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

Michel Rueher

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

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

with

- $\mathbf{X} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$: 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 $\mathbf{X}$

Trends in global optimisation

- **Performance**

  Most successful systems (Baron, αBB, ...) use local methods and linear relaxations
  $\rightarrow$ **not rigorous** (work with floats)

- **Rigour**

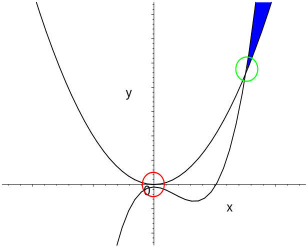
  Mainly rely on interval computation
  ... available systems (e.g., Globsol) are **quite slow**

- **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:

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



Baron 6.0 and Baron 7.2 find $\color{red}{0}$ 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 $\underline{\mathbf{f}}$ and $\overline{\mathbf{f}}$
  - 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
- $\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ is a set of domains
  ($\mathbf{X}_i$ contains all acceptable values for variable x_i)

  $\mathbf{X}_i = [\underline{\mathbf{x}_i}, \overline{\mathbf{x}_i}]$
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ 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 $\mathbf{X} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ is 2B–consistent for constraint $f(x, x_1, \ldots, x_n) = 0$ if $\underline{\mathbf{x}}$ and $\overline{\mathbf{x}}$ 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 $\mathbf{X} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ is Box–Consistent for constraint $f(x, x_1, \ldots, x_n) = 0$ if $\underline{\mathbf{x}}$ and $\overline{\mathbf{x}}$ are the leftmost and the rightmost zero of $\mathbf{F}(\mathbf{x}, \mathbf{X_1}, \ldots, \mathbf{X_n})$, the optimal interval extension of $f(x, x_1, \ldots, x_n)$

Filtering

- **2B–filtering Algorithms** $\rightsquigarrow$ **projection functions**
- **Box–filtering Algorithms** $\rightsquigarrow$ **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 $X = [0, 5]$
  $\mathbf{X} - \mathbf{X} = [\mathbf{0} - \mathbf{5}, \mathbf{5} - \mathbf{0}] =$ **[-5,5]** instead of **[0,0] !**
  $\mathbf{X^2} - \mathbf{X} = [\mathbf{0}, \mathbf{25}] - [\mathbf{0}, \mathbf{5}] =$ **[-5,25]**
  $\mathbf{X}(\mathbf{X} - \mathbf{1}) = [\mathbf{0}, \mathbf{5}]([\mathbf{0}, \mathbf{5}] - [\mathbf{1}, \mathbf{1}])$
  $= [\mathbf{0}, \mathbf{5}][-\mathbf{1}, \mathbf{4}] =$ **[-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)
  $\longrightarrow$ capture some dependencies between constraints
  $\longrightarrow$ visiting numerous combinations

$\Rightarrow$ A **global constraint** to handle a **linear approximation** with LP solvers
$\longrightarrow$ safe linear relaxations

A Global Constraint for Safe Linear Relaxation

- works on **quadratic terms and bilinear terms**

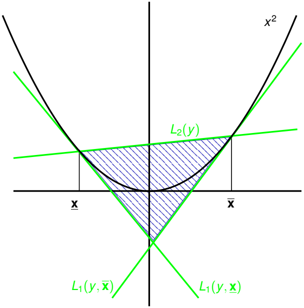
  $\longrightarrow$ to rewrite power terms and product terms

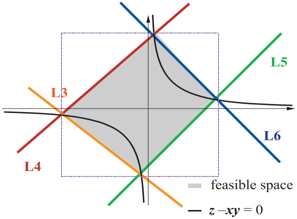
  - **quadrification technique** derived from Sheraldi techniques
  - **Critical issue:** to find a good trade off between a tight relaxation and the number of generated terms

- Quadratic terms and bilinear terms are approximated by tight redundant constraints

The QUAD process

- **Reformulation**
  - capture the linear part
    $\longrightarrow$ replace non linear terms by new variable
    eg x^2 by y_i
- **Linearisation**
  - introduce **redundant linear constraints**
    $\longrightarrow$ tight approximations (RLT)
- **Computing min(X) $= \underline{\mathbf{x}}_i$** and **max(X) $= \overline{\mathbf{x}}_i$** in *LP*





Reformulation for x^2

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

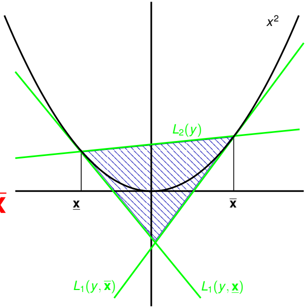
$\mathbf{L_1}(\mathbf{y}, \alpha) \equiv \mathbf{y} \geq \mathbf{2}\alpha\mathbf{x} - \alpha^{\mathbf{2}}$

