Hybrid Methods for NLP

15/06/10
Outline

Motivations

Basics

A Global Constraint for Safe Linear Relaxation

Computing “sharp” upper bounds

Using CSP to boost safe OBR

A challenging finite-domain optimization application

Conclusion
The Problem

We consider the continuous global optimisation problem

\[
\mathcal{P} \equiv \begin{cases} 
\min & f(x) \\
\text{s.c.} & g_i(x) = 0, \ j = 1..k \\
& g_j(x) \leq 0, \ j = k + 1..m \\
& \underline{x} \leq x \leq \bar{x}
\end{cases}
\]

with

- \( X = [\underline{x}, \bar{x}] \): a vector of intervals of \( R \)
- \( f : R^n \rightarrow R \) and \( g_j : R^n \rightarrow R \)
- Functions \( f \) and \( g_j \): are continuously differentiable on \( X \)
Trends in global optimisation

- **Performance**
  Most successful systems (Baron, αBB, ...) use local methods and linear relaxations
  → **not rigorous** (work with floats)

- **Rigour**
  Mainly rely on interval computation
  ... available systems (e.g., Globsol) are **quite slow**

- **Challenge**: to combine the advantages of both approaches in an **efficient** and **rigorous** global optimisation framework
Example of flaw due to a lack of rigour

Consider the following optimisation problem:

\[
\begin{align*}
\text{min} & \quad x \\
\text{s. t.} & \quad y - x^2 \geq 0 \\
& \quad y - x^2 \ast (x - 2) + 10^{-5} \leq 0 \\
& \quad x, y \in [-10, +10]
\end{align*}
\]

Baron 6.0 and Baron 7.2 find 0 as the minimum . . .
Basics

- Branch and Bound Algorithm
- Basics on Numeric CSP
Branch and Bound Algorithm

► **BB Algorithm:**
While $\mathcal{L} \neq \emptyset$ do
- $\% \mathcal{L}$ initialized with the input box
  - Select a box $B$ from the set of current boxes $\mathcal{L}$
  - Reduction (filtering or tightening) of $B$
  - Lower bounding of $f$ in box $B$
  - Upper bounding of $f$ in box $B$
  - Update of $\underline{f}$ and $\bar{f}$
  - Splitting of $B$ (if not empty)

► **Upper Bounding – Critical issue:**
  to prove the existence of a feasible point in a reduced box

► **Lower Bounding – Critical issue:**
  to achieve an efficient pruning
Numeric CSP

- \( \mathcal{X} = \{x_1, \ldots, x_n\} \) is a set of variables

- \( \mathbf{X} = \{X_1, \ldots, X_n\} \) is a set of domains
  (\( X_i \) contains all acceptable values for variable \( x_i \))
  \[
  X_i = [\underline{x}_i, \overline{x}_i]
  \]

- \( \mathcal{C} = \{c_1, \ldots, c_m\} \) is a set of constraints
Numeric CSP: Overall scheme

A Branch & Prune schema:

1. Pruning the search space
2. Making a choice to generate two (or more) sub-problems

▶ The pruning step → filtering techniques to reduce the size of the intervals
▶ The branching step → splits the intervals (uses heuristics to choose the variable to split)
Local consistencies

- **2B–consistency** only requires to check the Arc–Consistency property for each bound of the intervals
  
  Variable $x$ with $X = [\underline{x}, \overline{x}]$ is 2B–consistent for constraint $f(x, x_1, \ldots, x_n) = 0$ if $\underline{x}$ and $\overline{x}$ are the leftmost and the rightmost zero of $f(x, x_1, \ldots, x_n)$

- **Box–consistency**:
  
  $\rightarrow$ coarser relaxation of AC than 2B–consistency
  $\rightarrow$ better filtering

  Variable $x$ with $X = [\underline{x}, \overline{x}]$ is Box–Consistent for constraint $f(x, x_1, \ldots, x_n) = 0$ if $\underline{x}$ and $\overline{x}$ are the leftmost and the rightmost zero of $F(x, X_1, \ldots, X_n)$, the optimal interval extension of $f(x, x_1, \ldots, x_n)$
Filtering