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

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

$\mathbf{L_2}(\mathbf{y}) \equiv \mathbf{y} \leq (\underline{\mathbf{x}} + \overline{\mathbf{x}})\mathbf{x} - \underline{\mathbf{x}} * \overline{\mathbf{x}}$

$\mathbf{L_2}(\mathbf{y}) : \mathbf{y} \leq \mathbf{x} + \mathbf{20}$



Quad filtering algorithm

Function Quad_filtering (`IN`: $\mathbf{X}$, $\mathcal{C}$, ϵ) `return` $\mathbf{X}'$

1. ***Reformulation***
   $\rightarrow$ linear inequalities L_i for the nonlinear terms in $\mathcal{C}$
2. ***Linearisation/relaxation of the whole system***
   $\rightarrow$ a linear system LR
3. $\mathbf{X}'$:= $\mathbf{X}'$
4. ***Pruning***:
   While reduction of some bound $> \epsilon$ **and** $\emptyset \notin \mathbf{X}'$ **Do**
   4.1 Reduce the lower and upper bounds $\underline{\mathbf{x}}'_i$ and $\overline{\mathbf{x}}'_i$ of each ***initial*** variable $x_i \in \mathcal{X}$
   $\rightarrow$ Computing min and max of $\mathbf{X_i}$ with a LP solver
   4.2 Update the coefficients of L_i according to the new bounds

Issues in the use of linear relaxation

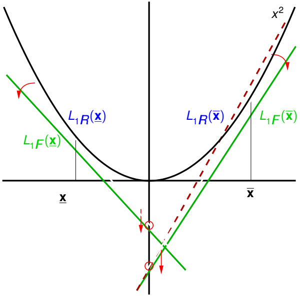
- Coefficients of linear relaxations are scalars
  $\Rightarrow$ computed with ***floating point numbers***
- Efficient implementations of the simplex algorithm
  $\Rightarrow$ use ***floating point numbers***
- All the computations with floating point numbers require ***right corrections***

Safe approximations of L_1

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

Effects of rounding:

- rounding of 2α
  - $\Rightarrow$ rotation on y axis
- rounding of α^2
  - $\Rightarrow$ translation on y axis



Correction of the Simplex algorithm

Consider the following LP :

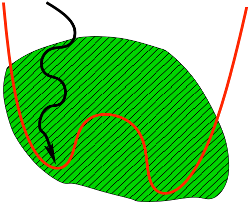
$$\text{minimise } c^T x$$

$$\text{subject to } \underline{\mathbf{b}} \leq Ax \leq \overline{\mathbf{b}}$$

- Solution $=$ vector $x_R \in R^n$
- LP solver computes a vector $x_F \in F^n \neq x_R$
- x_F is safe for the objective if $c^T x_R \geq c^T x_F$
- **Neumaier & Shcherbina**
  $\rightarrow$ cheap method to obtain a **rigorous bound** of the objective
  (use of the approximation solution of the dual)

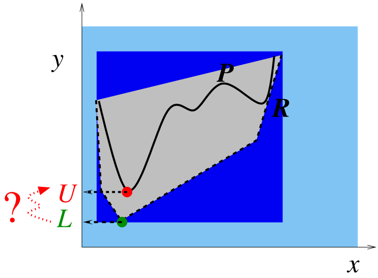
Computing “sharp” upper bounds

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



- **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)

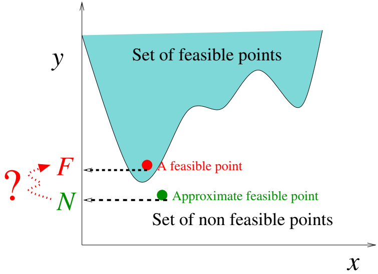
Using the lower bound to get an upper-bound



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, optimal solution of R, not a feasible point of P 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

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

Good convergence when the starting point is nearly feasible

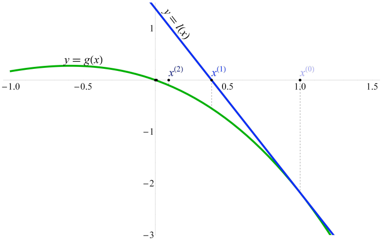
Handling square systems of equations

- $g = (g_1, \ldots, g_m) : R^n \longrightarrow R^m$ **($n = m$)**

  $\longrightarrow$ 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**



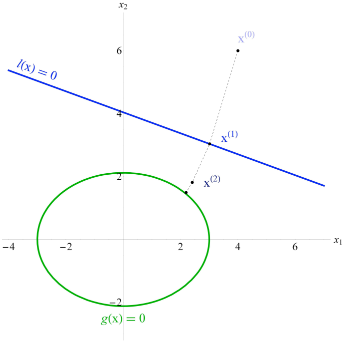
Handling under-constrained systems of equations

Manifold of solutions
$\rightarrow$ linear system $l(x) = 0$ is under-constrained
$\rightarrow$ **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**:

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

$A_g^+ \in R^{n \times m}$ is the Moore-Penrose inverse of A_g, solution of the equation which minimizes $||x^{(1)} - x^{(0)}||$)

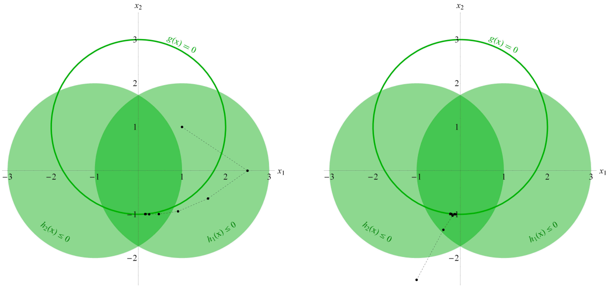


Handling under-constrained systems of equations and inequalities

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

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

  Slightly positive value
  $\longrightarrow$ to break the symmetry
  $\longrightarrow$ good convergence



A new upper bounding strategie

Function `UpperBounding`(`IN` $\mathbf{x}$, x^*_{LP}; `INOUT`$\mathcal{S}'$)

% $\mathcal{S}'$: list of proven feasible boxes
% x^*_{LP}: the optimal solution of the LP relaxation of $\mathcal{P}(\mathbf{x})$
$\mathcal{S}'$:= $\emptyset$
x^*_{corr} := FeasibilityCorrection(x^*_{LP}) % Improving x^*_{LP} feasibility
$\mathbf{x}_p$:= InflateAndProve(x^*_{corr}, $\mathbf{x}$)
if $\mathbf{x}_p \neq \emptyset$ **then**
    $\mathcal{S}'$:= $\mathcal{S}' \cup \mathbf{x}_p$
endif
return $\mathcal{S}'$

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)

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

Using CSP to boost safe OBR

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

- **Refutation** techniques $\rightarrow$ **boosting safe OBR**

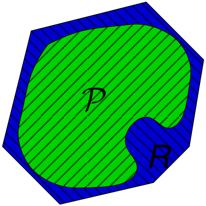
Lower bounding

- **Relaxing the problem**
  - linear relaxation R of $\mathcal{P}$

$$\begin{array}{ll} min & d^T x \\ s.t. & Ax \leq b \end{array}$$

  - LP solver $\rightarrow \underline{\mathbf{f}}^*$

$\rightarrow$ 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 intial problem $\mathcal{P}$**
  - L is a **lower-bound of a convex relaxation R of $\mathcal{P}$**

  If the constraint $\mathbf{x_i} - \overline{\mathbf{x}}_\mathbf{i} \leq \mathbf{0}$ is **active** at the optimal solution of R and has a corresponding multiplier $\lambda_i^* > 0$ (λ^* is the optimal solution of the dual of R), then

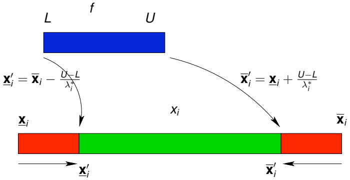
$$\mathbf{x_i} \geq \underline{\mathbf{x}}'_\mathbf{i} \text{ with } \underline{\mathbf{x}}'_\mathbf{i} = \overline{\mathbf{x}}_\mathbf{i} - \frac{\mathbf{U} - \mathbf{L}}{\lambda_\mathbf{i}^*}$$

  if $\underline{\mathbf{x}}'_i > \underline{\mathbf{x}}_i$, the domain of x_i can be shrinked to $[\underline{\mathbf{x}}'_i, \overline{\mathbf{x}}_i]$ **without loss of any global optima**

- similar theorems for $\underline{\mathbf{x}}_i - x_i \leq 0$ and $g_i(x) \leq 0$.

OBR: intuitions

- Ryoo & Sahinidis 96



$$x_i \geq \underline{\mathbf{x}}_i' \text{ with } \underline{\mathbf{x}}_i' = \overline{\mathbf{x}}_i - \frac{U-L}{\lambda_i^*}$$

- **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 $\longrightarrow$ the available dual solution λ^* is an **approximation** if used in OBR ...
    ... $\longrightarrow$ **OBR may remove actual optimum !**

- **Solutions:** two ways to take advantage of OBR
  1. **prove dual solution** (Kearfott): combininig 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**

$$\begin{array}{l} L \leq f(x) \leq U \\ g_i(x) = 0, \; i = 1..k \\ g_j(x) \leq 0, \; j = k+1..m \end{array}$$

  has no solution when the domain of x is set to $[\underline{\mathbf{x}}_i, \underline{\mathbf{x}}'_i]$, **the reduction computed by OBR is valid**