- 2B–filtering Algorithms $\leadsto$ projection functions
- Box–filtering Algorithms $\leadsto$ monovariate version of the interval Newton method
- Based on Interval Arithmetic
Limits of Interval Arithmetic

- **Wrapping effect**: overestimate by a unique interval the image of $f$ over an interval vector

- **Dependency problem**: independence the different occurences of some variable during the evaluation of an expression

Consider $X = [0, 5]$

$X - X = [0 - 5, 5 - 0] = [-5, 5]$ instead of $[0,0]$!

$X^2 - X = [0, 25] - [0, 5] = [-5, 25]$

$X(X - 1) = [0, 5]([0, 5] - [1, 1])$

$= [0, 5][-1, 4] = [-5, 20]$
Limits of Local Consistencies

- A constraint is handled as a black-box by local consistencies (2B, BOX,...)
  - No way to catch the dependencies between constraints (amplified by constraint decomposition)
  - Splitting is behind the success for small dimensions

- Higher consistencies (KB–filtering, Bound–filtering)
  → capture some dependencies between constraints
  → visiting numerous combinations

⇒ A global constraint to handle a linear approximation with LP solvers
  → safe linear relaxations
A Global Constraint for Safe Linear Relaxation

- works on **quadratic terms and bilinear terms**
  → to rewrite power terms and product terms

  - **quadrification technique** derived from Sheraldi techniques
  - **Critical issue:** to find a good trade off between a tight relaxation and the number of generated terms

- Quadratic terms and bilinear terms are approximated by tight redundant constraints
The QUAD process

- **Reformulation**
  - capture the linear part
  - → replace non linear terms by new variable
  - eg $x^2$ by $y_i$

- **Linearisation**
  - introduce redundant linear constraints
  - → tight approximations (RLT)

- **Computing**
  - $\min(X) = x_i$ and $\max(X) = \bar{x}_i$ in $LP$
Reformulation for $x^2$

$y = x^2$ with $x \in [-4, 5]$

$L_1(y, \alpha) \equiv y \geq 2\alpha x - \alpha^2$

$L_1(y, -4) : y \geq -8x - 16$

$L_1(y, 5) : y \geq 10x - 25$

$L_2(y) \equiv y \leq (x + \bar{x})x - x \cdot \bar{x}$

$L_2(y) : y \leq x + 20$
Quad filtering algorithm

**Function** Quad_filtering (IN: \(X, C, \epsilon\)) return \(X'\)

1. **Reformulation**
   → linear inequalities \(L_i\) for the nonlinear terms in \(C\)

2. **Linearisation/relaxation of the whole system**
   → a linear system \(LR\)

3. \(X' := X'\)

4. **Pruning**:
   While reduction of some bound \(> \epsilon\) and \(\emptyset \not\in X'\) Do
   4.1 Reduce the lower and upper bounds \(x'_i\) and \(\bar{x}'_i\) of each
   initial variable \(x_i \in X\)
   → Computing \(\min\) and \(\max\) of \(X_i\) with a LP solver
   4.2 Update the coefficients of \(L_i\) according to the new bounds
Issues in the use of linear relaxation

- Coefficients of linear relaxations are scalars
  ⇒ computed with *floating point numbers*

- Efficient implementations of the simplex algorithm
  ⇒ use *floating point numbers*

- All the computations with floating point numbers require *right corrections*
Safe approximations of $L_1$

$L_1(y, \alpha) \equiv y \geq 2\alpha x - \alpha^2$

Effects of rounding:
- rounding of $2\alpha$
  $\Rightarrow$ rotation on $y$ axis
- rounding of $\alpha^2$
  $\Rightarrow$ translation on $y$ axis

Effects of rounding:
$\Rightarrow$ rotation on $y$ axis
$\Rightarrow$ translation on $y$ axis
Correction of the Simplex algorithm

Consider the following LP:

\[
\begin{align*}
\text{minimise} & \quad c^T x \\
\text{subject to} & \quad \underline{b} \leq Ax \leq \overline{b}
\end{align*}
\]

- Solution = vector \( x_R \in \mathbb{R}^n \)
- LP solver computes a vector \( x_F \in \mathbb{F}^n \neq x_R \)
- \( x_F \) is safe for the objective if \( c^T x_R \geq c^T x_F \)

- Neumaier & Shcherbina
  - cheap method to obtain a rigorous bound of the objective
    (use of the approximation solution of the dual)
Computing “sharp” upper bounds

- **Upper bounding**
  - local search
    → approximate feasible point $x_{approx}$
  - epsilon inflation process and proof
    → provide a feasible box $x_{proved}$
  - compute $f^* = min(f(x_{proved}), f^*)$

- **Critical issue**: to prove the existence of a feasible point in a reduced box
  - Singularities
  - Guess point too far from a feasible region (local search works with floats)
Using the lower bound to get an upper-bound

Branch&Bound step where $P$ is the set of feasible points and $R$ is the linear relaxation

Idea: modify the safe lower bound ... to get an upper-bound!
Lower bound: a good starting point to find a feasible upper-bound?

\[ N, \text{optimal solution of } R, \text{not a feasible point of } P \text{ but (may be) a good starting point:} \]

- BB splits the domains at each iteration:
  smaller box \( \rightsquigarrow N \) nearest from the optima of \( P \)
- Proof process inflates a box around the guess point \( \rightsquigarrow \)
  compensate the distance from the feasible region
Method

- Correction procedure to get a better feasible point from a given approximate feasible point

  → to exploit **Newton-Raphson for under-constrained systems** of equations (and Moore-Penrose inverse)

  **Good convergence** when the starting point is nearly feasible
Handling square systems of equations

- $g = (g_1, \ldots, g_m) : \mathbb{R}^n \rightarrow \mathbb{R}^m \ (n = m)$

  $\rightarrow$ Newton-Raphson step:

  $$x^{(i+1)} = x^{(i)} - J_g^{-1}(x^{(i)})g(x^{(i)})$$

  **Converges well** if the exact solution to be approximated is **not singular**
Handling under-constrained systems of equations

**Manifold of solutions**
linear system \( l(x) = 0 \) is under-constrained
Choose a solution \( x^{(1)} \) of \( l(x) = 0 \)

**Best choice:**
Solution of \( l(x) = 0 \) close to \( x^{(0)} \)
Can easily be computed with the Moore-Penrose inverse:

\[
x^{(i+1)} = x^{(i)} - A_g^+ (x^{(i)}) g(x^{(i)})
\]

\( A_g^+ \in \mathbb{R}^{n \times m} \) is the Moore-Penrose inverse of \( A_g \), solution of the equation which minimizes \( \|x^{(1)} - x^{(0)}\| \)
Handling under-constrained systems of equations and inequalities

- Under-constrained systems of equations and **inequalities**
  - introduce **slack variables**

- **Initial values** for the slack variables have to be provided

  Slightly positive value
  → to break the symmetry
  → good convergence
A new upper bounding strategy

Function UpperBounding(IN x, x^*_LP; INOUT S')

% S': list of proven feasible boxes
% x^*_LP: the optimal solution of the LP relaxation of P(x)
S' := ∅
x^*corr := FeasibilityCorrection(x^*_LP) % Improving x^*_LP feasibility
x_p := InflateAndProve(x^*corr, x)
if x_p ≠ ∅ then
    S' := S' ∪ x_p
endif
return S'
Experiments

- Significant set of benchmarks of the COCONUT project

- Selection of 35 benchmarks where Icos did find the global minimum while relying on an unsafe local search

- 31 benchmarks are solved and proved within a 30s time out

- Almost all benchmarks are solved in much less time and with much more proven solutions
### Experiments (2)