- **Try to reject $[\underline{\mathbf{x}}_i, \underline{\mathbf{x}}'_i]$** with **classical filtering techniques;**
  otherwise add this box to the list of boxes to process

CP algorithm

$\mathcal{L}_r$:= $\emptyset$ % set of potential non-solution boxes

for each variable x_i **do**
  Apply OBR
    and add the generated potential non-solution boxes to $\mathcal{L}_r$

for each box $\mathbf{B_i}$ in $\mathcal{L}_r$ **do**
  $\mathbf{B_i'}$:= **2B-filtering**($\mathbf{B_i}$)
  if $\mathbf{B_i'} = \emptyset$ **then** reduce the domain of x_i
  else $\mathbf{B_i''}$:= **QUAD-filtering**($\mathbf{B_i'}$)
      if $\mathbf{B_i''} = \emptyset$ **then** reduce the domain of x_i
      else add $\mathbf{B_i}$ to global list of box to be handled **endif**
  endif

Compute $\underline{f}$ with QUAD_SOLVER 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:

	$\Sigma_t(s)$	$\%saving$
no OBR	2384.36	-
unsafe OBR	881.51	63.03%
safe OBR Kearfott	1975.95	17.13%
safe OBR CP	**454.73**	**80.93%**

(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 !!

Finite domains CSP & Global Optimisation

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
- **Mancoosi** project (FP7/2007-2013, `http://www.mancoosi.org/`)

Solving software upgradeability problems

Computing a final package configuration from an intial 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**
  - $p_{i,j}$ = (package name p_i, package version v_j) is unique for each problem in CUDF
  - The $p_{i,j}$ are basic variables
    $\rightarrow$ **solvers have to instantiate $p_{i,j}$ with true or false**
- Package **dependencies** and **conflicts**: set of contraints between the $p_{i,j}$ (CNF formula)
- Provided **features:** if package p_1 depends on feature f_λ provided by q_1 and q_2, then installing q_1 or q_2 will fulfill p_1's dependency on f_λ.

A Problem: list of package descriptions & requests (2)

- **Requests** are:
  - **Commands/actions** on the initial configuration: install, remove and/or upgrade package instructions
    - **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_v be the highest version installed in the initial configuration, then p'_v with $v' \geq v$ 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**:

$$\mathcal{D}epend(p_v) = \bigwedge_{i=1}^{n} p_i \;\rightsquigarrow\; \mathbf{-n * p_v} + \sum_{i=1}^{n} p_i >= 0$$

   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**

$$\mathcal{D}epend(p_v) = \bigvee_{k=1}^{l_m} p_k \;\rightsquigarrow\; \mathbf{-p_v} + \sum_{k=1}^{l_m} p_k >= 0$$

   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
$\rightarrow$ inequality:

$$\mathbf{n'} * \mathbf{p_v} + \sum_{p_c \in \mathcal{C}onflict(p_v)} p_c <= \mathbf{n'}$$

where $\mathcal{C}onflict(p_v)$ is the set of package conflicting with p_v and $n' = \mathcal{C}ard(\mathcal{C}onflict(p_v))$

- $\rightarrow$ if p_v is installed, none of the p_v conflicting packages can be installed
- $\rightarrow$ 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

$$min \sum_{p \in F_{\mathcal{I}nstalled}} -p$$

where $F_{\mathcal{I}nstalled}$ 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

$$min \sum_{p_i \in P_{\mathcal{I}nstalled}} -p_i + \sum_{p_u \in P_{\mathcal{U}ninstalled}} p_u$$

where $P_{\mathcal{I}nstalled}$ is the set of installed versioned packages and $P_{\mathcal{U}ninstalled}$ 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:

$$\sum_{p \in F_{\mathcal{I}nstalled}} -\mathcal{C}ard(P) * p \;+ \sum_{p_i \in P_{\mathcal{I}nstalled}} -p_i \;+ \sum_{p_u \in P_{\mathcal{U}ninstalled}} p_u$$

where $P = P_{\mathcal{I}nstalled} \cup P_{\mathcal{U}ninstalled}$

Multiplying first criterion coefficients by $\mathcal{C}ard(P)$ 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**

	IBM CPLEX 11.1	SCIP 1.2	WBO
Time out	0	0	1
No sol	58	58	58
Min time (s)	0.54	0.54	0.53
Max time (s)	7.83	193.73	300
Geometric Mean time (s)	**2.5**	**10.29**	**23.6**

- 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: $\longrightarrow$ good starting point for a Newton method $\rightsquigarrow$ feasible region
- **drastically improve the performances** of the upper-bounding process

? CP and Robustness

? Large finite-domain optimization problems