<table>
<thead>
<tr>
<th>Name</th>
<th>(n,m)</th>
<th>LS: t(s)</th>
<th>UB/LB: t(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alkyl</td>
<td>(14, 7)</td>
<td>-</td>
<td>1.54</td>
</tr>
<tr>
<td>circle</td>
<td>(3, 10)</td>
<td>1.98</td>
<td>0.84</td>
</tr>
<tr>
<td>ex14_1_2</td>
<td>(6, 9)</td>
<td>-</td>
<td>1.74</td>
</tr>
<tr>
<td>ex14_1_3</td>
<td>(3, 4)</td>
<td>-</td>
<td>0.42</td>
</tr>
<tr>
<td>ex14_1_6</td>
<td>(9, 15)</td>
<td>-</td>
<td>12.44</td>
</tr>
<tr>
<td>ex14_1_8</td>
<td>(3, 4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ex2_1_1</td>
<td>(5, 1)</td>
<td>0.09</td>
<td>0.04</td>
</tr>
<tr>
<td>ex2_1_2</td>
<td>(6, 2)</td>
<td>-</td>
<td>0.24</td>
</tr>
<tr>
<td>ex2_1_3</td>
<td>(13, 9)</td>
<td>-</td>
<td>1.32</td>
</tr>
<tr>
<td>ex2_1_4</td>
<td>(6, 5)</td>
<td>0.52</td>
<td>0.43</td>
</tr>
<tr>
<td>ex2_1_6</td>
<td>(10, 5)</td>
<td>1.61</td>
<td>0.35</td>
</tr>
<tr>
<td>ex3_1_3</td>
<td>(6, 6)</td>
<td>1.03</td>
<td>0.29</td>
</tr>
<tr>
<td>ex3_1_4</td>
<td>(3, 3)</td>
<td>6.51</td>
<td>0.14</td>
</tr>
<tr>
<td>ex4_1_2</td>
<td>(1, 0)</td>
<td>18.84</td>
<td>17.03</td>
</tr>
<tr>
<td>ex4_1_6</td>
<td>(1, 0)</td>
<td>0.11</td>
<td>14.28</td>
</tr>
<tr>
<td>ex4_1_7</td>
<td>(1, 0)</td>
<td>0.07</td>
<td>0.01</td>
</tr>
<tr>
<td>ex5_4_2</td>
<td>(8, 6)</td>
<td>-</td>
<td>18.15</td>
</tr>
<tr>
<td>ex6_1_2</td>
<td>(4, 3)</td>
<td>0.51</td>
<td>0.52</td>
</tr>
<tr>
<td>ex6_1_4</td>
<td>(6, 4)</td>
<td>7.45</td>
<td>8.92</td>
</tr>
<tr>
<td>ex7_3_5</td>
<td>(13, 15)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ex8_1_6</td>
<td>(2, 0)</td>
<td>-</td>
<td>0.39</td>
</tr>
<tr>
<td>ex9_1_1</td>
<td>(13, 12)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ex9_1_10</td>
<td>(14, 12)</td>
<td>-</td>
<td>3.76</td>
</tr>
<tr>
<td>ex9_1_4</td>
<td>(10, 9)</td>
<td>-</td>
<td>0.49</td>
</tr>
<tr>
<td>ex9_1_5</td>
<td>(13, 12)</td>
<td>-</td>
<td>2.68</td>
</tr>
<tr>
<td>ex9_1_8</td>
<td>(14, 12)</td>
<td>-</td>
<td>3.76</td>
</tr>
<tr>
<td>ex9_2_1</td>
<td>(10, 9)</td>
<td>-</td>
<td>0.68</td>
</tr>
<tr>
<td>ex9_2_4</td>
<td>(8, 7)</td>
<td>2.94</td>
<td>0.69</td>
</tr>
<tr>
<td>ex9_2_5</td>
<td>(8, 7)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ex9_2_7</td>
<td>(10, 9)</td>
<td>-</td>
<td>0.68</td>
</tr>
<tr>
<td>ex9_2_8</td>
<td>(6, 5)</td>
<td>-</td>
<td>0.53</td>
</tr>
<tr>
<td>house</td>
<td>(8, 8)</td>
<td>-</td>
<td>0.90</td>
</tr>
<tr>
<td>nemhaus</td>
<td>(5, 5)</td>
<td>0.02</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Using CSP to boost safe OBR

- **OBR** (optimal based reduction): known bounds of the objective function → to reduce the size of the domains

- **Refutation** techniques → boosting safe OBR
Lower bounding

- Relaxing the problem
  - linear relaxation $R$ of $\mathcal{P}$
    
    \[
    \begin{align*}
    \min & \quad d^T x \\
    \text{s.t.} & \quad Ax \leq b
    \end{align*}
    \]
  - LP solver $\rightarrow f^*$

  $\rightarrow$ numerous splitting

- OBR is a way to speed up the reduction process
Optimality Base Reduction

- Introduced by Ryoo and Sahinidis

- to take advantage of the known bounds of the objective function to reduce the size of the domains

- uses a well known property of the saddle point to compute new bounds for the domains with the known bounds of the objective function
Theorems of OBR

- Let \([L, U]\) be the domain of \(f\):
  - \(U\) is an upper-bound of the initial problem \(\mathcal{P}\)
  - \(L\) is a lower-bound of a convex relaxation \(R\) of \(\mathcal{P}\)

If the constraint \(x_i - \bar{x}_i \leq 0\) is active at the optimal solution of \(R\) and has a corresponding multiplier \(\lambda_i^* > 0\) (\(\lambda^*\) is the optimal solution of the dual of \(R\)), then

\[
x_i \geq x'_i \text{ with } x'_i = \bar{x}_i - \frac{U - L}{\lambda_i^*}
\]

if \(x'_i > x_i\), the domain of \(x_i\) can be shrunked to \([x'_i, \bar{x}_i]\) without loss of any global optima

- similar theorems for \(x_i - x_i \leq 0\) and \(g_i(x) \leq 0\).
OBR: intuitions

- Ryoo & Sahinidis 96

\[ x'_i = x_i - \frac{U - L}{\lambda_i} \]
\[ x'_i = x_i + \frac{U - L}{\lambda_i} \]

\[ x_i \geq x'_i \text{ with } x'_i = \bar{x}_i - \frac{U - L}{\lambda_i^*} \]

- does not modify the very branch and bound process
- almost for free!
OBR Issues

▶ Critical issue: basic OBR algorithm is unsafe
  • it uses the dual solution of the linear relaxation
  • Efficient LP solvers work with floats → the available dual solution $\lambda^*$ is an **approximation** if used in OBR ...
  ... → **OBR may remove actual optimum**!

▶ Solutions: two ways to take advantage of OBR
  1. **prove dual solution** (Kearfott): combining the dual of linear relaxation with the Kuhn-Tucker conditions
  2. **validate the reduction** proposed by OBR with CP!
CP approach: intuition

- **Essential observation:** if the constraint system

\[
L \leq f(x) \leq U \\
g_i(x) = 0, \ i = 1..k \\
g_j(x) \leq 0, \ j = k + 1..m
\]

has no solution when the domain of \(x\) is set to \([x_i, x'_i]\\),
the reduction computed by OBR is valid

- **Try to reject** \([x_i, x'_i]\\) with classical filtering techniques;
otherwise add this box to the list of boxes to process
CP algorithm

\[ L_r := \emptyset \quad \% \text{set of potential non-solution boxes} \]

\textbf{for} each variable \( x_i \) \textbf{do}
   \hspace{1cm} Apply OBR
   \hspace{1cm} and add the generated potential non-solution boxes to \( L_r \)

\textbf{for} each box \( B_i \) in \( L_r \) \textbf{do}
   \hspace{1cm} \( B_i' := 2B\text{-filtering}(B_i) \)
   \hspace{1cm} \textbf{if} \( B_i' = \emptyset \) \textbf{then} reduce the domain of \( x_i \)
   \hspace{1cm} \textbf{else} \( B_i'' := \text{QUAD-filtering}(B_i') \)
   \hspace{1cm} \hspace{1cm} \textbf{if} \( B_i'' = \emptyset \) \textbf{then} reduce the domain of \( x_i \)
   \hspace{1cm} \hspace{1cm} \textbf{else} add \( B_i \) to global list of box to be handled \textbf{endif}
\hspace{1cm} \textbf{endif}

\textbf{Compute} \( f \) with \text{QUAD\_SOLVER} \text{ in } X
Experiments

- Compares 4 versions of the branch and bound algorithm:
  - without OBR
  - with unsafe OBR
  - with safe OBR based on Kearfott’s approach
  - with safe OBR based on CP techniques implemented with Icos using Coin/CLP and Coin/IpOpt

- On 78 benches (from Ryoo & Sahinidis 1995, Audet thesis and the coconut library)

- All experiments have been done on PC-Notebook/1Ghz.
Experimental Results (2): Synthesis

Synthesis of the results:

<table>
<thead>
<tr>
<th></th>
<th>$\Sigma_t(s)$</th>
<th>%saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>no OBR</td>
<td>2384.36</td>
<td>-</td>
</tr>
<tr>
<td>unsafe OBR</td>
<td>881.51</td>
<td>63.03%</td>
</tr>
<tr>
<td>safe OBR Kearfott</td>
<td>1975.95</td>
<td>17.13%</td>
</tr>
<tr>
<td>safe OBR CP</td>
<td>454.73</td>
<td>80.93%</td>
</tr>
</tbody>
</table>

(with a timeout of 500s)

Safe CP-based OBR faster than unsafe OBR!

... because wrong domains reductions prevent the upper-bounding process from improving the current upper bound!!
Handling software upgradeability problems

- A critical issue in modern operating systems
  - Finding the “best” solution to install, remove or upgrade packages in a given installation.
  - The complexity of the upgradeability problem itself is NP complete.
  - Modern OS contain a huge number of packages (often more than 20,000 packages in a Linux distribution).

- Several optimisation criteria have to be considered, e.g., stability, memory efficiency, network efficiency.

Solving software upgradeability problems

Computing a final package configuration from an initial one

- A configuration states which package is installed and which package is not installed:
  - **Problem** (in CUDF): list of package descriptions (with their status) & a set of packages to install/remove/upgrade
  - **Final configuration**: list of installed packages (uninstalled packages are not listed)

- **Expected Answer**: best solution according to multiple criteria
A Problem: list of package descriptions & requests (1)

A package description provides:

- the **package name** and **package version**
  - \( p_{i,j} = (\text{package name } p_i, \text{package version } v_j) \) is unique for each problem in CUDF
  - The \( p_{i,j} \) are basic variables
    → solvers have to instantiate \( p_{i,j} \) with true or false

- Package **dependencies** and **conflicts**: set of constraints between the \( p_{i,j} \) (CNF formula)

- Provided **features**: if package \( p_1 \) depends on feature \( f_\lambda \) provided by \( q_1 \) and \( q_2 \), then installing \( q_1 \) or \( q_2 \) will fulfill \( p_1 \)'s dependency on \( f_\lambda \).
A Problem: list of package descriptions & requests (2)

- **Requests** are:
  - **Commands/actions** on the initial configuration:
    - **install p**: at least one version of p must be installed in the final configuration
    - **remove p**: no version of p must be installed in the final configuration
    - **upgrade p**: let \( p_\nu \) be the highest version installed in the initial configuration, then \( p'_\nu \) with \( \nu' \geq \nu \) must be the only version installed in the final configuration
  - **Mandatory**: the final configuration must fulfill all the requests (otherwise there is no solution to the problem)
  - **Requests** induce **additional constraints** on the problem to solve
Finding the best solution

- **Best solution**
  - multiple criteria, e.g.,
    - minimize the number of **removed** packages, and,
    - minimize the number of **changed** packages

- **Mono criteria optimization solvers**
  - using a **linear combination** of the criteria
  - solving each criteria sequentially
MILP model: handling dependencies

1. **Conjunction**:

   \[
   \text{Depend}(p_v) = \bigwedge_{i=1}^{n} p_i \implies -n \times p_v + \sum_{i=1}^{n} p_i \geq 0
   \]

   if \( p_v = 1 \) (installed), then all \( p_i = 1 \); if \( p_v = 0 \) (not installed), then the \( p_i \) can take any value

2. **Disjunction**

   \[
   \text{Depend}(p_v) = \bigvee_{k=1}^{l_m} p_k \implies -p_v + \sum_{k=1}^{l_m} p_k \geq 0
   \]

   thus, if \( p_v = 1 \), at least one of the \( p_k \) will be installed.
MILP model: handling conflicts

**Conflict property**: a simple conjunction of packages

→ inequality:

\[ n' \times p_v + \sum_{p_c \in Conflict(p_v)} p_c \leq n' \]

where \( Conflict(p_v) \) is the set of package conflicting with \( p_v \)
and \( n' = Card(Conflict(p_v)) \)

→ if \( p_v \) is installed, none of the \( p_v \) conflicting packages can be installed

→ if \( p_v \) is not installed, then the conflicting packages can freely be either installed or not
MILP model: handling multi criteria (1)

Assume the following 2 criteria:

► **First criterion**: minimize the number of removed functionalities among the installed ones

\[
\min \sum_{p \in F_{\text{Installed}}} -p
\]

where \( F_{\text{Installed}} \) is the set of installed functionalities

► **Second criterion**: minimize the number of modifications; if package \( p \), version \( i \) is installed keep it installed, if package \( p \) version \( u \) it is not installed keep it uninstalled

\[
\min \sum_{p_i \in P_{\text{Installed}}} -p_i + \sum_{p_u \in P_{\text{Uninstalled}}} p_u
\]

where \( P_{\text{Installed}} \) is the set of installed versioned packages and \( P_{\text{Uninstalled}} \) is the set of uninstalled versioned packages.
MILP model: handling multi criteria (2)

Handling these criteria in a lexical order

→ **criteria are aggregated** in the following way:

\[
\sum_{p \in F_{\text{Installed}}} \neg \text{Card}(P) \cdot p + \sum_{p_i \in P_{\text{Installed}}} -p_i + \sum_{p_u \in P_{\text{Uninstalled}}} p_u
\]

where \( P = P_{\text{Installed}} \cup P_{\text{Uninstalled}} \)

**Multiplying first criterion coefficients by \( \text{Card}(P) \)** lets any of them have a higher value than any combination of the second criterion.
Experiments

- A set of **200 problems**, ranging from random problems to real one and from **20000 up to 50000 packages**

- **MILP solvers & Pseudo boolean solvers**

<table>
<thead>
<tr>
<th></th>
<th>IBM CPLEX 11.1</th>
<th>SCIP 1.2</th>
<th>WBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time out</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>No sol</td>
<td>58</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>Min time (s)</td>
<td>0.54</td>
<td>0.54</td>
<td>0.53</td>
</tr>
<tr>
<td>Max time (s)</td>
<td>7.83</td>
<td>193.73</td>
<td>300</td>
</tr>
<tr>
<td>Geometric Mean time (s)</td>
<td>2.5</td>
<td>10.29</td>
<td>23.6</td>
</tr>
</tbody>
</table>

- **IBM CP**: could not find any solution within 300s
Examples of optimization criteria (ongoing solver competition)

- **paranoid**:
  minimizing the packages removed in the solution
  &
  minimizing packages changed by the solution

- **trendy**:
  minimizing packages removed in the solution
  &
  minimizing outdated packages in the solution
  &
  minimizing package recommendations not satisfied
  &
  minimizing extra packages installed.
Open questions

- How to boost CP?
  - Taking advantage of the dependency graph
  - Combining CP and MILP

- Better handling of preferences?
Conclusion

+ **CSP refutation techniques**
  - allow a *safe* and *efficient* implementation of OBR
  - can *outperform standard mathematical methods*
  - might be suitable for other unsafe methods

+ **Safe global constraints**
  - provide an efficient alternative to local search:
    - good starting point for a Newton method $\leadsto$ feasible region
  - *drastically improve the performances* of the upper-bounding process

? **CP and Robustness**

? **Large finite-domain optimization problems**